# Hobo - Bug #24099

## mutlitenant: optimisations accès à une base-de-donnée répliquée

26 mai 2018 16:58 - Christophe Siraut

| | | | |
|---|---|---|---|
| **Statut:** | Rejeté | **Début:** | 26 mai 2018 |
| **Priorité:** | Normal | **Echéance:** | |
| **Assigné à:** | | **% réalisé:** | 0% |
| **Catégorie:** | | **Temps estimé:** | 0:00 heure |
| **Version cible:** | | | |
| **Patch proposed:** | Oui | **Planning:** | |

**Description**

Afin d'améliorer les performances de l'hébergement mutualisé, proposition d'adapter hobo-multitenant/django-tenant-schemas pour supporter plusieurs bases-de-données [1].

https://docs.djangoproject.com/en/2.0/topics/db/multi-db/

Actuellement le code de ces modules utilise intensément django.db.connection lequel renvoie à connections[DEFAULT_DB_ALIAS]

La proposition est de continuer est de continuer à utiliser DATABASES['default'] pour la connexion à la db primaire pour les opérations normales, et de permettre un routage personnalisé vers des connexions additionnelles (par exemple pour les requêtes web en lecture seule), cela nécessite une adaptation minimale du middleware, voir patch attaché. (Le cas d'usage est exclusivement des bases-de-données répliquées, ne fonctionnerait pas pour des tenants dispersés sur plusieurs db)

### Historique

**#1 - 26 mai 2018 17:00 - Christophe Siraut**

*- Fichier 0001-mutlitenant-support-inital-de-base-de-donn-es-mutlip.patch ajouté*

*- Patch proposed changé de Non à Oui*

**#2 - 26 mai 2018 17:15 - Christophe Siraut**

*- Fichier 0002-multitenant-add-ClusterRouter-which-separate-reads-a.patch ajouté*

un patch pour fournir un router multidb générique.

**#3 - 26 mai 2018 17:18 - Christophe Siraut**

*- Fichier 0002-multitenant-add-ClusterRouter-which-separate-reads-a.patch supprimé*

**#4 - 26 mai 2018 17:18 - Christophe Siraut**

*- Fichier 0002-multitenant-add-ClusterRouter-which-separate-reads-a.patch ajouté*

**#5 - 28 mai 2018 13:02 - Benjamin Dauvergne**

Ça ne marchera pas avec les transactions (les lectures iront sur la réplique et les écritures sur le maître, il manquera les verrous sur les lignes lues de la réplique), ce n'est donc pas possible de faire comme cela, sans devoir repasser sur tout le code qui utiliserait une transaction.

Pour de la répartition il vaut mieux regarder du coté de pgpool qui répartira les requêtes en lecture en dehors des transactions sur les esclaves et sinon le reste sur le maître (et au moins ça ne complexifie par le code Django).

**#6 - 28 mai 2018 15:08 - Christophe Siraut**

Benjamin: autre idée, surcharger get_queryset() en ajoutant .using(db=...) si la requête est de type GET.

Commencé à lire à doc de pgpool.

**#7 - 28 mai 2018 15:55 - Thomas Noël**

Christophe Siraut a écrit :

> Benjamin: autre idée, surcharger get_queryset() en ajoutant .using(db=...) si la requête est de type GET.

Je pense que c'est mort, car comme le dit Benjamin, ce qui compte c'est la notion de transaction. Les requêtes ne sont pas indépendantes les unes

des autres, mais groupées "dans un lot", une transaction.

> Commencé à lire à doc de pgpool.

Dans quel but ? De toute façon on est obligé d'envoyer au master (du moins c'est ma lecture).

#### #8 - 28 mai 2018 15:58 - Frédéric Péters

Parce que Benjamin écrit « [pgpool] répartira les requêtes en lecture en dehors des transactions sur les esclaves ».

#### #9 - 28 mai 2018 16:03 - Thomas Noël

Nickel donc ; et demain j'apprends à lire.

#### #10 - 28 mai 2018 17:12 - Christophe Siraut

> Parce que Benjamin écrit « [pgpool] répartira les requêtes en lecture en dehors des transactions sur les esclaves ».

Voir le parsing intellignet dans le § 5.7.2. Load Balancing in Streaming Replication:

http://www.pgpool.net/docs/latest/en/html/runtime-config-load-balancing.html

et l'option backend_weight.

#### #11 - 29 mai 2018 10:06 - Benjamin Dauvergne

Apparemment même dans une transaction il ne commence à envoyer au master que dès qu'il y a une écriture et avant il rejoue les lectures, je ne sais pas trop à quel point c'est correct de faire cela, je pense qu'ils y ont pensé plus que moi mais ça reste étrange (il y aura toujours une race condition entre le begin sur le slave et sur le master, ils ne verront pas le même état de la base et donc les premiers select émis ne renverront pas la même chose que les deuxièmes ré-émis, voilà c'est juste zarb).

#### #12 - 29 mai 2018 12:26 - Benjamin Dauvergne

Et je me répond à moi même, ça vient du fait que le mode sérialisation par défaut est "read comitted" qui permet les lectures fantômes, i.e. le résultat d'un select peut varier pendant une transaction (la raison pour laquelle get_or_update() ne marche que si les critères de recherche recouvre un index d'unicité). Ils précisent justement que si le niveau est serializable (et depuis la version 9.5, avec le niveau repeatable read aussi je suppose car les phantom read sont devenus interdites aussi à ce niveau), alors on ne peut jouer la transaction que sur le maître.

#### #13 - 30 mai 2018 11:51 - Christophe Siraut

Discussions avec Thomas et Emmanuel: comment pgpool pourrait anticiper qu'une transaction sera en lecture seule ou en écriture?

Tenté de configurer pgpool en mode streaming replication, très bons résultats en lecture (qui nous pointent d'activer la persistance des connexions dans django via CONN_MAX_AGE), par contre quand j'essaie de me logguer à l'admin, je reçois "cannot execute INSERT in a read-only transaction"
Exception Location: /usr/lib/python2.7/dist-packages/django/db/backends/utils.py in execute, line 64

Autres infos dans #24007.

#### #14 - 30 mai 2018 14:54 - Christophe Siraut

En persévérant pgpool semble fonctionner pour nous, mais actuellement l'affichage d'une bête page combo est encore plus lent (1,5x) que de s'adresser directement à la db primaire, (pgpool ouvre plusieurs connexions concurrentes tant qu'il ne sait pas à qui la transaction doit s'adresser)
Probablement que cet affichage génère une session anonyme ou quelque chose comme ça qui peut être désactiver (à suivre)

Je dois encore lire au sujet du mode de sérialisation read commited. (https://docs.postgresql.fr/9.6/transaction-iso.html)

#### #15 - 30 mai 2018 15:45 - Thomas Noël

Discuté avec Christophe : regarder quelle est l'écriture qui fait que la transaction passe sur le master. S'il s'agit juste de la gestion des sessions, voir si on pourrait passer sur un redis qui serait plus efficace. Redis ou autre : on n'a pas besoin d'un niveau de sécurité aussi fort que le postgresql pour les sessions, donc on pourrait chercher un moyen plus rapide, il faut cependant qu'il soit partagé entre les 3 machines.

#### #16 - 01 juin 2018 15:11 - Christophe Siraut

*- Fichier fredcooklog-une-autre-page.sql ajouté*

ajouté le détail des requêtes SQL générés par /une-autre-page (page de texte de combo anonyme).

en comparant avec la liste des requêtes qui requièrent la db primaire (§5.7.2 de http://www.pgpool.net/docs/latest/en/html/runtime-config-load-balancing.html), il y a 3 occurrences de 'SHOW default_transaction_isolation'

#### #17 - 01 juin 2018 15:27 - Benjamin Dauvergne

Mais c'est les requêtes au niveau de pgpool ou de postgres ? Parce que pgpool lui même interroge la DB pour savoir qu'elle est le niveau de transaction par défaut (si par défaut c'est serializable, alors tout part toujours vers le maître).

Concernant l'écriture des sessions, normalement c'est fait par le middleware après que toute transaction dans une vue soit commité, i.e. donc en mode autocommit qui est le mode par défaut en dehors de toute transaction. Il me semble que pgpool est capable de router les premières requêtes vers la db secondaire et la dernière d'écriture de la session vers la db primaire, on ne devrait avoir la latence de db primaire que sur une seule requête ce qui est acceptable.

Est-ce qu'au niveau de pgpool on peut logger toutes les commandes SQL ainsi que le backend sur lequel c'est envoyé ?

#### #18 - 01 juin 2018 15:34 - Benjamin Dauvergne

Christophe Siraut a écrit :

> Discussions avec Thomas et Emmanuel: comment pgpool pourrait anticiper qu'une transaction sera en lecture seule ou en écriture?

Deux choses:

- par défaut on fait de l'autocommit[1] et donc il n'a pas besoin d'anticiper quoique ce soit, chaque commande est enfermée dans une mini-transaction.
- quand c'est vraiment nécessaire (transaction en lecture seul, pour avoir une cohérence), il est possible de faire un SET TRANSACTION READ ONLY[1].

[1] https://docs.djangoproject.com/fr/2.0/topics/db/transactions/#django-s-default-transaction-behavior

[2] https://www.postgresql.org/docs/9.1/static/sql-set-transaction.html

#### #19 - 01 juin 2018 20:02 - Christophe Siraut

J'ai continué à tester pgpool2 cette aprèm, jusqu'à tenter un backport de buster.

Quand on renseigne un seul backend à pgpool, celui de la db locale, le temps de chargement de la page de test combo est de 130ms.

Quand on renseigne un second backend à pgpool, celui de la db primaire, il route correctement la centaine de requêtes de la page de test vers la réplique locale, mais le temps le temps de chargement est toujours naze (j'arrive à optimiser un peu en chipotant mais ce n'est pas suffisant, 330ms).

L'explication: pour chacune de la centaine de transaction il initie des connections/checkpoints des 2 côtés, afin de pouvoir demander un phantom read quand c'est nécessaire.

TLDR; pgpool2 ça marche bien en l'état pour charger des pages en lecture depuis le replica + écrire sur le master (!); par contre ça n'accélère pas le chargement d'une page combo de 150 transactions.

#### #20 - 01 juin 2018 20:22 - Christophe Siraut

> Mais c'est les requêtes au niveau de pgpool ou de postgres ? Parce que pgpool lui même interroge la DB pour savoir qu'elle est le niveau de transaction par défaut (si par défaut c'est serializable, alors tout part toujours vers le maître).

C'étaient les requêtes au niveau de postgresql. Entre temps le routage pgpool est fonctionnel (déso je n'avais pas vu ton message plus tôt.

> …
> Est-ce qu'au niveau de pgpool on peut logger toutes les commandes SQL ainsi que le backend sur lequel c'est envoyé ?

Oui c'est possible pour la première partie de la question, pour la seconde j'utilise la commande suivante (interceptée par pgpool2):

```
psql -h 10.0.0.10 -U pgpool -p5433 -dpostgres -c "show pool_nodes;"
 node_id |  hostname  | port | status | lb_weight |  role   | select_cnt | load_balance_node | replication_del
ay
---------+------------+------+--------+-----------+---------+------------+-------------------+----------------
---
 0       | 172.22.0.1 | 5432 | up     | 0.010000  | primary | 176        | false             | 0
 1       | 127.0.0.1  | 5432 | up     | 0.990000  | standby | 5868       | true              | 0
```

Et en effet comme tu l'anticipais, les requêtes d'une page combo anonyme ne génère aucune requête sur la dbprimaire, les requêtes d'une authentification à l'admin django génère des requêtes balancées sur les 2 backends.

#### #21 - 01 juin 2018 20:28 - Christophe Siraut

> L'explication: pour chacune de la centaine de transaction il initie des connections/checkpoints des 2 côtés, afin de pouvoir demander un phantom read quand c'est nécessaire.

C'est très mal exprimé: un phantom read c'est justement ce qu'on ne veut pas et qui est interdit, ce serait un résultat différent pour une requête identique. Replacez "phantom read" dans la phrase ci-dessus par 'les résultats de la requête'.

#### #22 - 01 juin 2018 21:12 - Benjamin Dauvergne

Je me demande si le fait d'enfermer toutes les requêtes dans une seule transaction n'accélérerait pas les choses. Pourrais-tu tenter de mettre un

```
DATABASES['default']['ATOMIC_REQUESTS'] = True
```

#### #23 - 01 juin 2018 21:17 - Benjamin Dauvergne

Christophe Siraut a écrit :

> [...]
>
> Et en effet comme tu l'anticipais, les requêtes d'une page combo anonyme ne génère aucune requête sur la dbprimaire, les requêtes d'une authentification à l'admin django génère des requêtes balancées sur les 2 backends.

Mais je ne comprends pas pourquoi on obtient avec le load balacing la même latence que directement sur le primaire seul (les 330ms dont tu parles plus haut), le nombre de select listés ne semble pas indiqué que chaque SELECT soit reproduit sur le primaire (et puis ce serait contraire à l'idée du load balancing). Est-ce que tu pourrais expliciter ce que tu veux dire par cette phrase :

> L'explication: pour chacune de la centaine de transaction il initie des connections/checkpoints des 2 côtés, afin de pouvoir demander un phantom read quand c'est nécessaire.

Tu veux dire que même si ce n'est pas nécessaire il crée une nouvelle connection pour chaque requête SQL ? Je trouve ça doublement bizarre, un parce que je ne vois pas à quoi ça sert et deuxième parce que pgpool est d'abord un pooler de connection, ça veut dire qu'il ne devrait pas ouvrir de nouvelles connections mais les réutiliser.

#### #24 - 01 juin 2018 21:35 - Benjamin Dauvergne

Autre idée, mettre backend_weight1 = 999 pour forcer pgpool à préférer le slave au master en général.

#### #25 - 01 juin 2018 21:44 - Benjamin Dauvergne

> To which node the load balancing mechanism sends read queries is decided at the session start time and will not be changed until the session ends. The only exception is by writing special SQL comments. See below for more details.

Donc en gros de temps en temps sans un backend_weight0 = 0 on va taper en lecture sur le noeud primaire distant, et uniquement sur lui durant une même session.

#### #26 - 01 juin 2018 22:41 - Benjamin Dauvergne

Bon après le fait de faire 150 requête pour afficher une page de CMS pour moi c'est problématique (j'ai eu une bonne expérience de django-cachalot de mon coté, c'est une couche de cache au dessus de l'ORM de Django, il suffit d'exclure les objets un peu dynamique, User, BasketItem, etc..).

#### #27 - 02 juin 2018 08:28 - Frédéric Péters

Dans combo, j'ai créé #24237 et de lui #24238 et #24239.

#### #28 - 02 juin 2018 12:33 - Christophe Siraut

> Je me demande si le fait d'enfermer toutes les requêtes dans une seule transaction n'accélérerait pas les choses. Pourrais-tu tenter de mettre un [...]

Quand je met "['ATOMIC_REQUESTS'] = True" les requêtes sont routées vers la db primaire (quand j'enlève cette option les requêtes aboutissent au réplica)

> Est-ce que tu pourrais expliciter ce que tu veux dire par cette phrase […]

en effet ce n'est pas clair, besoin d'analyser d'avantage, à suivre

> il ne devrait pas ouvrir de nouvelles connections mais les réutiliser.

je suis bien d'accord avec toi: les connexions restent ouvertes.

> Autre idée, mettre backend_weight1 = 999 pour forcer pgpool à préférer le slave au master en général.

yep c'est déjà ce que nous avions, et la commande "show pool_nodes;" rapporte que le load_balancer route tout vers le réplica, néanmoins c'est 1,5x plus lent que quand on supprime backend0 (=db primaire) de la config de pgpool.

**#29 - 04 juin 2018 10:27 - Christophe Siraut**

*- Fichier on_primary ajouté*

*- Fichier on_replica ajouté*

*- Fichier pglog ajouté*

> à suivre

J'attache les log de pgpool, postgresql_primary et postgresql_replica pour un appel à notre page test (backend0=primary, backend1=replica).

1ère observation: il y a au moins le 'SET search_path = combo_fredcook_dev_entrouvert_org,public' qui est joué des 2 cotés à chaque transaction.

**#30 - 04 juin 2018 12:30 - Benjamin Dauvergne**

Putain mais c'est ça, c'est le SET search_path qui fait qu'on a une au moins une requête qui part à chaque fois vers le primary même en autocommit.

La façon de faire de django-multitenant-schemas est effectivement violente, il pourrait se contenter de n'envoyer que SET search_path qu'un fois en début de requête mais le fais à chaque fois "au cas où", on pourrait travailler sur cela.

**#31 - 04 juin 2018 15:34 - Christophe Siraut**

*- Fichier 0001-do-not-reset-tenant-schema-when-already-set.patch ajouté*

*- Sujet changé de mutlitenant: support inital de base-de-données mutliples à mutlitenant: optimisations accès à une base-de-donnée répliquée*

Le patch attaché permet d'éviter les "SET search_path" inutiles.

**#33 - 04 juin 2018 15:36 - Benjamin Dauvergne**

On avait déjà un fork non ?

**#34 - 04 juin 2018 15:37 - Benjamin Dauvergne**

Christophe Siraut a écrit :

> Le patch attaché permet d'éviter les "SET search_path" inutiles.

Et donc tu sens un changement au niveau des perfs ?

**#35 - 04 juin 2018 15:48 - Christophe Siraut**

*- Fichier primary ajouté*

> Et donc tu sens un changement au niveau des perfs ?

Oui c'est déjà mieux, il reste encore les requêtes suivantes qui continuent d'aboutir à la db primaire, je ne sais pas pourquoi (voir doc attaché).

(et middleware.py fait actuellement 3 connexions avec "SET search_path" en dur)

**#37 - 04 juin 2018 15:54 - Christophe Siraut**

> il reste encore les requêtes suivantes qui continuent d'aboutir à la db primaire

La branche de Fred wip/24237-sql-reduction s'attaque à celles-ci.

#### #38 - 04 juin 2018 15:54 - Benjamin Dauvergne

A priori les requêtes aux métadonnées postgres ne viennent pas de Django, je regarde le code de Django et je ne vois rien qui ressemble à ça, donc ça viendrait plutôt de pgpool (et ça se verra certainement dans les logs de pgpool). Avec un peu de chance elles ne sont pas systématique (une fois qu'il a une vision des objets dans la base possible qu'il ne requête plus).

#### #39 - 04 juin 2018 16:06 - Christophe Siraut

Activé le cache de pgpool (cache de type "shared memory", memcached est possible et probablement plus performant), les perfs commencent à être très bonnes (130ms pour notre page de test, bingo), voici les requêtes vers la db primaire qui subsistent:

```
2018-06-04 14:03:08.101 UTC [1814167] combo@combo LOG:  statement: SELECT current_setting('transaction_isolati
on')
2018-06-04 14:03:08.104 UTC [1814167] combo@combo LOG:  statement: SET search_path = combo_fredcook_dev_entrou
vert_org,public
2018-06-04 14:03:08.210 UTC [1814167] combo@combo LOG:  statement:  DISCARD ALL
2018-06-04 14:03:08.481 UTC [1814153] combo@combo LOG:  statement: SELECT current_setting('transaction_isolati
on')
2018-06-04 14:03:08.484 UTC [1814153] combo@combo LOG:  statement: SET search_path = combo_fredcook_dev_entrou
vert_org,public
2018-06-04 14:03:08.489 UTC [1814153] combo@combo LOG:  statement:  DISCARD ALL
```

#### #40 - 04 juin 2018 18:46 - Christophe Siraut

*- Statut changé de Nouveau à Résolu (à déployer)*

mon patch est inutile, il suffit d'ajouter un paramètre dans settings:

```
TENANT_LIMIT_SET_CALLS = True
```

et l'autre partie de la solution c'est d'activer le cache des requêtes propres à pg_pool.

#### #41 - 04 juin 2018 22:36 - Benjamin Dauvergne

Le souci du cache de requête de pgpool c'est qu'il n'est invalidé que quand il voit passer une écriture, or si on a un pgpool par cluster (un à gra un à rbx, ce qui paraîtrait logique, on ne veut pas 1,5ms de latence vers un pgpool distant) leurs caches vont devenir incohérent, l'utilisation de pgpool suppose que toutes les instances attaquent le même pgpool.

Avec memcached ce sera pareil on aura 1,5ms de latence vers un memcached commun entre gra et rbx; vraiment avoir plus de 20 ou 30 requêtes sur l'affichage d'une page fréquente, c'est le mal; ou alors il faut oublier d'avoir du load balancing sur des sites distants, et considérer le distant comme un fail-over à chaud seulement; ou alors faire du load balancing par domaine.

#### #42 - 05 juin 2018 11:29 - Christophe Siraut

*- Statut changé de Résolu (à déployer) à En cours*

#### #43 - 05 juin 2018 11:55 - Christophe Siraut

Désactivé le cache pgpool, la page répond en 200ms (versus 89ms avec le cache). En diminuant les requêtes vers primary ("SELECT count(*) FROM pg_class …", voir doc attaché "primary") on devrait continuer à améliorer la vitesse, par exemple avec le nouveau jsonfield de combo. Je ne comprends pas encore ce que sont ces "SELECT count", ni pourquoi ils sont routés vers la db primaire, elles sont évoquées dans la FAQ:

http://pgpool.net/mediawiki/index.php/FAQ#When_I_check_pg_stat_activity_view.2C_I_see_a_query_like_.22SELECT_count.28.2A.29_FROM_pg_catalog.pg_class_AS_c_WHERE_c.oid_.3D_pgpool_regclass.28.27pgbench_accounts.27.29_AND_c.relpersistence_.3D_.27u.27.22_in_active_state_for_very_long_time._Why.3F

#### #44 - 05 juin 2018 13:18 - Benjamin Dauvergne

Tu peux activer le cache des query uniquement sur les tables postgres via

```
white_memqcache_table_list = "pg_class,pg_namespace,pg_catalog.pg_class"
```

#### #45 - 06 juin 2018 14:38 - Christophe Siraut

> Tu peux activer le cache des query uniquement sur les tables postgres via [...]

Fait, j'ai réactivé l'ensemble des noeuds du cluster, les perfs sont excellentes:

https://combo-fredcook.dev.entrouvert.org/

#### #46 - 06 juin 2018 14:55 - Frédéric Péters

```
fred@leucas:~$ ab -t 20 https://combo-fredcook.dev.entrouvert.org/une-autre-page/
...
Percentage of the requests served within a certain time (ms)
  50%     86
  66%     87
  75%     88
  80%     89
  90%     95
  95%     98
  98%    131
  99%    146
 100%    359 (longest request)
```

Il reste des moments qui sont doublement longs, une idée ?

#### #47 - 06 juin 2018 15:15 - Christophe Siraut

> Il reste des moments qui sont doublement longs, une idée ?

Une fois de plus j'avais parlé trop vite: mes maj n'étaient pas terminées quand j'ai posté ce message (et dans le cas présent il subsitait un problème de permissions pgpool maintenant résolu)

Je viens de relancer ab:

Percentage of the requests served within a certain time (ms)
50%  84
66%  85
75%  86
80%  87
90%  95
95%  118
98%  130
99%  146
100%  167 (longest request)

Il ya effectivement des moments 2x plus long (lesquels restent très rapides), je n'ai pas encore d'explication. Et je pensais ajouter un header avec les infos de routage haproxy.

#### #48 - 31 août 2018 17:23 - Christophe Siraut

*- Statut changé de En cours à Rejeté*

> mon patch est inutile, il suffit d'ajouter un paramètre dans settings

[…]

### Fichiers

| | | | |
|---|---|---|---|
| 0001-mutlitenant-support-inital-de-base-de-donn-es-mutlip.patch | 1,97 ko | 26 mai 2018 | Christophe Siraut |
| 0002-multitenant-add-ClusterRouter-which-separate-reads-a.patch | 2,17 ko | 26 mai 2018 | Christophe Siraut |
| fredcooklog-une-autre-page.sql | 91,8 ko | 01 juin 2018 | Christophe Siraut |
| on_primary | 41,7 ko | 04 juin 2018 | Christophe Siraut |
| on_replica | 63,2 ko | 04 juin 2018 | Christophe Siraut |
| pglog | 420 ko | 04 juin 2018 | Christophe Siraut |
| 0001-do-not-reset-tenant-schema-when-already-set.patch | 1,06 ko | 04 juin 2018 | Christophe Siraut |
| primary | 9,93 ko | 04 juin 2018 | Christophe Siraut |