Passerelle - Development #88761

Partitionner la table ResourceLog

27 mars 2024 11:58 - Pierre Ducroquet

Statut:	Nouveau	**Début:**	27 mars 2024
Priorité:	Normal	**Echéance:**	
Assigné à:		**% réalisé:**	0%
Catégorie:		**Temps estimé:**	0:00 heure
Version cible:			
Patch proposed:	Non	**Planning:**	Non

Description

Cette table contient des données historisées, et qui sont régulièrement purgées.
L'autovacuum est supposé aider ce genre de situation, mais avec le modèle en multi-tenant, il est nécessaire de ne pas compter que sur lui et de l'assister un peu (la table se retrouve avec un bloat significatif).
De plus, même avec l'autovacuum, de nombreuses entrées invalides finissent par augmenter la taille de l'index.
Je voudrais donc que l'on change la table base_resourcelog afin qu'elle soit partitionnée.
J'envisage d'avoir une partition par semaine, avec un cron côté passerelle qui crée 2 partitions par avance, et supprime les anciennes partitions quand elles sont vides.
Le partitionnement PostgreSQL est transparent pour Django, mais il existe dans django-postgres-extra tout un ensemble d'éléments pour automatiser le fonctionnement. Cf. https://django-postgres-extra.readthedocs.io/en/master/table_partitioning.html
Est-ce-que l'on peut utiliser ces outils dans passerelle, ou est-ce-que je vois pour mettre en place manuellement le partitionnement ?

Demandes liées:

Lié à Passerelle - Development #88953: Optimiser les requêtes de suppressions...	**Solution proposée** 02 avril 2024
Lié à Passerelle - Development #88960: ResourceLog: utiliser un index BRIN su...	**Solution proposée** 02 avril 2024

Historique

#1 - 27 mars 2024 12:10 - Frédéric Péters

Je dirais d'attendre qu'il n'y ait plus personne en buster, le paquet est disponible à partir de bookworm, https://packages.debian.org/bookworm/python3-django-postgres-extra (version 2.0.8).

#2 - 28 mars 2024 00:31 - Benjamin Dauvergne

Actuellement on a que deux index sur cette table :

- transaction_id <- uuid.uuid4() (donc random) créé à chaque instanciation d'une instance
- appname, -timestamp (le -timestamp me semble inutile, ça triera aussi bien dans un sens ou dans l'autre)

Est-ce que si on remplaçait le uuid4() par un UUID croissant (UUIDv7 basé sur un timestamp, https://www.ietf.org/archive/id/draft-peabody-dispatch-new-uuid-format-04.html, en python[1]) et qu'on se basait seulement sur un index BRIN pour les deux et du scan séquentiel pour filtrer appname on améliorerait pas le comportement des index vis à vis d'autovacuum sans y perdre grand chose ?

Parce qu'une table de log donc append only ça me semble le cas idéal pour un ramasse miette, ça ne devrait pas poser de problème.

[1] https://github.com/oittaa/uuid6-python/blob/main/src/uuid6/__init__.py

---

Autre chose, le clean_logs est implémenté ainsi :

```
    def clean_logs(self):
        # clean logs
        timestamp = timezone.now() - datetime.timedelta(
            days=self.logging_parameters.log_retention_days or settings.LOG_RETENTION_DAYS
        )
        ResourceLog.objects.filter(
            appname=self.get_connector_slug(), slug=self.slug, timestamp__lt=timestamp
        ).delete()
```

On passe sur chaque instance de modèle pour supprimer ses logs spécifiques. J'ai regardé en prod on a cette répartition des délais sur les instances de connecteurs :

jours	nombre de modèles
7	1277
10	1

15	1
30	1
60	1
93	1
300	1

Pour les valeurs qui sortent de la valeur par défaut les connecteurs concernés sont :

```
10 {'ToulouseMaelis'}
15 {'SolisAfiMss'}
30 {'CaluireAxel'}
60 {'ToulouseAxel'}
93 {'OVHSMSGateway'}
300 {'Greco'}
```

Je me dis qu'on pourrait déjà optimiser ce code de suppression pour ne pas supprimer par connecteur mais tout supprimer d'un coup quand le délai est commun; et peut-être ne plus permettre de paramétrer cette durée par connecteur et ainsi simplifier définitivement.

#3 - 02 avril 2024 10:55 - Pierre Ducroquet

Benjamin Dauvergne a écrit :

> Actuellement on a que deux index sur cette table :
>
> - transaction_id <- uuid.uuid4() (donc random) créé à chaque instanciation d'une instance
> - appname, -timestamp (le -timestamp me semble inutile, ça triera aussi bien dans un sens ou dans l'autre)
>
> Est-ce que si on remplaçait le uuid4() par un UUID croissant (UUIDv7 basé sur un timestamp, https://www.ietf.org/archive/id/draft-peabody-dispatch-new-uuid-format-04.html, en python[1]) et qu'on se basait seulement sur un index BRIN pour les deux et du scan séquentiel pour filtrer appname on améliorerait pas le comportement des index vis à vis d'autovacuum sans y perdre grand chose ?
>
> Parce qu'une table de log donc append only ça me semble le cas idéal pour un ramasse miette, ça ne devrait pas poser de problème.

Sauf que ce n'est pas comme ça que fonctionne l'autovacuum. Le comparer à un ramasse-miettes perd la "subtilité" du stockage par fichiers comparé au stockage en RAM. Là où un ramasse-miettes à la JVM peut se permettre de bouger des objets après coup pour éviter la fragmentation, le vacuum de PG ne peut que marquer de l'espace comme réutilisable ultérieurement. De ce fait, avec la méthode de suppression actuelle, le BRIN va en plus perdre en efficacité (tout en restant plus léger que le BTREE, juste au prix d'un filtrage plus long post-scan, c'est pas la fin du monde normalement mais ça se mesure, je ferai l'expérience sur un tenant). Changer la génération des uuid n'aidera pas grand chose à la situation, ça rendrait juste le brin utilisable sur le transaction_id, pas un changement radical.

Un second problème, plus fondamental et sur lequel je travaille, c'est qu'on approche des limites de l'autovacuum avec notre nombre de tables. Je cherche encore quels indicateurs utiliser pour vérifier ça, mais quand je montre le nombre de tables à n'importe quel développeur PG, la première réaction est que l'autovacuum ne peut pas tenir la charge. Donc je préfère préparer le terrain sur ce qui peut lui simplifier la tâche, voire le rendre inutile. À court terme, à défaut de partitionnement, un vacuum analyze post-purge de la table ne serait pas du luxe (et est d'ailleurs recommandé même quand l'autovacuum n'est pas sous l'eau).

[1] https://github.com/oittaa/uuid6-python/blob/main/src/uuid6/__init__.py

---

> Autre chose, le clean_logs est implémenté ainsi :
> [...]
> On passe sur chaque instance de modèle pour supprimer ses logs spécifiques. J'ai regardé en prod on a cette répartition des délais sur les instances de connecteurs :

jours	nombre de modèles
7	1277
10	1
15	1
30	1
60	1
93	1
300	1

> Pour les valeurs qui sortent de la valeur par défaut les connecteurs concernés sont :
> [...]
>
> Je me dis qu'on pourrait déjà optimiser ce code de suppression pour ne pas supprimer par connecteur mais tout supprimer d'un coup quand le délai est commun; et peut-être ne plus permettre de paramétrer cette durée par connecteur et ainsi simplifier définitivement.

Si on pouvait simplifier, ça ferait du bien dans tous les cas, vu qu'on augmenterait le nombre de pages libres et donc on réduirait la fragmentation du log

#4 - 02 avril 2024 11:42 - Benjamin Dauvergne

Pierre Ducroquet a écrit :

> Benjamin Dauvergne a écrit :
>
> > Actuellement on a que deux index sur cette table :
> >
> > - transaction_id <- uuid.uuid4() (donc random) créé à chaque instanciation d'une instance
> > - appname, -timestamp (le -timestamp me semble inutile, ça triera aussi bien dans un sens ou dans l'autre)
> >
> > Est-ce que si on remplaçait le uuid4() par un UUID croissant (UUIDv7 basé sur un timestamp, https://www.ietf.org/archive/id/draft-peabody-dispatch-new-uuid-format-04.html, en python[1]) et qu'on se basait seulement sur un index BRIN pour les deux et du scan séquentiel pour filtrer appname on améliorerait pas le comportement des index vis à vis d'autovacuum sans y perdre grand chose ?
> >
> > Parce qu'une table de log donc append only ça me semble le cas idéal pour un ramasse miette, ça ne devrait pas poser de problème.
>
> Sauf que ce n'est pas comme ça que fonctionne l'autovacuum. Le comparer à un ramasse-miettes perd la "subtilité" du stockage par fichiers comparé au stockage en RAM. Là où un ramasse-miettes à la JVM peut se permettre de bouger des objets après coup pour éviter la fragmentation, le vacuum de PG ne peut que marquer de l'espace comme réutilisable ultérieurement.

Je sais ça, ramasse miette c'est un terme générique pour moi je ne pensais pas au ramasse miette de la JVM en particulier. Je sais juste que quand les allocations et désallocation sont faite en masse de manière contiguë ça marche mieux, en général, quelque soit l'algo de gestion mémoire.

> De ce fait, avec la méthode de suppression actuelle, le BRIN va en plus perdre en efficacité (tout en restant plus léger que le BTREE, juste au prix d'un filtrage plus long post-scan, c'est pas la fin du monde normalement mais ça se mesure, je ferai l'expérience sur un tenant). Changer la génération des uuid n'aidera pas grand chose à la situation, ça rendrait juste le brin utilisable sur le transaction_id, pas un changement radical.

J'aurai du expliciter ma compréhension du problème. Pour moi il y a deux aspects les suppressions dans la table et dans les index.

Coté table le fait de supprimer les lignes de chaque connecteur séparément ça crée un gruyère progressif, peut-être qu'à la fin on a des pages complètement libre mais je me dis que libérer les pages d'un coup via un DELETE ... WHERE timestamp < ... est peut-être plus efficace, ça reste une intuition.

Coté index le transaction_id actuel produit du gruyère quand on les supprime, et du gruyère persistant, i.e. des pages pas complètement inutilisés, que seul un appel à VACUUM FULL pourra beaucoup améliorer (je suppose que postgres réutilise l'espace vide dans les pages aussi, m'enfin ça reste perfectible) il m'aurait semblé que l'utilisation d'un identifiant séquentiel associé à un index BRIN, dont il me semble l'avantage est quand la progression de la clé est le même que celle des insertions en table, cas typique d'un timestamp.

Idem pour l'index appname,-timestamp le fait de mettre appname en premier au lieu de l'élément qui suit les insertions ça rend le clustering de l'index moins bon je pense, pour un gain en index scan qui doit pas être énorme (c'est pas comme si on pensait notre temps à consulter les logs, 99,9999999% des logs ne sont jamais consultés).

Autre possibilité avec un uuid intégrant un timestamp comme préfixe, on pourrait complètement supprimer l'index sur transaction_id et se servir de l'index sur le timestamp comme hint de recherche, (SELECT ... WHERE transaction_id = ? AND timestamp BETWEN transaction_id.tstamp - 1 minute AND transaction_id.stamp + 1 minute), on gagne un index.

> Un second problème, plus fondamental et sur lequel je travaille, c'est qu'on approche des limites de l'autovacuum avec notre nombre de tables. Je cherche encore quels indicateurs utiliser pour vérifier ça, mais quand je montre le nombre de tables à n'importe quel développeur PG, la première réaction est que l'autovacuum ne peut pas tenir la charge. Donc je préfère préparer le terrain sur ce qui peut lui simplifier la tâche, voire le rendre inutile. À court terme, à défaut de partitionnement, un vacuum analyze post-purge de la table ne serait pas du luxe (et est d'ailleurs recommandé même quand l'autovacuum n'est pas sous l'eau).

Je rajoute l'aspect VACUUM ANALYZE au point suivant.

> Si on pouvait simplifier, ça ferait du bien dans tous les cas, vu qu'on augmenterait le nombre de pages libres et donc on réduirait la fragmentation du log

Ok sur ça on est bien d'accord, je vais ouvrir un ticket dans ce sens.

#5 - 02 avril 2024 11:46 - Benjamin Dauvergne

- Lié à Development #88953: Optimiser les requêtes de suppressions des ResourceLog expirés ajouté

#6 - 02 avril 2024 13:24 - Benjamin Dauvergne

- Lié à Development #88960: ResourceLog: utiliser un index BRIN sur timestamp ajouté