w.c.s. - Development #33280

distribuer le cron sur plusieurs CPU

20 mai 2019 17:10 - Frédéric Péters

Statut:	Fermé	**Début:**	20 mai 2019
Priorité:	Normal	**Echéance:**	
Assigné à:	Frédéric Péters	**% réalisé:**	0%
Catégorie:		**Temps estimé:**	0:00 heure
Version cible:			
Patch proposed:	Oui	**Planning:**	Non

Description

On a une tâche cron qui occupe un CPU longtemps, à passer sur tous les tenants; ceux-ci sont indépendants, le taf pourrait être distribué sur plusieurs processeurs.

Révisions associées

Révision f78f46ed - 20 octobre 2022 20:54 - Frédéric Péters

cron: allow some cron workers to be run in parallel (#33280)

Révision d937607c - 25 octobre 2022 17:13 - Frédéric Péters

Revert "cron: allow some cron workers to be run in parallel (#33280)"

This reverts commit f78f46ed2d1bc8145dc6bc1df6886fad074c1514.

Révision 1717a737 - 31 octobre 2022 16:48 - Frédéric Péters

cron: allow some cron workers to be run in parallel (#33280)

Historique

#2 - 13 juillet 2022 10:56 - Frédéric Péters

Pour atteindre ça l'idée basique était d'utiliser multiprocessing et distribuer le tenants dessus mais ça garde comme inconvénient qu'un tenant qui fait des choses qui prennent du temps trainera les autres.

Plutôt, je serais pour retirer le lock sur le job cron en tant que tel et modifier le fonctionnement pour avoir un lock par tenant, dans la db wcs_meta, ça pourrait être key=cron-<tenant>, value=(needed, running, done).

En continuant avec le cron lancé toutes les minutes,

- on démarre minute 0, ça met tous les tenants sur "needed",
  - ça commence à traiter les premiers, [R, N, N, N, N, N, N, N, N, N] (running, needed, needed, etc.)
- minute 1, cron à nouveau déclenché, on arrive sur la situation [D, D, R, N, N, N, N, N, N, N] (done, done, running, needed, etc.)
  - ça se lance donc sur le premier "needed", puis ça suit
- minute 2, re-cron, on est peut-être alors sur [D, D, D, D, D, R, R, N, N, N]
  - ça continue
- minute 3, [D, D, D, D, D, R, D, D, R, R]
  - aucun needed, ça peut transformer les "done" en "needed" et reprendre
- ...

Si jamais on arrive à n tenants en cours ("running"), on quitte sans rien faire. (le n étant un paramètre qu'on taperait en settings).

Avec l'exécution parallèle, il devient nécessaire de faire attention aux écritures dans les logs (cron.log-...), mais on pourrait se dire que la vue globale linéaire n'est plus nécessaire maintenant que les exécutions ne sont plus ordonnées, et plutôt créer un log par tenant, ce qui supprime la question.

Via la colonne updated_at, si un job est en "running" depuis trop longtemps (genre plus de 24h), on transforme en "needed" et raise CommandError(), ce qui fera alerte minimale, proche de ce qu'on a aujourd'hui,

```
            if age > JUMP_TIMEOUT_INTERVAL * 10:
                raise CommandError(
                    'can not start cron job, locked by %s for %s seconds (seems old)' % (lockfile, age)
                )
```

#3 - 02 août 2022 16:12 - Thomas Noël

Simple et efficace, sans grande magie, compatible Devuan et sans doute docker (aka sans uwsgi et/ou sans systemd), c'est bien.

J'imagine bien une commande « cron-status » qui liste les tenants et nous dit là où les cron travaillent, ça sera bien utile. Un prometheus montrera les temps d'exécution du cron tenant par tenant (on peut imaginer une commande cron-prometheus directement compatible prometheus) et alertera quand ça déborde (running trop long).

Le update_at j'y vois plusieurs usages :

- sur "running" : comme tu as dit, pour repérer ceux qui sont en route depuis plus de 24h, et prometheus nous alertera (cf ci-dessus)-- je ne suis pas certain qu'on ait envie maintenant d'un "stop" brutal, j'aimerai éviter le temps de voir la réalité et de faire un peu d'analyse des lenteurs ?
- sur "needed" : on garderait le "update_at" du "done" et cela donne l'ordre de passage en "running" (i.e. les cron qui se sont terminés il y a longtemps sont prioritaires pour être relancés)

Quant au "n" je verrais bien un « CRON_WORKERS = os.cpu_count()*3 » dans les settings par défaut.

Question : il y a une table meta par tenant, donc pourquoi key=cron-<tenant> et pas juste key=cron ?

Question 2 : quid de parvenir à répartir ensuite sur les multiples node ? Si on gère bien les lock sur les meta ça pourrait être juste gratuit, il "suffirait" d'activer le système de cron sur tous les nodes. (je suis pas un grand spécialiste, mais bon, select for update et ce genre de chose ?)

#4 - 02 août 2022 16:27 - A. Berriot

Ça a l'air d'impliquer beaucoup de développements et de maintenance sur des questions relativements complexes (timeouts, retry, locks, distribution sur plusieurs noeuds...). Est-ce qu'il y a des raisons de ne pas utiliser des outils existants type Celery, RQ ou Procrastinate pour l'exécution des jobs et si oui lesquelles ?

#5 - 02 août 2022 17:00 - Thomas Noël

Agate Berriot a écrit :

> Ça a l'air d'impliquer beaucoup de développements et de maintenance sur des questions relativements complexes (timeouts, retry, locks, distribution sur plusieurs noeuds...). Est-ce qu'il y a des raisons de ne pas utiliser des outils existants type Celery, RQ ou Procrastinate pour l'exécution des jobs et si oui lesquelles ?

Parce que c'est de la m*rde ? Ok j'ai rien dit. Blague (?) à part c'est assez overkill par rapport à nos besoins, et on a aussi besoin de métriques que ces outils ne proposent pas facilement. On a vécu quelques années difficiles avec AMQP/RabbitMQ/Celery, qu'on utilise encore un peu, mais qu'on essaye de quitter pour des solutions plus simples (messages http en synchrone, qui plantent clairement, etc.).

C'est notre côté "vieux cons qui n'aiment pas ne pas voir les jobs avec ps" (je parle pour moi qui n'a jamais compris comment voir ce qui se passe dans RabbitMQ).

#6 - 02 août 2022 17:13 - Frédéric Péters

> Question : il y a une table meta par tenant, donc pourquoi key=cron-<tenant> et pas juste key=cron ?

Oui ça pourrait tout à fait; je ne saurais plus dire d'où mon plan était parti pour vouloir une clé unique mais ça n'est plus nécessaire.

> Question 2 : quid de parvenir à répartir ensuite sur les multiples node ? Si on gère bien les lock sur les meta ça pourrait être juste gratuit, il "suffirait" d'activer le système de cron sur tous les nodes. (je suis pas un grand spécialiste, mais bon, select for update et ce genre de chose ?)

Oui ça doit tourner tout seul.

#7 - 02 août 2022 18:07 - A. Berriot

Thomas Noël a écrit :

> Agate Berriot a écrit :
>
> > Ça a l'air d'impliquer beaucoup de développements et de maintenance sur des questions relativements complexes (timeouts, retry, locks, distribution sur plusieurs noeuds...). Est-ce qu'il y a des raisons de ne pas utiliser des outils existants type Celery, RQ ou Procrastinate pour l'exécution des jobs et si oui lesquelles ?
>
> Parce que c'est de la m*rde ? Ok j'ai rien dit. Blague (?) à part c'est assez overkill par rapport à nos besoins, et on a aussi besoin de métriques que ces outils ne proposent pas facilement. On a vécu quelques années difficiles avec AMQP/RabbitMQ/Celery, qu'on utilise encore un peu, mais qu'on essaye de quitter pour des solutions plus simples (messages http en synchrone, qui plantent clairement, etc.).
>
> C'est notre côté "vieux cons qui n'aiment pas ne pas voir les jobs avec ps" (je parle pour moi qui n'a jamais compris comment voir ce qui se passe dans RabbitMQ).

Il y a évidemment des trucs plus lourds que d'autres. Mais en 3 ans, depuis l'ouverture du ticket, le paysage a quand même évolué un peu. Je n'ai

pas l'impression que ce type d'outils soient spécialement overkill par rapport aux besoin décrits ici. Au contraire, ça gère précisément ce qui est exprimé : des locks, de la distribution, des timeouts, de l'exécution récurrente.

Si je prends l'exemple de procrastinate, ça se base uniquement sur du PostgreSQL, donc ça ne fait aucune nouvelle brique type rabbitmq/redis. Les jobs sont stockés dans une table en base, et c'est donc possible de faire des stats dessus, de sortir des métriques, etc.

Il me semble qu'à partir du moment ou on rajoute de la distribution dans l'équation, la complexité est vouée à augmenter, quel que soit l'outil. Et le debug avec ps sera vite pénible pour débugger un problème d'exécution sur un pool de 3 noeuds, ou 10.

Développer en interne un moteur d'exécution qui fasse tout ça et qui reste compréhensible, maintenable et utilisable par l'ensemble de l'équipe, ça me paraît compliqué. Ça sera certainement aussi merdique que l'existant, voire pire.

#8 - 02 août 2022 18:21 - Thomas Noël

Agate Berriot a écrit :

> Développer en interne un moteur d'exécution qui fasse tout ça et qui reste compréhensible, maintenable et utilisable par l'ensemble de l'équipe, ça me paraît compliqué. Ça sera certainement aussi merdique que l'existant, voire pire.

Non mais hé, ho, dis, l'existant n'est pas merdique... Il est juste séquentiel :)

En fait je ne vois pas d'énorme travail ici, du moins rien de si compliqué. Je n'ai surtout pas envie d'utiliser un truc qui ne va plus marcher dans n mois, ou ne pas être à jour dans Debian, etc. En gain supplémentaire, on gagnera le fait de contrôler complétement comment les jobs sont lancés, répartis, suivi, supervisés, etc. On est sur une partie très importante de wcs, le cron pilote énormément de choses, qu'on ne peut pas juste déléguer à un outil qu'on ne va pas complétement maîtriser.

#9 - 02 août 2022 18:46 - A. Berriot

Thomas Noël a écrit :

> Agate Berriot a écrit :
>
> > Développer en interne un moteur d'exécution qui fasse tout ça et qui reste compréhensible, maintenable et utilisable par l'ensemble de l'équipe, ça me paraît compliqué. Ça sera certainement aussi merdique que l'existant, voire pire.
>
> Non mais hé, ho, dis, l'existant n'est pas merdique... Il est juste séquentiel :)

L'existant : les briques qui existent déjà en libre, comme celles que j'ai mentionnées dans mes messages précédents.

#10 - 02 août 2022 19:03 - Frédéric Péters

Pour donner ma perspective, sur l'historique, sur des années il n'y a personne qui a pu prendre celery/rabbitmq et permettre de rendre possible le debug; on se trouve encore à lancer un restart de l'hobo-agent après un déploiement parce que sinon il y a mille threads qui viennent s'ajouter et ne partent jamais. Il y a de l'expiration et pas de rejeu et pas vraiment de vue dessus, messages perdus. Il y a des graphes ajoutés dans munin qui n'indiquent rien d'utile.

Pour les autres outils cités procrastinate ça n'est pas packagé dans debian (bien sûr on peut le faire mais ça fait un autre truc à maintenir maison). Pour rq c'est dans debian mais ça ajoute une dépendance redis qui m'est peu enthousiasmante (biais personnel).

Plus pratiquement, dans ce qu'on a pu voir on se trouve souvent à devoir bricoler pour l'intégration avec les tenants, avec aucune solution évidente. Basiquement ici pour w.c.s. on n'a même pas de base unique pour tous les tenants donc c'est un prérequis de procrastinate qui va demander une réflexion particulière, en anticipant peut-être des galères là où on est on premise sans possibilité facile de créer une nouvelle base dédiée.

Passé ça très certainement ça pourrait tenir, on enregistrerait un job par tenant "exécute les cron de tel tenant" et on utiliserait une "queueing lock" par tenant, et le contenu de la tâche serait en gros un subprocess.call(["wcs-manage", "cron", "-d", tenant_name, "--job", "evaluate_jumps"]) pour lancer le job en question). Ça retire en passant l'argument "pouvoir voir ce qui tourne avec ps" et ça fait sans doute quelque chose d'assez propre.

Vraiment perso c'est la mise en place d'un nouveau module qui me bloque là-dessus, vs la possibilité que je vois d'apporter ça avec zéro installation / intervention sur les déploiements existants, juste la mise à jour wcs.

Je ne pense pas que ça apportera une complexité particulière ici, et elle sera de toute façon posée dans un unique fichier.

Aussi, ça mettra de toute façon en place ce qui serait nécessaire quelle que soit l'infrastructure, comme les logs par tenant plutôt que globaux; si jamais on se trouve à intégrer procrastinate à un moment dans l'infra, pour un autre besoin dans un autre module, on aura de toute façon déjà le principal.

(bref je vais sans doute regarder pour coder ça bientôt)

#11 - 03 août 2022 09:18 - A. Berriot

Merci pour les explications, effectivement, le fonctionnement un tenant/un schema sans base unique apporte une complexité supplémentaire.

Néanmoins, il me semble que dans le cas de notre plateforme en Saas, créer une base dédiée à procrastinate ne pose pas problème. Et pour les on-premise, on peut simplement utiliser la base unique et existante du tenant pour y créer les tables de procrastinate, sans avoir créer une nouvelle base.

Vu que les jobs en eux même sont simplement "récupérer la liste des tenants" et exécuter un job par tenant, cela ne devrait pas demander d'autre adaptation. Comme tu le décris :

1. le worker procrastinate démarre
2. il lance le cron qui récupère la liste des tenants et lance un job par tenant
3. le ou les autres workers procrastinante recoivent ce job et l'exécutent via un subprocess, dans le contexte du tenant

> Vraiment perso c'est la mise en place d'un nouveau module qui me bloque là-dessus, vs la possibilité que je vois d'apporter ça avec zéro installation / intervention sur les déploiements existants, juste la mise à jour wcs.

Effectivement, ça demanderait le packaging debian de procrastinate (ceci dit, j'ai l'impression que toutes les dépendances de procrastinate sont déjà packagées, à l'exception de attrs).

#12 - 04 août 2022 09:49 - Frédéric Péters

J'ai passé un peu de temps hier soir sur le sujet, sans finir par être suffisamment à l'aise avec l'introduction de procrastinate pour cette tâche. Je me dis qu'une bonne introduction pourrait être le parallélisme des jobs bijoe/wcs-olap (#67927) avec une "queueing lock" par hobo primaire. Ça permettrait d'y prendre le temps du packaging, du déploiement et de nous faire la main sur le suivi. (alternativement / plus ambitieux, ça serait pour remplacer rabbitmq pour les messages de déploiements "hobo_deploy", même si là je suis peut-être tenté par juste éliminer le côté distribué de l'affaire).

~~

Une fois cette décision posée, j'ai codé le truc; ça fait un diff assez peu lisible à cause de l'élimination de locket qui faisait que tout était indenté de deux niveaux (try/with) (pareil côté tests).

Je notais plus haut qu'on aurait de toute façon ici le déplacement des logs pour qu'ils soient par tenant, que ça servirait aussi pour une adoption de procrastinate, mais c'est en fait assez dérisoire comme changement.

Pour le reste, ça tourne ainsi, on itère sur les tenants et :

- on récupère le statut de l'exécution (needed/running/done)
- si c'est needed on modifie la ligne pour noter que c'est running (de manière pseudo atomique, les personnes qui veulent faire de l'SQL pourront ajuster get_and_update_cron_status).
- si le nombre de running dépasse le nombre de workers autorisés, on abandonne (et on remet la ligne en needed si nécessaire)
- si on est donc sur un tenant à traiter, on le traite
- et à la fin on dit qu'il est traité

Une fois bouclé si on n'a vu aucun running en cours, on modifie les lignes de db pour passer de done en needed.

> Question : il y a une table meta par tenant, donc pourquoi key=cron-<tenant> et pas juste key=cron ?

Fun fact, dans la db j'ai finalement nommé la ligne cron-status-le-nom-du-tenant, c'est parce que les tests d'exécution sur plusieurs tenants (préexistants) ont été bricolés pour partager la même base de données (parce que c'était plus rapide et plus facile à nettoyer ainsi).

~~

(jenkins actuellement fâché sur du détail pylint)

#13 - 05 août 2022 19:01 - Thomas Noël

Juste pour que je sois bien sûr de moi : dans ce code, tu comptes sur le lancement chaque minute pour obtenir le parallélisme ? (je m'étais dit que tu lancerais à un moment des subprocess-ou-whatever, mais comme écrit là c'est tout simple et au bout de quelques minutes on aura bien les cron en parallèle comme il faut).

#14 - 05 août 2022 19:28 - Frédéric Péters

Oui ça ne change rien au lancement actuel. Aujourd'hui le lock global faisait que le processus s'arrêtait tôt, alors qu'avec le patch on parcourt tous les tenants pour traiter un autre tenant en parallèle, puis minute d'après un autre, etc. jusqu'à avoir n tenants en parallèle, et une fois cette limite atteinte après le parcours des tenants et vu qu'il y en a déjà n en cours, chaque nouveau processus s'arrête sans traiter aucun tenant.

#15 - 05 août 2022 19:38 - Frédéric Péters

En réalité cependant, on garde la découpe "granularité 20 minutes" et donc on va plutôt avoir :

- minute 0 : un processus, qui va commencer à traiter séquentiellement les tenants,
- minutes 1, 2, 3… 19 : les processus vont tourner mais ne pas trouver de job à lancer pour la minute en question,
- minute 20 : nouveau processus et celui-ci étant sur une minute "de démarrage", va trouver des jobs.

Note que je simplifie un peu en parlant uniquement des jobs d'évaluation des sauts, mais ce sont les plus nombreux et ceux qui ont ce

déclenchement fréquent.

Éventuellement une possibilité plus loin pourra être de permettre au job "évaluation saut" de tourner toutes les minutes, pour charger plus rapidement la barque mais ça aurait des conséquences indésirables sur un hébergement type imio, où le décalage en 20 minutes, avec des offsets différents, permet de ne pas surcharger les fermes de conteneurs. Mais je ne touche pas à ça pour le moment.

#16 - 06 août 2022 01:53 - Thomas Noël

Ok. On pourra éventuellement jouer sur WCS_JUMP_TIMEOUT_CHECKS sur notre SaaS, genre le passer à 60 (ou 20 si on veut conserver notre réputation de conservateurs). On pourrait aussi le passer à 60 par défaut et dire à IMIO de bouger sa valeur.

(notons que c'est actuellement une variable d'environnement mais ça peut facilement devenir aussi un settings).

#17 - 30 août 2022 14:14 - Frédéric Péters

- Fichier 0001-cron-allow-some-cron-workers-to-be-run-in-parallel-3.patch ajouté

- Statut changé de Nouveau à Solution proposée

- Assigné à mis à Frédéric Péters

- Patch proposed changé de Non à Oui

je me rends compte que j'avais oublié de proposer ça.

#18 - 30 août 2022 14:36 - Thomas Noël

J'avais déjà lu et tout me parait OK, sauf la partie dans sql.py (get_and_update_cron_status / mark_cron_status) où j'aimerais être rassuré par un PostgreSQLman si on a bien impossibilité de superposition depuis plusieurs nodes (dans l'idée qu'on va activer les cron wcs sur les deux nodes grâce à ça).

#19 - 30 août 2022 14:53 - A. Berriot

Ça ne constitue pas une relecture complète, mais peut-être passer `CRON_WORKERS = os.cpu_count() * 3` dans une variable d'environnement avec un default a os.cpu_count() * 3 pour pouvoir adapter le nombre de workers facilement?

#20 - 30 août 2022 15:18 - Thomas Noël

Agate Berriot a écrit :

> Ça ne constitue pas une relecture complète, mais peut-être passer `CRON_WORKERS = os.cpu_count() * 3` dans une variable d'environnement avec un default a os.cpu_count() * 3 pour pouvoir adapter le nombre de workers facilement?

C'est déjà possible avec notre système de settings : pour adapter une valeur par défaut, on pose un /etc/wcs/settings.d/cron_workers.py avec un CRON_WORKERS = 36 (voir la fin de debian/debian_config.py qui est chargé en fin de wcs/settings.py).

#21 - 30 août 2022 15:25 - A. Berriot

Thomas Noël a écrit :

> Agate Berriot a écrit :
>
> > Ça ne constitue pas une relecture complète, mais peut-être passer `CRON_WORKERS = os.cpu_count() * 3` dans une variable d'environnement avec un default a os.cpu_count() * 3 pour pouvoir adapter le nombre de workers facilement?
>
> C'est déjà possible avec notre système de settings : pour adapter une valeur par défaut, on pose un /etc/wcs/settings.d/cron_workers.py avec un CRON_WORKERS = 36 (voir la fin de debian/debian_config.py qui est chargé en fin de wcs/settings.py).

ok (je trouve personnellement plus simple de pouvoir lancer un programme et modifier son comportement sans avoir nécessairement à passer par un fichier)

#22 - 01 septembre 2022 16:22 - Thomas Noël

À ce sujet,

```
CRON_WORKERS = os.cpu_count() * 3
```

vu que nos nodes SaaS ont 32 processeurs, je propose qu'on se calme

```
CRON_WORKERS = os.cpu_count()
```

voire

```
CRON_WORKERS = int(os.cpu_count()/2) + 1
```

#23 - 09 septembre 2022 13:39 - Frédéric Péters

- Fichier 0001-cron-allow-some-cron-workers-to-be-run-in-parallel-3.patch ajouté

Voilà avec le nombre de workers abaissé.

#24 - 09 septembre 2022 14:19 - Thomas Noël

Il me reste juste ma question :

> (...) sauf la partie dans sql.py (get_and_update_cron_status / mark_cron_status) où j'aimerais être rassuré par un PostgreSQL-man si on a bien impossibilité de superposition depuis plusieurs nodes (dans l'idée qu'on va activer les cron wcs sur les deux nodes grâce à ça).

Dans ma tête, avec une analogie mutex, j'imagine qu'il faut locker un peu quelque part autour des select/insert/update pour assurer l'affaire ? Ou bien les transactions sont là pour ça et suffisent ? (Question posée à la cantonade)

#25 - 09 septembre 2022 14:57 - Frédéric Péters

Pour être sûr de situer, pour les personnes fan d'SQL et de situations improbables,

```
+    cur.execute("SELECT value FROM wcs_meta WHERE key = %s", (key,))
+    row = cur.fetchone()
+    if row is None:

# oh mais si jamais entre temps un autre process passe au même endroit ?!!

+        cur.execute("INSERT INTO wcs_meta (id, key, value) VALUES (DEFAULT, %s, 'running')", (key,))
+        status = 'needed'
+    elif row[0] == 'needed':
+        cur.execute("UPDATE wcs_meta SET value = 'running', updated_at = NOW() WHERE key = %s", (key,))
+        status = 'needed'
```

ou c'est autre chose ?

#26 - 09 septembre 2022 15:19 - Thomas Noël

Frédéric Péters a écrit :

> Pour être sûr de situer, pour les personnes fan d'SQL et de situations improbables,
> [...]
> ou c'est autre chose ?

C'est ça. J'imagine que la transaction permet de bloquer ça... ou pas ? (Qui est nul ici en SQL ? moi).

#27 - 09 septembre 2022 15:54 - Benjamin Dauvergne

Thomas Noël a écrit :

> Frédéric Péters a écrit :
>
> > Pour être sûr de situer, pour les personnes fan d'SQL et de situations improbables,
> > [...]
> > ou c'est autre chose ?
>
> C'est ça. J'imagine que la transaction permet de bloquer ça... ou pas ? (Qui est nul ici en SQL ? moi).

Non la transaction ne protège de rien ici, on est en READ COMMITTED, une transaction voit tout ce qui a été commité jusqu'à maintenant mais y a aucun verro, donc deux transactions concurrentes peuvent insérer deux fois running pour la même clé; à moins qu'il y ait un index d'unicité sur key mais je n'en vois pas (mais dans ce cas il pourrait y avoir des erreurs SQL à gérer proprement avec un ON CONFLICT).

Le plus simple c'est de faire un LOCK wcs_meta avant tout ça, pas besoin de se prendre la tête sur une table avec peu d'écritures et de lectures.

#28 - 09 septembre 2022 16:35 - Pierre Ducroquet

INSERT avec une clause ON CONFLICT.

https://www.postgresql.org/docs/11/sql-insert.html#SQL-ON-CONFLICT

```
INSERT INTO wcs_meta (id, key, value) VALUES (DEFAULT, %s, 'running')
```

```
  ON CONFLICT (id) DO UPDATE SET value = 'running', updated_at = NOW();
```

#29 - 09 septembre 2022 16:38 - Pierre Ducroquet

Benjamin Dauvergne a écrit :

> Le plus simple c'est de faire un LOCK wcs_meta avant tout ça, pas besoin de se prendre la tête sur une table avec peu d'écritures et de lectures.

C'est un coup à créer de la contention plus tard quand tout le monde aura oublié le lock. Autant l'éviter vu la simplicité de l'upsert, non ? (Puis un lock sans paramètre c'est un access exclusive, ça bloque même les autres select, ce qui est complètement inutile ici)

#30 - 09 septembre 2022 16:57 - Frédéric Péters

Ok mais.

La préoccupation elle est de ne pas avoir en parallèle deux exécutions qui traitent le même tenant; la proposition là elle va pas juste faire que justement ça sera le cas ?

Le souhait ici c'est dans le même temps obtenir le statut actuel de la ligne donnée, si elle existe, et le changer :

- SELECT value FROM wcs_meta WHERE key = ... → le statut
- s'il n'y a rien dans la db
  - INSERT INTO wcs_meta ... running + considérer que le statut actuel était "needed"
- s'il y avait une ligne, avec comme valeur "needed"
  - UPDATE wcs_meta SET value = 'running'
- s'il y avait une ligne, avec un autre statut
  - ne rien changer
- retourner le statut tel qu'il était à l'entrée de la fonction

#31 - 10 septembre 2022 21:47 - Benjamin Dauvergne

Pierre Ducroquet a écrit :

> INSERT avec une clause ON CONFLICT.
>
> https://www.postgresql.org/docs/11/sql-insert.html#SQL-ON-CONFLICT
>
> [...]

Il n'y a pas d'index d'unicité sur la colonne key, aucun conflit n'arrivera jamais. Mais donc je sais tout ça, j'ai juste voulu donner la solution en une ligne au code proposé qui évitera à Fred de s'endormir en lisant mon commentaire. Maintenant je te laisse la place tu es beaucoup de toute façon plus écouté et plus pédagogue que moi. De plus il manque un WHERE dans ton ON CONLICT DO UPDATE pour ne pas ignorer un running nouvellement inséré (plus loin mon idée).

Frédéric Péters a écrit :

> Ok mais.
>
> La préoccupation elle est de ne pas avoir en parallèle deux exécutions qui traitent le même tenant; la proposition là elle va pas juste faire que justement ça sera le cas ?
>
> Le souhait ici c'est dans le même temps obtenir le statut actuel de la ligne donnée, si elle existe, et le changer :
>
> - SELECT value FROM wcs_meta WHERE key = ... → le statut
> - s'il n'y a rien dans la db
>   - INSERT INTO wcs_meta ... running + considérer que le statut actuel était "needed"
> - s'il y avait une ligne, avec comme valeur "needed"
>   - UPDATE wcs_meta SET value = 'running'
> - s'il y avait une ligne, avec un autre statut
>   - ne rien changer
> - retourner le statut tel qu'il était à l'entrée de la fonction

La commande donnée par Pierre accepte en plus une directive RETURNING value, ça donnerait ça au total :

```
INSERT INTO wcs_meta (id, key, value) VALUES (DEFAULT, %s, 'running')
  ON CONFLICT (id) DO UPDATE SET value = 'running', updated_at = NOW() WHERE value = 'needed' RETURNING value;
```

- si ça retourne 'running' ok on a inséré 'running' ou mis à jour un 'needed' existant, on peut retourner 'needed'
- si ça ne retourne rien, c'est que c'était déjà en running, on retourne 'running'

Au risque de me répéter ça exige l'ajout d'un index d'unicité sur key pour fonctionner. J'avais un doute sur les version de posgresql pour tout ça mais ça tourne en 9.6 (https://www.postgresql.org/docs/9.6/sql-insert.html).

#33 - 11 octobre 2022 10:34 - Frédéric Péters

```
INSERT INTO wcs_meta (id, key, value) VALUES (DEFAULT, %s, 'running')
  ON CONFLICT (id) DO UPDATE SET value = 'running', updated_at = NOW() WHERE value = 'needed' RETURNING value;
```

j'ai l'impression que ça zappe le moment initial,

> s'il n'y a rien dans la db
> INSERT INTO wcs_meta ... running + considérer que le statut actuel était "needed"

que sur ce moment ça va retourner "running" alors qu'on voudrait "needed".

(... du temps passé ...)

Trop compliqué pour moi tout ça, je me trouve à modifier ce qui est proposé ici un peu au hasard des messages et échecs, type,

```
2022-10-11 10:18:06 CEST [979970-6] fred@wcstests860 LOG:  statement: INSERT INTO wcs_meta (id, key, value)
                                      VALUES (DEFAULT, 'cron-status-example.net', 'running')
                                ON CONFLICT (key)
                                  DO UPDATE SET value = 'running',
                                                updated_at = NOW()
                                                WHERE key = 'cron-status-example.net' AND value = 'needed'
                                  RETURNING value
2022-10-11 10:18:06 CEST [979970-7] fred@wcstests860 ERROR:  column reference "key" is ambiguous at character
324
```

Je remonte donc dans l'historique du ticket et si la proposition "LOCK" était bien juste de faire ça :

```
     do_meta_table(conn, cur, insert_current_sql_level=False)
     key = 'cron-status-%s' % get_publisher().tenant.hostname
+    cur.execute("LOCK wcs_meta")
     cur.execute("SELECT value FROM wcs_meta WHERE key = %s", (key,))
```

c'est ce qui est désormais dans la branche.

#34 - 11 octobre 2022 11:51 - Frédéric Péters

- Fichier 0001-cron-allow-some-cron-workers-to-be-run-in-parallel-3.patch ajouté

#35 - 11 octobre 2022 12:48 - Benjamin Dauvergne

Frédéric Péters a écrit :

> [...]
>
> j'ai l'impression que ça zappe le moment initial,
>
> > s'il n'y a rien dans la db
> > INSERT INTO wcs_meta ... running + considérer que le statut actuel était "needed"
>
> que sur ce moment ça va retourner "running" alors qu'on voudrait "needed".

C'est exactement ce que j'écrivais, si le SQL retourne running, la fonction python doit retourner 'needed', si ça retourne None c'est que rien n'a été fait, donc le python retourne running.

```
cur.execute('....')
result = cur.fetchone()
if result is None:
    return 'running'
else:
    return 'needed'
```

y a même pas à tester la valeur, s'il y a une valeur renvoyée ce sera toujours 'running'.

```
    2022-10-11 10:18:06 CEST [979970-6] fred@wcstests860 LOG:  statement: INSERT INTO wcs_meta (id, key, value
    )
                                       VALUES (DEFAULT, 'cron-status-example.net', 'running')
                                 ON CONFLICT (key)
                                   DO UPDATE SET value = 'running',
```

```
                                                updated_at = NOW()
                                                WHERE key = 'cron-status-example.net' AND value = 'needed'
                                    RETURNING value
2022-10-11 10:18:06 CEST [979970-7] fred@wcstests860 ERROR:  column reference "key" is ambiguous at charac
ter 324
```

Le rajout du WHERE key n'était pas nécessaire et il y effectivement une ambiguité avec la table virtuelle EXCLUDED qui contient les valeurs qu'on voulait insérer (le DO UPDATE s'éxécute dans le contexte d'une ligne refusée parce que collision sur la clé "key" dont "key" a déjà la bonne valeur). Ma requête ne levait pas d'erreur il me semble et fonctionne, mais je répète il faut introduire un index d'unicité sur "wcs_meta.key" pour utiliser cette requête (et s'assurer que c'est possible).

> Je remonte donc dans l'historique du ticket et si la proposition "LOCK" était bien juste de faire ça :
>
> [...]
>
> c'est ce qui est désormais dans la branche.

Cette partie va fonctionne avec LOCK, oui, mais j'ai plusieurs interrogations :

- je vois bien que ça permet de lancer la commande cron sur plusieurs noeuds voir plusieurs fois sur le même noeud, par contre je ne vois pas ou ça répartit les crons sur les cpus d'un même noeud (il doit bien y avoir un objectif à l'utilisation de os.cpu_count()), où est-ce qu'on forke ? Aussi la référence à os.cpu_count() sachant qu'on aura <nb de noeud> * os.cpu_count() m'interroge, au final on est quand même plus limité par la base que par les CPUs je pense, on devrait fixer un nombre max de tenant à tourner en paralèle et un nombre de noeud et chacun exécute ça. En prod on a 32 cpus par machines, sur deux noeuds ça va faire 34 cron en parallèle avec la configuration du patch actuel,
- je ne vois pas l'implémentation du commentaire plus haut à propos d'un statut == running avec un updated_at plus vieux que 24h (et ça nécessiterait un kill du worker je suppose),
- la façon dont tout est remis à needed si tout le monde est done et personne en running ou needed me chagrine, ça limite la progression de toutes les tenants. On devrait juste remettre à needed tous les tenants qui sont passés à done dans la commande en cours, ça permet au moins au tenant les plus rapide de progresser même si des tenants sont lents. Idéalement on devrait même trier les tenants selon la valeur d'updated_at au moment du done (et donc quand on repasse à needed ne pas modifier updated_at) ce qui ont fini il y a le plus longtemps passent en premier, ça garantit de trier progressivement les tenants rapides en premier.

#36 - 11 octobre 2022 13:18 - Frédéric Péters

> Cette partie va fonctionne avec LOCK, oui, mais j'ai plusieurs interrogations :

Ok donc c'est bon pour oublier toute la partie on conflict update.

> où est-ce qu'on forke

Le cron est lancé régulièrement, la première fois qu'il est lancé il y a un processus, la deuxième fois, si le processus passé n'est pas encore terminé, on se trouve avec deux processus, etc. cf plus haut, #33280#note-15.

> je ne vois pas l'implémentation du commentaire plus haut

C'était un commentaire de ma part, avant d'écrire le code, et en écrivant le code j'ai jugé qu'on pouvait s'en passer.

> la façon dont tout est remis à needed si tout le monde est done et personne en running ou needed me chagrine.

Je comprends mais pour le moment je préfère en rester à cette version, mesurer les résultats, itérer.

#37 - 11 octobre 2022 15:11 - Benjamin Dauvergne

Frédéric Péters a écrit :

> > la façon dont tout est remis à needed si tout le monde est done et personne en running ou needed me chagrine.
>
> Je comprends mais pour le moment je préfère en rester à cette version, mesurer les résultats, itérer.

Ok donc si le but n'est pas de distribuer les tenants sur plusieurs CPU, c'est bien trop compliqué, autant utiliser un simple verrou.

#38 - 11 octobre 2022 16:12 - Frédéric Péters

Je ne comprends pas cette phrase, dans le contexte de relecture, ça dit oui ok c'est bon comme ça, ou autre chose ?

#39 - 11 octobre 2022 16:13 - Benjamin Dauvergne

Frédéric Péters a écrit :

> Je ne comprends pas cette phrase, dans le contexte de relecture, ça dit oui ok c'est bon comme ça, ou autre chose ?

Ça veut dire que je laisse la place à quelqu'un qui trouve ce ticket utile, là moi je ne vois pas.

#40 - 13 octobre 2022 08:24 - Benjamin Dauvergne

Benjamin Dauvergne a écrit :

> Ok donc si le but n'est pas de distribuer les tenants sur plusieurs CPU, c'est bien trop compliqué, autant utiliser un simple verrou.

Si cette remarque n'est pas comprise : le fait de permettre plusieurs exécution concurrentes de la commande cron ne demande pas cette algo compliqué avec 3 statuts que je ne comprends même pas, ni de vérifier CRON_MAX_WORKERS ainsi, un simple verrou de la forme suivante suffirait (en pseudo-python code) et c'est plus facile de s'assurer que ça marche formellement et d'obtenir des alertes sur les tenants trop long ou si on saute des slots de 20 minutes parce que tous les workers autorisés sont déjà pris (encore qu'en fonctionnant sur deux noeuds ce n'est pas évident non plus de voir si on saute ou pas un créneau).

```
@guard_postgres
def is_locked(name):
    conn, cur = get_connection_and_cursor()
    cur.execute('LOCK wcs_meta')
    key = f'lock-{name}'
    cur.execute('SELECT value, updated FROM wcs_meta WHERE key = %s', [key])
    row = cur.fetchone()
    conn.commit()
    return row is not None

@guard_postgres
def try_lock(name):
    conn, cur = get_connection_and_cursor()
    cur.execute('LOCK wcs_meta')
    key = f'lock-{name}'
    cur.execute('SELECT value, updated FROM wcs_meta WHERE key = %s', [key])
    row = cur.fetchone()
    if row is not None:
       conn.commit()  # unlock wcs_meta
       return False, row[0], row[1]

    value = f'host {socket.getfqdn()} pid {os.getpid()}'  # for debug
    cur.execute('INSERT INTO wcs_meta (key, value, updated) VALUES (%s, %s, NOW())', [key, value])
    conn.commit()
    return True, None

@guard_postgred
def unlock(name):
    conn, cur = get_connection_and_cursor()
    key = f'lock-{name}'
    cur.execute('DELETE FROM wcs_meta WHERE key = %s', [key])
    conn.commit()
....

lock_count = 0
for tenant in tenants:
    # set the tenant
    if is_locked():
        lock_count += 1

if lock_count > CRON_MAX_WORKERS:
    alert('all workers slot already used')
    return

for tenant in tenants:
    # set the tenant
    locked, value, timestamp = try_lock('cron')
    if locked:
       try:
            cron_worker()
       finally:
```

```
            unlock('cron')
    else:
         if now() - timestamp > timedelta(hours=6):
             alert_running_too_long(f'{tenant} cron locked for more than 6 hours by {value}')
```

#41 - 13 octobre 2022 12:18 - Frédéric Péters

Je vois mais une préoccupation était pour le deuxième (et suivant) processus de reprendre "là où on en était", vs cette proposition qui, si je comprends bien, va donner un avantage particulier aux "premiers tenants", qui auraient leur exécution garantie régulière, alors que les "derniers tenants", pas.

Alors j'imagine qu'on pourrait ajouter un random.shuffle() des tenants pour annuler ça (mais pour le suivi/debug, j'aime quand même bien la garantie d'ordre).

Par rapport à mon code proposé, je peux voir la préoccupation comme quoi c'est dommage d'attendre vraiment le tout dernier tenant avant d'autoriser des processus à repartir, qu'on finit sur un seul processus, avec tout le monde qui attend le tenant lent; sans tout reprendre ça pourrait être évité en faisant le reset des états une fois qu'il y a zéro à "needed" et le nombre de "running" inférieur au nombre de workers max.

#42 - 13 octobre 2022 15:03 - Benjamin Dauvergne

Frédéric Péters a écrit :

> Je vois mais une préoccupation était pour le deuxième (et suivant) processus de reprendre "là où on en était", vs cette proposition qui, si je comprends bien, va donner un avantage particulier aux "premiers tenants", qui auraient leur exécution garantie régulière, alors que les "derniers tenants", pas.
>
> Alors j'imagine qu'on pourrait ajouter un random.shuffle() des tenants pour annuler ça (mais pour le suivi/debug, j'aime quand même bien la garantie d'ordre).

Quel importance à l'ordre d'exécution entre tenant ? C'est juste au sein d'un même tenant que ça a de l'importance (en plus on sépare les logs là).

> Par rapport à mon code proposé, je peux voir la préoccupation comme quoi c'est dommage d'attendre vraiment le tout dernier tenant avant d'autoriser des processus à repartir, qu'on finit sur un seul processus, avec tout le monde qui attend le tenant lent; sans tout reprendre ça pourrait être évité en faisant le reset des états une fois qu'il y a zéro à "needed" et le nombre de "running" inférieur au nombre de workers max.

J'ai du mal à imaginer le comportement même si je comprends bien que ça va repartir plus vite. Ça reste compliqué et ça a la même conséquence que mon code, l'ordre va totalement changer entre tenant; l'état needed n'est juste pas nécessaire.

Du coté de mon code plutôt qu'un shuffle on peut laisser la valeur de la clé cron-lock mais la mettre à vide, voulant dire non verrouillé et classer les tenants par la clé la plus ancienne (timestamp au moment de la pose du verrou) ça donne l'avantage à celui qui a commencé à s'exécuter il y a le plus longtemps (i.e. toodego attendra 3h comme avant, puisque de toute façon ça prend 3h mais dès qu'il a fini il peut recommencer au prochain cron en premier, c'est lui le plus vieux).

#43 - 13 octobre 2022 15:27 - Frédéric Péters

> Quel importance à l'ordre d'exécution entre tenant ?

Ce que j'écrivais juste au-dessus : cette proposition [...] va donner un avantage particulier aux "premiers tenants", qui auraient leur exécution garantie régulière, alors que les "derniers tenants", pas. La question est pourquoi cette tentative d'égalité de traitement est importante ? (je pense répondre en reformulant ainsi, sur le principe).

> au sein d'un même tenant que ça a de l'importance
> l'ordre va totalement changer entre tenant;

Mais je pense que je ne comprends pas ce que tu appelles cet "ordre entre tenant" et "au sein d'un même tenant".

> J'ai du mal à imaginer le comportement

On imagine 4 workers et dix tenants (t0 à t9), aujourd'hui on peut avoir à la fin uniquement le t7 qui tourne (parce qu'il prend plus de temps que les autres). Le comportement proposé c'est qu'on se remette à traiter les tenants passés (= passer de "done" à "needed") dès qu'il y a moins que 4 tenants "running".

~~

J'aimerais vraiment arriver à avancer sur base de ce qui est dans la branche, plutôt que repartir de loin (le comportement de la branche étant annoncé il y a trois mois #33280#note-2).

#44 - 13 octobre 2022 15:32 - Benjamin Dauvergne

Frédéric Péters a écrit :

> J'aimerais vraiment arriver à avancer sur base de ce qui est dans la branche, plutôt que repartir de loin (le comportement de la branche étant annoncé il y a trois mois #33280#note-2).

Je n'ai jamais dit que je relirai ou que j'avais lu à ce moment là. Je ne suis pas d'accord avec cet approche, mais comme Thomas et Agate étaient intéressés au début, ils peuvent relire si ma relecture n'est pas jugée constructive (ou quelqu'un d'autre). Si j'ai juste le choix entre dire oui ou oui, autant ne pas participer.

#45 - 13 octobre 2022 16:11 - Frédéric Péters

Clairement noter qu'il faut tout reprendre, après qu'on ait déjà avancé à plusieurs, j'ai du mal, mais je voyais un espace de discussion possible entre "faut tout reprendre" (ma caricature de ce que tu écris) et "juste dire oui" (ta caricature de ce que je demande).

#46 - 13 octobre 2022 16:37 - Benjamin Dauvergne

- Assigné à changé de Frédéric Péters à Thomas Noël

Thomas vient de me dire sur jabber qu'il allait relire et de lui assigner, chose faite.

#47 - 18 octobre 2022 09:43 - Pierre Ducroquet

J'ai relu la partie SQL, et je continue de ne pas voir l'intérêt du LOCK wcs_meta alors qu'il s'agit d'un bazooka pointé sur nos pieds (par exemple, en cas de coupure réseau pendant qu'on tient le lock...). Le but est de gérer la concurrence, mais au final on ne fait que créer une contention sur la base de données avec cette solution.
Le code suivant devrait faire le boulot tout aussi bien, mais avec un lock sur la ligne uniquement. Et il faut ajouter dans def do_meta_table une unicité sur la colonne key, c'est un oubli dangereux actuellement.

```
@guard_postgres
def get_and_update_cron_status():
    conn, cur = get_connection_and_cursor()
    do_meta_table(conn, cur, insert_current_sql_level=False)
    key = 'cron-status-%s' % get_publisher().tenant.hostname
    cur.execute("SELECT value FROM wcs_meta WHERE key = %s FOR UPDATE", (key,))
    row = cur.fetchone()
    if row is None:
        cur.execute("INSERT INTO wcs_meta (key, value) VALUES (%s, 'running') ON CONFLICT DO NOTHING", (key,))
        if cur.rowcount != 1:
            # since we could not insert, it means somebody else did meanwhile, and thus we can assume it's run
ning
            status = 'running'
        else:
            status = 'needed'
    elif row[0] == 'needed':
        cur.execute("UPDATE wcs_meta SET value = 'running', updated_at = NOW() WHERE key = %s", (key,))
        status = 'needed'
    else:
        status = row[0]
    conn.commit()
    cur.close()
    return status
```

#48 - 18 octobre 2022 14:46 - Frédéric Péters

De ce que j'avais lu le lock était juste là le temps de la transaction, et sur la table uniquement (ce qui faisait que ça bloquait rien d'autre); j'ai tapé ton code dans la branche.

#49 - 19 octobre 2022 03:12 - Thomas Noël

Relu, et oui il y a clairement des axes de progression possible (typiquement pour l'ordre ça serait chouette de relancer en priorité les cron qui ont été terminés y'a longtemps, ce genre de truc).

Mais le code me semble encore lisible à ce stade.

Je pense quand même à un problème, c'est que mon idée de CRON_WORKERS dépendant de os.cpu_count() va faire que ça peut changer d'un node à l'autre, les machines n'ayant pas toutes forcément le même nombre de CPU. J'ai un peu de mal à voir ce que ça donnerait dans ce cas, en tout l'algo ne sera plus uniforme et ça me semble aller au dérapage possible.

J'imagine que pour s'assurer qu'une machine avec plus de CPU puisse manger plus de cron, il faudrait compter "ses" cron, donc retenir par quelle machine un cron est lancé ("running-on-<machine-id>").

A décider :

- soit on laisse le os.cpu_count()

- soit on fixe un CRON_WORKERS = 8 par défaut et on le monte dans le settings.d des machines solides (genre 16 sur notre SaaS à 2*32 processeurs) -- sans doute le plus simple aujourd'hui

... et dans les deux cas, on prépare un ticket pour le cas des nodes déséquilibrés (ça arrivera)

#50 - 19 octobre 2022 03:43 - Thomas Noël

- Assigné à changé de Thomas Noël à Frédéric Péters

#51 - 19 octobre 2022 07:43 - Frédéric Péters

> J'ai un peu de mal à voir ce que ça donnerait dans ce cas, en tout l'algo ne sera plus uniforme et ça me semble aller au dérapage possible.

Je ne pense pas (du tout), actuellement le seul endroit où le nombre de workers est utilisé est :

```
                if len(tenant_status['running']) >= settings.CRON_WORKERS:
                    if cron_status == 'needed':
                        # unmark current tenant as being running
                        sql.mark_cron_status('needed')
                    if verbosity > 1:
                        print(hostname, 'skip running, too many workers')
                    break
                if cron_status in ('running', 'done'):
                    if verbosity > 1:
                        print(hostname, 'skip running, already handled')
                    continue
```

c'est-à-dire que la seule action possible est sur le seul unique tenant qui "dépasse", où on assure qu'il reste en "needed", action pour laquelle je ne vois pas de conséquence.

Surtout/aussi, on est toujours sur une exécution sur un seul nœud, (même si par la suite on pourrait), on n'a rien changé ici,

```
        if getattr(settings, 'DISABLE_CRON_JOBS', False) and not options['force_job']:
            if verbosity > 1:
                print('Command is ignored because DISABLE_CRON_JOBS is set in settings')
            return
```

> ... et dans les deux cas, on prépare un ticket pour le cas des nodes déséquilibrés (ça arrivera)

Ça pourra être une considération dans le ticket qui demandera d'ignorer DISABLE_CRON_JOBS.

#52 - 20 octobre 2022 01:25 - Thomas Noël

- Statut changé de Solution proposée à Solution validée

Ok, laissons alors le os.cpu_count()//2+1 !

#53 - 20 octobre 2022 20:55 - Frédéric Péters

- Statut changé de Solution validée à Résolu (à déployer)

```
commit f78f46ed2d1bc8145dc6bc1df6886fad074c1514
Author: Frédéric Péters <fpeters@entrouvert.com>
Date:   Thu Aug 4 09:09:08 2022 +0200

    cron: allow some cron workers to be run in parallel (#33280)
```

#54 - 20 octobre 2022 22:14 - Transition automatique

- Statut changé de Résolu (à déployer) à Solution déployée

#55 - 25 décembre 2022 04:41 - Transition automatique

Automatic expiration

Fichiers

0001-cron-allow-some-cron-workers-to-be-run-in-parallel-3.patch	29,4 ko	30 août 2022	Frédéric Péters
0001-cron-allow-some-cron-workers-to-be-run-in-parallel-3.patch	29,4 ko	09 septembre 2022	Frédéric Péters