Saurez-vous sauver cette instance ?

Configuration PostgreSQL

1.2

Table des matières

ATELIER - TP

Les VM hébergeant les instances PostgreSQL disposent de 8 Go de RAM et 4 CPU.

Afin de faciliter la mise en place de cet atelier, les paramètres suivants ont été mis en place dans le fichier `postgresql.conf`. Ils ne doivent pas être modifiés.

```
log_line_prefix = '%t [%p]: user=%u,db=%d,app=%a,client=%h '
log_checkpoints = on
log_connections = on
log_disconnections = on
log_lock_waits = on
log_temp_files = 0
log_autovacuum_min_duration = 0
log_error_verbosity = default
log_min_duration_statement = 0
lc_messages = 'C'
max_parallel_workers_per_gather = 0
jit = off
```

Optimisation de l'instance

Vérifier les paramètres de bases de l'instance et les corriger si nécessaire. Voici une liste non exhaustive de paramètres importants à vérifier :

- autovacuum
- shared_buffers
- maintenance_work_mem
- autovacuum_work_mem
- checkpoint_completion_target
- effective_cache_size
- random_page_cost
- effective_io_concurrency
- maintenance_io_concurrency

Optimisation des requêtes

Requête 1

```
SELECT * FROM users u
JOIN votes v ON (u.id = v.userid) AND v.votetypeid = 5
JOIN votes v2 ON (u.id = v2.userid) AND v2.votetypeid = 6
JOIN votes v3 ON (u.id = v3.userid) AND v3.votetypeid = 8
JOIN votes v4 ON (u.id = v4.userid) AND v4.votetypeid = 10
JOIN votes v5 ON (u.id = v5.userid) AND v5.votetypeid = 4;
```

- Afficher le plan d'exécution de la requête.
- Vérifier la présence des index. Sont-ils suffisants ou pertinants ?
- Vérifier la configuration de PostgreSql et le nombre jointure.

Requête 2

```
select * from posthistory where extract('year' from creationdate) = 2010;
```

- Afficher le plan d'exécution de la requête.

Le champ `creationdate` est de type `timestamp without time zone` et aucun index n'est créé sur celui-ci. Nous allons donc le créer pour voir si cela améliore le comportement :

```
create index on posthistory (creationdate);
```

- Afficher le plan d'exécution de la requête
- L'index n'est pas utilisé, pourquoi ? Quel index doit-on alors utiliser ?

Requête 3

```
SELECT COUNT(ph.posthistorytypeid) AS nb, pht.name AS name
FROM posthistory ph JOIN posthistorytypes pht ON (ph.posthistorytypeid = pht.id)
WHERE ph.creationdate < '2011-01-01' GROUP BY 2;
```

- Afficher le plan d'exécution de la requête.

Rien à signaler sur la table `posthistorytypes`, il y a bien un index sur le champ `id` car c'est une clé primaire. En revanche, sur la table `posthistory`, aucun index n'est présent sur le champ `creationdate`.

Voyons si la création de celui-ci change quelque chose :

```
create index on posthistory (creationdate);
```

Le temps de réponse est encore élevé, regardons s'il n'est pas possible de faire mieux.

- Quel index pourrait-on utiliser pour améliorer le temps d'exécution de cette requête ?

Requête 4

```
SELECT * FROM interaction WHERE interac = 22;
```

- Afficher le plan d'exécution de la requête.
- Vérifier le taux de fragmentation de l'index `interaction_interac_idx` pour voir son niveau de fragmentation (`bloat`).

La requête suivante est disponible pour avoir une estimation du niveau de bloat des index : requête bloat[1]. Cette requête nécessite d'être `superuser` et d'avoir des statistiques à jour.

- Réindexer `interaction_interac_idx`

Le parcours de l'index est-il plus rapide ?

Requête 5

```
select id from posts where closeddate is not null;
```

- Afficher le plan d'exécution de la requête.

Aucun index n'est créé sur le champ `closeddate`.

- Créer cet index.

```
create index on posts (closeddate);
```

Pas d'amélioration obtenu avec cet index.

- Créer un index partiel sur le champ `id'` avec une clause sur le champ `closeddate`.

[1] https://github.com/ioguix/pgsql-bloat-estimation/blob/master/btree/btree_bloat-superuser.sql

Requête 6

```
SELECT id FROM posts WHERE title LIKE 'pi%';
```

- Afficher le plan d'exécution de la requête.

Cette requête effectue une recherche sur le motif `pi%` en utlisant un `Seq Scan`.

Le champ `title` est de type `text` et n'est pas indexé. Créer l'index suivant.

```
create index on posts (title);
```

Aucun changement, l'index n'est même pas utilisé.

- Créer l'index suivant.

```
create index on posts (title text_pattern_ops);
```

Le gain est énorme. L'index est cette fois bien utilisé.

Requête 7

```
SELECT * FROM comments WHERE creationdate BETWEEN '2021-01-01' AND '2021-02-01';
```

- Afficher le plan d'exécution de la requête.

Il y a bien un index sur le champ `creationdate`. Cependant, en regardant de plus près on remarque le mot clé `INVALID` qui indique que celui-ci n'est pas utilisable. Ce problème peut aparaitre notamment lors des opérations de création d'index ou de réindexation avec la clause `CONCURRENTLY`.

La requête suivante permet de remonter l'ensemble des index invalides d'une base de données :

```
SELECT  indrelid::regclass, indexrelid::regclass FROM pg_index WHERE indisvalid ='f';
 indrelid |        indexrelid
----------+--------------------------
 comments | comments_creationdate_idx
```

On y retrouve bien l'index `comments_creationdate_idx` sur la table `comments`.

- Corriger le problème.

Requête 8

```
select * from comments where jsonfield is not null;
```

- Afficher le plan d'exécution de la requête.

Aucun index sur le champ `jsonfield`. Est-ce qu'un index classique sur ce champ permettrait d'améliorer la requête ?

- Quel index peut-on utiliser pour améliorer les performances de cette requête ?

Requête 9

```
SELECT u.displayname AS name, max(c.creationdate) AS date_last_comment
FROM comments c JOIN users u ON (c.userid = u.id)
WHERE userid = 664306 GROUP BY 1;
```

- Afficher le plan d'exécution de la requête.Affichons le plan :

Le principal problème ici vient du `Seq Scan` réalisé sur la table `comments` pour répondre à la clause `WHERE`. Seulement 6 lignes sont retournées et 5745561 sont supprimées par le clause `WHERE`.

Aucun index n'est présent sur le champ `userid`. La source de nos lenteurs vient donc à priori de là.

- Créer l'index.

```
create index on comments (userid);
```

Requête 10

```
SELECT DISTINCT location, age, displayname
FROM users
WHERE creationdate > '2021-01-01'
ORDER BY 1, 2;
```

- Afficher le plan d'exécution de la requête.Affichons le plan :

Un index est bien utilisé pour la clause WHERE et il n'y a pas de jointure. La piste d'un index manquant est donc à écarter.

- Afficher un plan plus détaillé avec la requête suivante :

```
explain (analyze,buffers) SELECT DISTINCT location, age, displayname
FROM users
WHERE creationdate > '2021-01-01'
ORDER BY 1, 2;
```

On peut constater que des données sont écrites sur disques : `Disk: 2192kB`. Cette écriture est provoquée par la clause `ORDER BY` de la requête.

- Quel paramètre doit-on regarder et modifier pour améliorer la performance de cette requête ?

Requête 11

```
select * from users where upper(displayname) = 'FRANZ';
```

- Afficher le plan d'exécution de la requête.Affichons le plan :

La requête réalise ici un `Seq Scan` et filtre énormement de lignes pour peu de résultats retournés. La piste d'un index manquant ou non utilisé est donc à prioriser.

Un index est bien présent, mais non utilisé.

- Quel index créer pour améliorer les performances de cette requête ?

Requête 12

```
SELECT displayname AS name, location FROM users WHERE location LIKE 'Reims%';
```

- Afficher le plan d'exécution de la requête.Affichons le plan :

Nous sommes ici dans un cas similaire à la requête 6. L'utilisation du mot clé `LIKE` dans la requête rend inutilisable un index utilisant la classe d'opérateur définit par défaut.

Aucun index n'est présent sur le champ `location`. Il faut donc en créer un mais en modifiant la classe d'opérateur pour qu'il puisse être utilisé par la clause `LIKE`.

- Créer l'index adéquat.

Requête 13

```
select * from users where age * age > 2500;
```

- Afficher le plan d'exécution de la requête.Affichons le plan :

Aucun index sur le champ age. Comme pour les requêtes 2 et 11, la clause WHERE filtre sur le résultat d'une fonction.

- Créer l'index fonctionnel correspondnat.