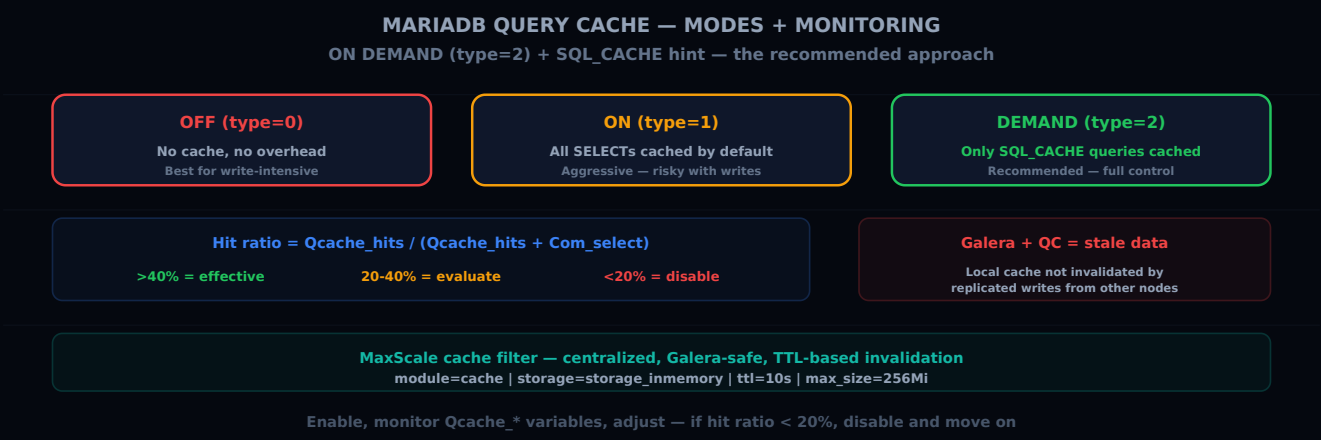
Expérimentation MariaDB : le query cache

Sylvain ARBAUDIE · 30 juin 2025

MARIADB QUERY-CACHE PERFORMANCE TUNING



Le concept du Query Cache

Le query cache de MariaDB / MySQL est un mécanisme intégré au serveur qui stocke le résultat de requêtes SELECT en mémoire. Quand une requête identique est soumise à nouveau, le serveur retourne directement le résultat en cache sans exécuter la requête. C'est simple, élégant, et potentiellement très efficace.

Le mot clé ici est "potentiellement". Le query cache est l'une des fonctionnalités les plus mal comprises et les plus mal configurées de MariaDB / MySQL. Utilisé correctement, il peut diviser les temps de réponse par 10. Mal configuré, il peut détruire les performances.

Les trois modes

Le query cache peut fonctionner dans trois modes, contrôlés par la variable `query_cache_type` :

OFF (0)

Le query cache est complètement désactivé. Aucune requête n'est mise en cache, aucune vérification de cache n'est effectuée. C'est l'option la plus sûre pour les charges de travail en écriture intensive.

ON (1)

Toutes les requêtes SELECT sont mises en cache par défaut, sauf celles marquées avec le hint `SQL_NO_CACHE`. C'est le mode le plus agressif.

```
-- Cette requête sera mise en cache
SELECT * FROM products WHERE category_id = 5;

-- Cette requête ne sera PAS mise en cache
SELECT SQL_NO_CACHE * FROM products WHERE category_id = 5;
```

ON DEMAND (2)

Aucune requête n'est mise en cache par défaut. Seules les requêtes explicitement marquées avec `SQL_CACHE` sont mises en cache. C'est le mode le plus contrôlé et souvent le plus efficace.

```
-- Cette requête ne sera PAS mise en cache (mode par défaut)
SELECT * FROM products WHERE category_id = 5;

-- Cette requête SERA mise en cache
SELECT SQL_CACHE * FROM products WHERE category_id = 5;
```

Le mode `ON DEMAND` est celui que je recommande dans la plupart des cas. Il vous force à réfléchir à quelles requêtes méritent d'être cachées, plutôt que de tout mettre en cache aveuglément.

L'invalidation : le talon d'Achille

Le query cache invalide TOUT le cache d'une table dès qu'une écriture est effectuée sur cette table. Pas juste les lignes affectées — toute la table.

```
-- Suppose que 1000 SELECT sur la table 'products' sont en cache
UPDATE products SET price = 19.99 WHERE product_id = 42;
-- → Les 1000 entrées de cache pour 'products' sont invalidées
```

C'est un mécanisme brutal mais nécessaire pour garantir la cohérence. Le problème, c'est que pour les tables fréquemment modifiées, le cache est constamment invalidé et reconstruit, ce qui consomme plus de ressources que de ne pas avoir de cache du tout.

Dimensionner le cache

La taille du query cache est contrôlée par `query_cache_size` :

```
[mysqld]
query_cache_type = 2
query_cache_size = 64M
query_cache_limit = 2M
query_cache_min_res_unit = 2048
```

- **query_cache_size** : taille totale du cache. Ne dépassez pas 256 Mo — au-delà, le mutex global du cache devient un goulot d'étranglement.
- **query_cache_limit** : taille maximale d'un résultat individuel. Les résultats plus gros ne sont pas mis en cache.
- **query_cache_min_res_unit** : taille du bloc d'allocation. Réduire cette valeur pour les résultats petits réduit la fragmentation.

Monitorer le cache

Les variables de statut `Qcache_*` sont essentielles pour évaluer l'efficacité :

```
SHOW GLOBAL STATUS LIKE 'Qcache%';
```

Les métriques clés :

Variable	Description
`Qcache_hits`	Nombre de requêtes servies depuis le cache
`Qcache_inserts`	Nombre de requêtes ajoutées au cache
`Qcache_not_cached`	Requêtes non cachées (trop grosses, hints, etc.)
`Qcache_lowmem_prunes`	Évictions par manque de mémoire
`Qcache_free_memory`	Mémoire libre dans le cache
`Qcache_total_blocks`	Nombre total de blocs alloués
`Qcache_free_blocks`	Blocs libres (fragmentation si élevé)

Le ratio d'efficacité

Le ratio le plus important est le **hit ratio** :

```
Hit ratio = Qcache_hits / (Qcache_hits + Com_select) × 100
```

Interprétation :

- **> 40%** : le cache est efficace, il vaut la peine d'être activé
- **20-40%** : zone grise, évaluer au cas par cas
- **< 20%** : le cache n'est pas efficace, envisager de le désactiver

Un second ratio important est le **ratio d'éviction** :

```
Eviction ratio = Qcache_lowmem_prunes / Qcache_inserts × 100
```

Si ce ratio dépasse 10%, le cache est trop petit — augmentez `query_cache_size` ou réduisez ce que vous cachez.

Le piège de la concurrence

Le query cache utilise un **mutex global**. Cela signifie qu'un seul thread peut lire ou écrire dans le cache à la fois. Sur un serveur avec 100 connexions simultanées, ce mutex devient un goulot d'étranglement sévère.

C'est la raison pour laquelle MySQL 8.0 a supprimé le query cache. L'équipe Oracle a estimé que le coût du mutex dépasse les bénéfices du cache dans les charges de travail modernes (haute concurrence, écritures fréquentes).

MariaDB a choisi de le conserver, estimant qu'il reste utile pour certains cas d'usage spécifiques (charges en lecture, faible concurrence, tables rarement modifiées).

Query cache et Galera : la combinaison difficile

Utiliser le query cache dans un cluster Galera est problématique. Galera réplique les écritures sur tous les nœuds, mais le query cache est local à chaque nœud. Résultat :

1. Nœud A reçoit un SELECT et met le résultat en cache
2. Nœud B reçoit un UPDATE via la réplication Galera
3. Le cache du nœud A n'est PAS invalidé — il ne sait pas que les données ont changé

4. Le prochain SELECT sur le nœud A retourne des données périmées

La seule façon sûre d'utiliser le query cache avec Galera est avec `wsrep_causal_reads = ON`, qui force une vérification de cohérence avant chaque lecture. Mais cela annule largement le bénéfice du cache.

L'alternative : MaxScale cache filter

Pour les architectures qui ont besoin d'un cache de requêtes distribué, le filtre cache de MaxScale est une meilleure approche :

```
[query-cache]
type = filter
module = cache
storage = storage_inmemory
ttl = 10s
max_size = 256Mi
```

Le cache MaxScale est centralisé (au niveau du proxy), ce qui élimine le problème d'incohérence entre nœuds Galera. De plus, MaxScale peut invalider le cache de manière intelligente basée sur le type de requête, pas seulement sur la table modifiée.

Quand activer le query cache

Le query cache est pertinent quand :

- La charge est **principalement en lecture** (> 80% de SELECT)
- Les tables sont **rarement modifiées** (tables de configuration, référentiels)
- La **concurrence est modérée** (< 50 connexions simultanées)
- Les **mêmes requêtes sont répétées** fréquemment (applications web avec cache miss)
- Vous êtes sur un **serveur standalone**, pas un cluster Galera

Le query cache n'est PAS pertinent quand :

- La charge est **mixte lecture/écriture**
- Les tables sont **fréquemment modifiées** (tables transactionnelles)
- La **concurrence est élevée** (mutex global)

- Vous êtes sur un **cluster Galera** (incohérence de cache)

Conclusion

Le query cache de MariaDB est un outil puissant mais délicat. Le mode `ON DEMAND` avec le hint `SQL_CACHE` offre le meilleur contrôle. Le monitoring via les variables `Qcache_*` est indispensable pour évaluer l'efficacité. Et pour les architectures distribuées, le filtre cache de MaxScale est souvent un meilleur choix.

Comme tout outil de performance, la clé est de mesurer. Activez, monitorez, ajustez. Si le hit ratio reste en dessous de 20%, désactivez et passez à autre chose.

Cet article a été initialement publié sur Medium.