MySQL - Optimisation

Optimisation de MySQL

Limitations et inconvénients conceptuels

Avec les tables de type **MyISAM**, MySQL utilise un verrouillage extrêmement rapide (plusieurs lecture / une seule écriture). Le plus gros problème survient quand vous avez un mélange de flux de modifications et des sélections lentes sur la même table. Si c'est un problème sur plusieurs tables, vous pouvez utiliser un autre type de table.

Suite de tests MySQL

crash-me est une page web qui test 450 points, utilisé notamment pour la portabilité. Actuellement vous pouvez vous faire une idée des tests en regardant le code et les résultats dans le répertoire **sql-bench**.

Les scripts de test ont été écrit en perl et utilisent le module **perl DBI**. Vous aurez aussi avoir besoin des pilotes spécifiques DBD de chaque serveurs que vous testez.

La suite de tests est située dans le dossier sql-bench. Pour exécuter la suite de tests, lancer run-all-tests
cd sql-bench
perl run-all-tests –server=server_name

Super Smack est un utilitaire de test, qui peut mettre le serveur à genou, utile pour tester les fortes charges du serveur avant sa mise en production.

Optimisation des commandes SELECT et autres requêtes

- Premièrement, ce qui affecte toutes les requêtes : plus votre système de droits est compliqué, plus vous aurez des baisses de performances.
- Si vous n'avez aucun **GRANT**, MySQL optimisera les vérifications de droit, pour les système volumineux, il est bénéfique d'éviter **GRANT**.
- Si vous n'avez pas de droits de niveau tables ou colonne, le serveur n'a pas à vérifier le contenu des tables **tables_priv** et **columns_priv**.
- Similairement, si vous n'avez pas de limites de ressources, le serveur n'a pas de comptes de ressources à faire.
- si le problème est spécifique à une expression MySQL ou une fonction, vous pouvez utiliser la fonction **BENCHMARK()** pour effectuer un test de performances. la syntaxe est **BENCHMARK(100000,1+1G) ;**

Syntaxe Explain (Obtenir des informations sur SELECT)

EXPLAIN tbl_name ou

EXPLAIN SELECT tbl_name

EXPLAIN tbl_name est un synonyme de **DESCRIBE tbl_name** ou **SHOW COLUMNS FROM tbl_name**

Lorsque vous faite précéder **SELECT** avec **EXPLAIN**, MySQL vous explique comment il va traiter la commande **SELECT**, choisir les tables et index pour les jointures. Avec **EXPLAIN**, vous pouvez identifier les index à ajouter pour accélérer les commandes **SELECT**.Vous devriez souvent utiliser la commande **ANALYSE TABLE** pour mettre à jour les statistiques de cardinalité de vos tables, qui affectent les choix de l'optimiseur.

En général, lorsque vous voulez rendre un **SELECT ... WHERE** plus rapide, la première chose à faire est de voir si vous pouvez ajouter des index. Toutes les références entre les tables doivent normalement être faites avec des index. Vous pouvez utiliser la commande **EXPLAIN** pour déterminer les index utilisés pour le **SELECT**.

Pour aider MySQL à mieux optimiser les requêtes, exécutez **myisamchk –analyze** sur une table après l'avoir remplie avec quelques données consistantes. Cela met à jour une valeur pour chaque partie de l'index qui indique le nombre moyen de lignes qui ont la même valeur. Vous pouvez vérifier le retour de l'exécution d'analyze en faisant **SHOW INDEX FROM nom_de_table** et examiner la colonne **Cardinality**.

Pour trier un index et des données par rapport à un index, utilisez **myisamchk –sort-index –sort-records=1** (si vous voulez trier selon le premier index). Notez, toutefois, que ce tri n'est pas le plus optimal et prendra beaucoup de temps pour une grosse table.**EXPLAIN** affiche la valeur **ALL** dans la colonne type lorsque MySQL utilise n scans de table pour résoudre une requête. Cela arrive par exemple lorsque la table est si petite qu'il est plus rapide d'analyser la table que d'utiliser les index. C'est un cas courant pour les tables de moins de 10 lignes, et de taille de ligne faible.

Pour éviter les scans de grosses tables
ANAYZE TABLE
Pour forcer l'utilisation d'un index
FORCE INDEX.

Conception

MySQL conserve les données et les index dans deux fichiers séparés. De nombreux (et en fait presque toutes) les autres bases mélangent les données et les index dans le même fichier.

- Rendre les tables aussi compact que possible
- réduire la taille des données optimise les performances, l'indexation de colonnes de petites taille prend aussi moins de ressources.
- utiliser les types les plus efficaces et les plus petits possible
- utiliser les types d'entiers les plus petits possibles, par exemple **MEDIUMINT** est préférable a **INT**
- Déclarez des colonnes **NOT NULL** si possibles colonnes de taille variable prennent moins de place sur le disque
- la clé primaire doit être aussi courte que possible, cela rend l'identification des lignes plus efficace.
- Ne créer que des index dont vous avez besoin, les index sont bons pour accélérer les lectures, mais sont plus lents pour les écriture des données
- s'il est probable qu'une colonne a un préfixe unique avec les premiers caractères, il est mieux de n'indexer que ce préfixe.
- il peut être parfois intéressant de séparer en 2 une table qui est scannée très souvent.
- tous les types de colonnes de MySQL peuvent être indexés.
- vous pouvez avec tous les gestionnaire de tables avoir au moins 16 clés et une taille d'index d'au moins 256 octets.
- pour les colonnes **CHAR** et **VARCHAR**, il est possible d'indexer un préfixe de la colonne.
- les moteurs de table **MyISAM** et **InnoDB** supportent aussi l'indexation des colonnes **BLOB** et **TEXT**. Lors de l'indexation, vous devez spécifier une taille pour l'index :

CREATE TABLE test (blob_col BLOB, INDEX(blob_col(10)));

- MySQL peut créer des index sur plusieurs colonnes. Un index peut comprendre jusqu'à 15 colonnes (sur les colonnes de type **CHAR** et **VARCHAR**, vous pouvez utiliser uniquement le début de la colonne pour l'indexation).
- Un index sur plusieurs colonnes peut être compris comme un tableau trié contenant des valeurs créées par concaténation des valeurs des colonnes indexées.

mysql> CREATE TABLE test (
-> id INT NOT NULL,
-> nom CHAR(30) NOT NULL,
-> prenom CHAR(30) NOT NULL,
-> PRIMARY KEY (id),
-> INDEX nom_index (nom,prenom));
l'index nom_index sera utilisé pour les requêtes suivantes :
mysql> SELECT * FROM test WHERE nom="Widenius";
mysql> SELECT * FROM test WHERE nom="Widenius"
-> AND prenom="Michael";
mysql> SELECT * FROM test WHERE nom="Widenius"
-> AND (prenom="Michael" OR prenom="Monty");
mysql> SELECT * FROM test WHERE nom="Widenius"
-> AND prenom >="M" AND prenom < "N";
mais nom_index ne sera pas utilisé pour :
mysql> SELECT * FROM test WHERE prenom="Michael";
mysql> SELECT * FROM test WHERE nom="Widenius"
-> OR prenom="Michael";
Tous les index de MySQL (PRIMARY, UNIQUE et INDEX) sont stockés sous la forme de B-tree.

Cache des tables MyISAM

MyISAM utilise un cache qui garde en mémoire les blocs de tables les plus souvent utilisés. Pour les blocs d'index, un buffer de clés est utilisé. Pour les blocs de données, MySQL n'utilise pas de cache, mais exploite le cache natif du système de fichiers.

Les accès aux caches de clés sont à accès simultanés entre les threads. Vous pouvez configurer plusieurs caches de clés, et assigner différents index de tables spécifiquement.

Pour contrôler la taille du cache : **key_buffer_size**. À 0, le cache est désactivé. Le cache de clé est désactivé si la taille du buffer est top petite.

Sans cache de clé, les fichiers d'index sont lus avec le cache du système de fichier.

Accès au cache : un buffer non modifié est en accès simultanné, s'il est modifié, les threads sont en attente.

Cache de clé multiple : pour assigner un index à un cache spécifique :

CACHE INDEX

Les 2 commandes suivantes assignent les index des tables t1, t2 et t3 au cache de clé appelé hit_cache
mysql> CACHE INDEX t1, t2, t3 IN hot_cache;

```
+------+-------------+-------+-------+
|-Table-|------Op------|-Msg_type-|-Msg_text-|
+------+-------------+-------+-------+
|-test.t1-|-assign_to_keycache-|-status-|---OK---|
|-test.t2-|-assign_to_keycache-|-status-|---OK---|
|-test.t3-|-assign_to_keycache-|-status-|---OK---|
+------+-------------+-------+-------+
```

le cache de clé peut être crée en spécifiant sa taille avec le paramètre
mysql> SET GLOBAL keycache1.key_buffer_size=128*1024;
pour détruire le cahce, lui donner une taille de 0
mysql> SET GLOBAL keycache1.key_buffer_size=0;

Pour un serveur en charge, nous recommandons la stratégie suivante pour le cache de clés :

- Un cache de clés principal qui représente 20% de l'espace alloué pour tous les caches de clés. Il sera utilisé par les tables qui sont le plus sollicitées, mais qui ne sont pas modifiées.
- Un cache de clé minoritaire qui représente 20%, Il sera utilisé pour les tables intermédiaires, qui sont intensivement modifiés, comme des tables temporaires par exemple.

- Un cache de clés secondaire qui représente 60% qui sera le cache par défaut, utilisé pour toutes les autres tables.

Stratégie d'insertion au milieu

Par défaut le système de gestion de cache utilise la stratégie **LRU** (le moins utilisé est remplacé en premier). On peut choisir la stratégie de l'insertion par le milieu.

Lors de l'utilisation de la stratégie d'insertion au milieu, la chaîne **LRU** est divisée en deux parties : une sous-chaîne principale, et une sous-chaîne secondaire. Le point de division entre les deux parties n'est pas fixé, mais le système s'assure que la partie principale n'est pas "trop petite", et qu'elle contient au moins **key_cache_division_limit %** de bloc de cache de clés. **key_cache_division_limit** est un composant d'une variable structurée de cache de clé, et sa valeur peut être modifiée indépendamment pour chaque cache.

Lorsqu'un bloc d'index est lu dans une table, depuis le cache de clé, il est placé à la fin de la sous-chaîne secondaire. Après un certain nombre d'accès, il est promu dans la sous-chaîne principale. Un bloc de la chaîne principale est placé à la fin de la chaîne. Le bloc circule alors dans la la sous-chaîne. Si le bloc reste à la fin de la sous-chaîne suffisamment longtemps, il est rétrogradé dans la chaîne secondaire. Ce temps est déterminé par la valeur du composant **key_cache_age_threshold**.

La stratégie de l'insertion au milieu vous permet de garder les blocs les plus utilisés dans le cache. Si vous préférez utiliser la stratégie **LRU** classique, laissez la valeur de **key_cache_division_limit** à 100.

La stratégie d'insertion au milieu aide à améliorer les performances lorsque l'exécution d'une requête qui requiert un scan d'index place dans le cache toutes les valeurs de l'index. Pour éviter cela, vous devez utiliser la stratégie d'insertion au milieu, avec une valeur très inférieure à 100 pour **key_cache_division_limit**. Les blocs les plus utilisés seront conservés dans le cache durant un tel scan.

Préchargement des index

S'il y a suffisamment de blocs dans le cache de clé pour contenir tout un index, ou au moins les blocs correspondant aux blocs non-terminaux, alors cela vaut la peine de pré-charger l'index avant de commencer à l'utiliser. Le pré-chargement vous permet de mettre les blocs d'index dans un buffer de cache le plus efficacement : il lit les blocs séquentiellement sur le disque.

Pour pré-charger un index dans un cache
LOAD INDEX INTO CACHE
Par exemple, la commande suivante précharge les index des tables t1 et t2
mysql> LOAD INDEX INTO CACHE t1, t2 IGNORE LEAVES ;
L'option
IGNORE LEAVES
fait que les blocs non-terminaux seuls seront lus dans l'index. Par conséquent, la commande ci-dessus va charger tous les blocs de l'index de t1, mais uniquement les blocs non-terminaux de t2
Un cache de clé peut être restructuré à tout moment, en modifiant les valeurs de ses paramètres
mysql> SET GLOBAL cold_cache.key_buffer_size=4*1024*1024 ;
Si vous assignez une nouvelle valeurs aux variables **key_buffer_size** ou **key_cache_block_size**, le serveur va détruire l'ancienne structure du cache, et en recréer un

Quand MySQL ouvre et ferme les tables

table_cache, max_connections et max_tmp_tables affectent le nombre maximum de tables que le serveur garde ouvertes.

table_cache est lié au **max_connections**.

Par exemple, pour 200 connexions simultanées, vous devriez avoir un cache de table d'environ **200*n**, où n est le nombre maximum de table dans une jointure. Vous devez aussi réserver des pointeurs de fichiers supplémentaires pour les tables temporaires et les fichiers. Assurez vous que votre système d'exploitation peut gérer le nombre de pointeurs de fichiers demandé par l'option **table_cache**. Si

table_cache est trop grand, MySQL peut être à court de pointeurs, et refuser des connexions, échouer à l'exécution de requêtes, ou être très instable. Vous devez aussi prendre en compte que les tables MyISAM peuvent avoir besoin de deux pointeurs de fichiers pour chaque table différente. Vous pouvez augmenter le nombre de pointeurs de fichiers disponibles pour MySQL avec l'option de démarrage **-open-files-limit=#**.

Le cache de tables ouvertes reste au niveau de **table_cache** entrées (par défaut, 64 ; cela peut être modifié avec l'option **-O table_cache=#** de mysqld). Notez que MySQL peut ouvrir temporairement plus de tables, pour être capable d'exécuter des requêtes.

Une table qui n'est pas utilisée est refermée, et supprimée du cache de table, dans les circonstances suivantes :

- Lorsque le cache est plein, et qu'un thread essaie d'ouvrir une table qui n'est pas dans le cache.
- Lorsque le cache contient plus de table_cache lignes, et qu'aucun thread n'utilise cette table.
- Lorsque quelqu'un utilise la commande mysqladmin refresh ou mysqladmin flush-tables.
- Lorsque quelqu'un exécute la commande FLUSH TABLES.

Si vous ouvrez une table avec **HANDLER table_name OPEN**, un objet de table dédié sera alloué pour le thread. Cet objet de table n'est pas partagé avec les autres threads, et il ne sera pas fermé avant que le thread n'appelle **HANDLER table_name CLOSE**, ou que le thread ne meurt

Optimiser le serveur mysql

Si vous avez bcp de RAM et que vous voulez des performances maximales, vous pouvez essayer
shell> safe_mysqld -O key_buffer=64M -O table_cache=256 -O sort_buffer=4M -O read_buffer_size=1M &
Si vous n'avez que 128 Mo et seulement quelques tables, mais que vous demandez beaucoup de classements, vous pouvez essayer cela
shell> safe_mysqld -O key_buffer=16M -O sort_buffer=1M
Si vous avez peu de mémoire et beaucoup de connections, essayez cela
shell> safe_mysqld -O key_buffer=512k -O sort_buffer=100k -O read_buffer_size=100k &
Ou encore
shell> safe_mysqld -O key_buffer=512k -O sort_buffer=16k -O table_cache=32 -O read_buffer_size=8k -O net_buffer_length=1K &
Si vous utilisez **GROUP BY** ou **ORDER BY** sur des fichiers de taille supérieure à la mémoire disponible, vous devriez augmenter la valeur de **record_rnd_buffer** pour accélérer la lecture des lignes après que le classement ait été fait

A l'installation de MySQL, un répertoire **support-files** est créé, et contient plusieurs exemples de fichiers **my.cnf** : **my-huge.cnf**, **my-large.cnf**, **my-medium.cnf** et **my-small.cnf**. Vous pouvez les utiliser comme base pour optimiser votre système.

Si vous avez vraiment beaucoup de connections, des problèmes peuvent apparaître avec le fichier d'échange si mysqld n'a pas été configuré pour utiliser peu de mémoire pour chaque connexion. mysqld fonctionne mieux si vous avez suffisamment de mémoire pour toutes les connections.

Pour voir les effets d'un changement de paramètre, essayez
shell> mysqld -O key_buffer=32m –help
MySQL 5.0.1 propose une nouvelle méthode plus souple pour l'optimisation, qui permet à l'utilisateur de contrôler l'exhaustivité de la recherche de l'optimisateur dans sa quête pour la méthode la plus efficace pour traiter une requête. L'idée générale est que plus le nombre de méthodes étudiées est petit, moins l'optimisateur prendra de temps à compiler la requête.

La variable **optimizer_prune_level** indique à l'optimisateur d'omettre des méthodes basées sur l'estimation du nombre de lignes utilisées dans les tables. Notre expérience montre que ce type de "prévision" échoue rarement, tout en réduisant considérablement le temps de compilation des requêtes. C'est pour cela que cette variable est active par défaut.

La variable **optimizer_search_depth** indique la "profondeur" d'analyse de l'optimisateur. Les valeurs les plus faibles de **optimizer_search_depth** peuvent conduire à de grandes différences dans le temps de compilation. Par exemple, une requête avec 12-13 ou plus peut facilement prendre des heures ou des jours à compiler si **optimizer_search_depth** a une valeur proche du nombre de tables à traiter. Mais, si **optimizer_search_depth** vaut 3 ou 4, le compilateur peut traiter cette requête en une minute environ. Si vous n'êtes pas

sûrs de la valeur raisonnable de **optimizer_search_depth**, donnez lui la valeur de 0 pour que l'optimisateur puisse déterminer la valeur automatiquement.

Influence de la compilation et des liaisons sur la vitesse de MySQL

Les exécutables les plus rapides sont obtenus en liant avec **-static**. le code le plus rapide sera obtenu en compilant avec **pgcc** et **-O3**.

Il est aussi possible d'utiliser **CXX=gcc** à la configuration de MySQL pour éviter l'inclusion de la bibliothèque libstdc++ (qui n'est pas nécessaire).

A la compilation de MySQL, vous devriez uniquement utiliser le support des caractères que vous allez utiliser. (Option **–with-charset=xxx**.)

Liste des mesures que nous avons effectués

- L'utilisation de **pgcc** et la compilation complète avec l'option **-O6** donne un serveur mysqld 1% plus rapide qu'avec **gcc 2.95.2**.
- Si vous utilisez la liaison dynamique (sans **-static**), le résultat est 13% plus lent sur Linux.
- Sachez que vous pouvez néanmoins utiliser la liaison dynamique pour les bibliothèques de MySQL. Seul le serveur a des performances critiques.
- Si vous allégez votre binaire mysqld avec l'option **strip libexec/mysqld**, vous obtenez un binaire jusqu'à 4% plus rapide.
- Si vous utilisez TCP/IP plutôt que les sockets Unix, le résultat est 7.5% plus lent sur le même ordinateur. (Si vous vous connectez sur localhost, MySQL utilisera les sockets par défaut.)
- Si vous vous connectez en TCP/IP depuis un autre ordinateur avec un lien Ethernet 100 Mo/s, le résultat sera 8 à 11% plus lent.
- L'utilisation de connections sécurisées (toutes les données chiffrées par le support interne de SSL) pour nos tests comparatifs a provoqué une perte de vitesse de 55%.
- Si vous compilez avec **–with-debug=full**, vous perdrez 20% de performances sur la plupart des requêtes, mais la perte peut être plus importante sur certaines requêtes
- La compilation sur Linux-x86 avec gcc sans les pointeurs **-fomit-frame-pointer** ou **-fomit-frame-pointer -ffixed-ebp** rend mysqld 1 à 4% plus rapide.

Gestion de la mémoire du serveur

- Le buffer de clés (variable key_buffer_size) est partagé par tous les threads. Les autres buffers sont alloués par le serveur suivant les besoins.
- Chaque connexion utilise un espace spécifique au thread :
  - une pile (par défaut, 64 ko, variable thread_stack),
  - un buffer de connexion (variable net_buffer_length),
  - un buffer de résultat (variable net_buffer_length).
  - Le buffer de connexion et celui de résultat sont dynamiquement élargit jusqu'à max_allowed_packet suivant les besoins. Lorsque la requête s'exécute, une copie de la chaîne de requête est aussi allouée.
- Tous les threads partagent la même mémoire de base.
- Chaque requête qui effectue une analyse séquentielle d'une table, alloue un buffer de lecture (variable record_buffer).
- Lors de la lecture de lignes en ordre 'aléatoire' (par exemple, après un tri), un buffer de lecture aléatoire est allouée pour éviter les accès disques (variable record_rnd_buffer).

- Toutes les jointures sont faîtes en une seule passe, et la plupart des jointure sont faîtes sans utiliser de table temporaire. La plupart des table temporaires sont faîtes en mémoire (table HEAP). Les tables temporaires avec beaucoup de données (calculées comme la somme des tailles de toutes les colonnes) ou qui contiennent des colonnes de type BLOB sont sauvées sur le disque.
- La plupart des requêtes qui sont triées allouent un buffer de tri, et entre 0 et 2 fichiers temporaires
- Toute l'analyse et les calculs sont faits en mémoire locale. Aucune mémoire supplémentaire n'est nécessaire pour les petits calculs, et les allocations et libérations de mémoire sont évités. La mémoire n'est allouée que pour les chaînes très grandes
- Chaque fichier d'index est ouvert une fois, et le fichier de données est ouvert pour chaque thread concurrent. Pour chaque thread concurrent, une structure de table, une structure de colonne pour chaque colonne et un buffer de taille 3 * n est alloué (où n est la taille maximale de ligne, en dehors des colonnes de type BLOB). Une colonne de type BLOB utilise 5 à 8 octets de plus que la taille des données du BLOB. Les gestionnaires de table ISAM/MyISAM utilisent un buffer d'une ligne de plus pour leur utilisation interne
- Pour chaque table qui a une colonne BLOB, un buffer est dynamiquement agrandi pour lire les valeurs BLOB. Si vous analysez toute une table, un buffer aussi grand que la plus grande valeur de la colonne BLOB sera alloué.
- Les gestionnaires de tables pour les tables en cours d'utilisation sont sauvées dans un cache, et géré comme une pile FIFO. Normalement, ce cache contient 64 lignes. Si une table doit être utilisée par deux threads concurrents simultanément, le cache contiendra deux entrées pour la table.

MySQL et DNS

- Si votre service DNS est très lent et que vous avez beaucoup d'hôtes, vous pouvez améliorer les performances soit en désactivant le DNS avec **–skip-name-resolve**, soit en augmentant la taille de **HOST_CACHE_SIZE** (par défaut : 128) et en recompilant mysqld
- Il est possible de désactiver le cache de noms d'hôte avec **–skip-host-cache**.
- Il est possible de vider le cache des noms d'hôtes avec **FLUSH HOSTS** ou avec **mysqladmin flush-hosts**.
- Si vous ne voulez pas autoriser les connections par TCP/IP, vous pouvez utiliser l'option **–skip-networking** au démarrage de mysqld.

Problème avec les disques

Utiliser des liens symboliques

Cela signifie que vous allez faire un lien symbolique sur le fichier d'index et/ou le fichier de données sur un autre disque. Cela améliore les lectures et écriture.

Sous Linux, vous pouvez améliorer les performances (jusqu'à 100% en charge n'est pas difficile) en utilisant **hdparm** pour configurer votre interface disque.

La commande suivante doit être une série de bonnes options de hdparm pour MySQL (et probablement d'autres applications)
hdparm -m 16 -d 1
Si vous n'avez pas besoin de savoir quand un fichier a été accédé la dernière fois (ce qui n'est pas utile avec un serveur de base de données), vous pouvez monter votre système de fichier avec l'option
-o noatime
vous pouvez monter des disques avec l'option
-o async
pour que le système de fichiers soit modifié de manière asynchrone.