Olivier DASINI
Forum PHP Novembre 2007

Optimisation de MySQL

http://dasini.net/blog/

Votre conférencier

Olivier DASINI

Formateur certifié MySQL

Consultant technologies Open Sources

olivier@dasini.net

Au programme...

- Introduction
- Pourquoi optimiser ?
- Comment
- Architectures de la base de données
- Créer des requêtes rapides
- Optimisation du serveur MySQL
- Questions / Réponses

Introduction

- Quel SGBD utilisez-vous ?
- Quel système d'exploitation ?
- Versions de MySQL utilisées ?
- Quels sont vos besoins ?

Introduction

- MySQL est le SGBDR open source le plus populaire
- MySQL à une double licence (GPL et commerciale
- Une multitude d'outils (Administrator, Query Browser, Migration Toolkit, mysqlslap, ...)
- S.E. supportés: beaucoup (Linux, Solaris, Windows, FreeBSD, Mac OS X, ...)

Pourquoi optimiser ?

- Tirer le maximum du matériel disponible
- Répondre à un accroissement des données et/ou de la charge
- Plus facile et moins couteux que de recoder l'application
- Satisfaire l'utilisateur final (et le boss :))

Comment

- Écrire des requêtes performantes
  - Avec des index pertinents
  - La différence de performances peut être importante
- Choisir le moteur de stockage adéquat
  - Toutes les données ne se valent pas
  - A chaque besoins, son moteur
- Choisir les bons types de données
  - MySQL possède un choix important de types de données
  - Chaque type à un coût de stockage !

Comment

- Paramétrer de façon optimale son serveur MySQL
  - Changer un paramètre a la fois
  - Les changements ne sont pas forcement immédiats
  - Les gains sont rarement spectaculaires
- Tester et superviser son serveur MySQL
  - Tester avec un jeu de données réaliste
  - Superviser pour détecter et anticiper les problèmes

Architectures de la base de données

- Normaliser (3ème forme normale)
  - Éliminer les données redondantes
  - Avoir des tables plus petites
  - Optimiser les jointures
  - Permettre à l'optimiseur d'être plus performant

Architectures de la base de données

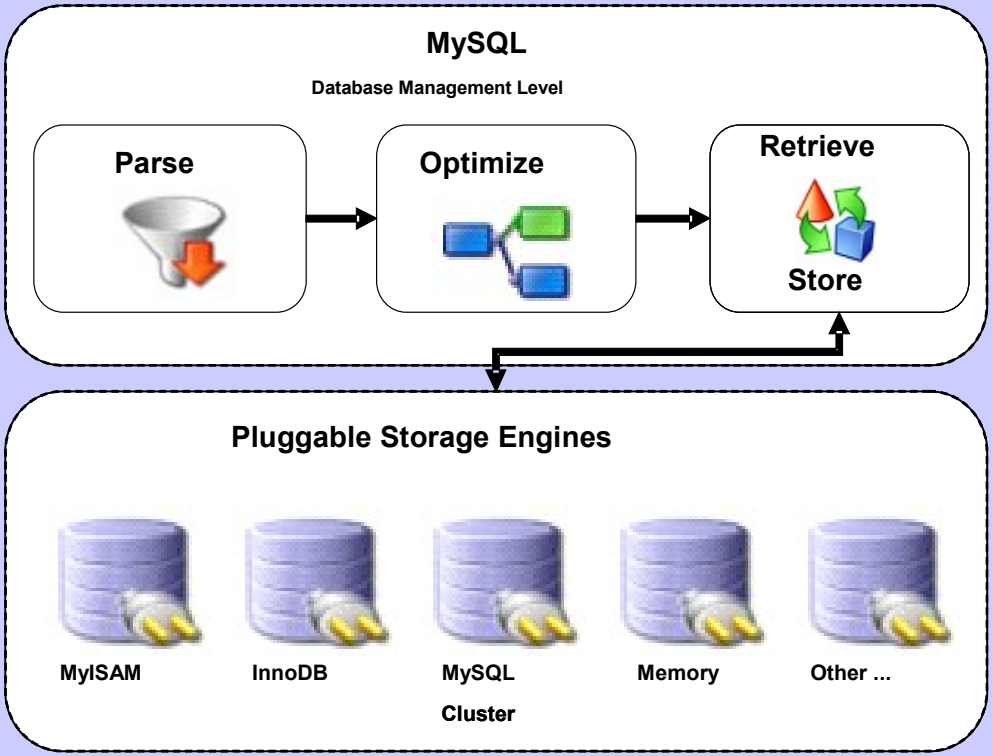
- Dénormaliser
  - Supprimer les jointures (ENUM, SET,...)
    - **CREATE TABLE country (**
      Code char(3) NOT NULL DEFAULT '',
      Continent enum('Asia','Europe','North America','Africa','Oceania','Antarctica' 'South America') NOT NULL DEFAULT 'Asia',
      ...
  - Stocker des données calculée
  - Ajouter des colonnes (auto_increment,...)
  - Table de résumé (snapshot)
    - Problème: gérer les mises à jours
  - Schéma en étoile...

Architectures de la base de données

- Choix du bon type de colonne
  - MySQL à un large choix de types de données
  - Adapter le type à la taille des données
    - code_postal CHAR(5) et non VARCHAR(255)
    - UNSIGNED
  - La clause NOT NULL
    - Risque de problèmes d'intégrité
    - Surcoût inutile pour MySQL
- procedure analyse()
  - ***SELECT col1, col2 FROM ma_table PROCEDURE ANALYSE();***

Architectures de la base de données

- Architecture à moteurs de stockage modulaire



Architectures de la base de données

- Propriétés qui dépendent du moteur de stockage
  - Support de stockage
  - Support des transactions
  - Type de verrou
  - Sauvegarde & restauration
  - Optimisation
  - Fonctionnalités

Architectures de la base de données

- Bien choisir son moteur de stockage
  - MyISAM
  - InnoDB
  - Memory
  - Archive
  - NDBCluster
  - Autres...

Architectures de la base de données

- Bien choisir son moteur de stockage: **MyISAM**
  - Moteur de stockage par défaut
  - Très rapide (insert, select count(*))
  - Verrou niveau table
  - Prend peu de place (40% de moins qu'InnoDB)
  - Index fulltext
  - Possibilité de SELECT & INSERT non bloquant concurrent_insert=2
  - Très portable
  - Statique, Dynamique, compressée

Architectures de la base de données

- Table MyISAM dynamique
  - Contient au moins un champs de type dynamique (CHAR, VARCHAR, TEXT)
  - Prend moins de place sur le disque
  - Fragmentation => OPTIMIZE TABLE
- Table MyISAM statique
  - Plus rapide, plus robuste
  - Pas de fragmentations, mais prend plus de place sur le disque
- Table MyISAM compressée
  - Lecture seule
  - Gain de place de l'ordre de 70%

Architectures de la base de données

- Bien choisir son moteur de stockage: **MyISAM**
  - Moteur non transactionnel
  - Pas de « Crash recovery »
  - Pas de sauvegarde cohérente non bloquante
  - Pas de support des clés étrangères (6.x ?)

Architectures de la base de données

- Bien choisir son moteur de stockage: **InnoDB**
  - Moteur transactionnel ACID
  - Verrou niveau enregistrement
  - Support des clés étrangères
  - Crash recovery
  - Multi-versionning (MVCC)
  - Sauvegarde cohérente non bloquante à chaud (physique & logique)

Architectures de la base de données

- Bien choisir son moteur de stockage: **InnoDB**
  - Pas de support de full-text
  - Sauvegarde physique cohérente non bloquante à chaud payante (InnoDB Hot Backup)

Architectures de la base de données

- Bien choisir son moteur de stockage: **Memory**
  - Données et index stockés en mémoire
  - Très rapide (un accès mémoire 1 million de fois plus rapide qu'un accès disque)
  - 2 algorithmes d'index hash & b-tree (MySQL 4.1+)
  - Utiliser init-file pour peupler les tables Memory au démarrage de MySQL

Architectures de la base de données

- Bien choisir son moteur de stockage: **Memory**
  - Données perdues lors de l'arrêt du serveur
  - Pas de full-text
  - Pas de blob ou de text
  - Pas transactionnel
  - Pas de clés étrangères
  - Verrou niveau table

Architectures de la base de données

- Bien choisir son moteur de stockage: **Archive**
  - Stockage de grands volumes de données en les compressant (70% de réduction)
  - Seulement SELECT, INSERT
  - Pas d'index
  - Verrou niveau ligne

Architectures de la base de données

- Bien choisir son moteur de stockage: **NDBCluster**
  - Moteur transactionnel
  - Haute disponibilité
  - Scalabilité
  - Verrou niveau ligne

Architectures de la base de données

- Bien choisir son moteur de stockage: **CSV**
  - Stock les données au format csv (nom_table.csv)
  - N'accepte pas d'index
  - Non transactionnel
  - Le fichier peut être lu avec
    - OpenOffice.org Calc
    - Microsoft Excel

Architectures de la base de données

- Bien choisir son moteur de stockage: **Autres...**
  - Falcon
    - Transactionnel
    - Verrou ligne
  - SolidDB
    - Transactionnel
    - Verrou ligne
  - PBXT
    - Transactionnel
    - Verrou ligne
  - ... le votre (?)

Architectures de la base de données

- Utiliser plusieurs moteurs de stockage
  - Données statiques en MyISAM
  - Données dynamiques critiques en InnoDB
  - Tables temporaires, tables résumés en Memory
  - Données d'archives (log) en... Archive
  - ...
  - ALTER TABLE <nom_table> ENGINE=<moteur>

Architectures de la base de données

- Utiliser plusieurs moteurs de stockage: inconvénients
  - Configuration plus compliquée
  - Partage de la RAM : moins d'espace pour chaque moteur
  - Jointure entre moteurs transactionnels et non transactionnels

Les index

- But:
  - Accélérer les accès en lecture à la base de données
  - Renforcer la cohérence de la base avec des contraintes
- Critères:
  - Pertinence de l'index (cardinalité, utilisé ?)
  - Unique, non unique ?
  - Alorithme (BTREE / HASH)

Les index: attention !

- Compromis entre vitesse et maintenance
- Risque de ralentissement des requêtes d'ecriture
- Prendre en compte la selectivité : moins les données sont redondantes, plus l'index est efficace
- Ne pas indexer plusieurs fois les mêmes colonnes
- N'indexer que les colonnes nécessaires
- Vérifier la pertinence des index (EXPLAIN)
- Mettre à jours les stats (ANALYZE TABLE)

Les index: Cardinalité

- Plus la cardinalité est élevée, plus un index est selectif, il est donc d'autant plus efficace
- SHOW INDEX FROM ma_table

```
CREATE PROCEDURE `sp_pert_idx`(IN _table CHAR(255), IN _col CHAR(255))

    COMMENT 'Calcul la pertinence d''un index'

BEGIN

  DECLARE req char(255);

  SET @req = CONCAT('SELECT count(*) AS "Total Rows", count(DISTINCT ', _col,')
    AS "Valeurs distinctes", count(*) - count(DISTINCT ', _col, ') AS "Valeurs
    dupliquées" FROM ', _table);

  PREPARE st_pert_idx FROM @req;   EXECUTE st_pert_idx;

  DEALLOCATE PREPARE st_pert_idx;

END $$
```

Algorithmes d'index de MySQL

- B-TREE (Arbre binaire équilibré)
  - Recherche dichotomique
  - Inférieur à log2(N)
  - Log2(1048576) = 20
  - Plusieurs variantes (B+tree, RB-tree, T-tree,...)
- HASH (table memory)
  - Très rapide pour recherche avec égalité (=)
  - Ne fonctionne pas avec inégalité (<, >, between,..)
  - Pas performant avec beaucoup de duplicats

Les types d'index de MySQL

- Primary key
  - unique
  - not null
- Unique
  - unique
  - accepte les NULL
- Index
  - Index simple (pas de contraintes)
- GIS
  - Gestion, génération, stockage et analyse des données spatiales

Les types d'index de MySQL

- Fulltext
  - Uniquement pour les tables MyISAM
  - Pour les champs de type char, varchar et text
  - Optimise la recherche de mots dans du texte
  - MATCH (index_fulltext) AGAINST (mot_recherché)

Les index: astuces

- Index composites
  - INDEX (col1, col2)
    permet d'utiliser l'index pour les colonnes suivantes
    - ... WHERE col1=valx AND col2=valy
    - ... WHERE col1=valx
    - SELECT col2 ... WHERE col1=valx
    - *SELECT col1 ... WHERE col2=valx (full scan index)*
    - *SELECT col2 ... WHERE col2=valx (full scan index)*
  - Mais pas pour
    - **SELECT * ... WHERE col2=valx**

Les index: astuces

- Préfixes d'index
  - Minimiser la taille de l'index
  - Indexer les n premiers caractères d'un champs de type texte
  - Ces n caractères doivent être suffisament discriminants
  - **CREATE INDEX pref_index ON ma_table(col_text(30));**

Les index: astuces

- Préfixes d'index

```
CREATE PROCEDURE `sp_pref_idx`(IN _table CHAR(255), IN _col
   char(255), IN _nbr tinyint unsigned)

    COMMENT 'Determine la pertinence du prefix d''un index'

BEGIN

  DECLARE req CHAR(255);

  SET @req = CONCAT('SELECT count(DISTINCT LEFT(', _col, ', ',
   _nbr,')) AS "Valeurs prefix distinctes", count(*) - count(DISTINCT
   LEFT(', _col,  ', ', _nbr,')) AS "Valeurs prefix dupliquées" FROM ',
   _table);

  PREPARE st_pref_idx FROM @req;

  EXECUTE st_pref_idx;

  DEALLOCATE PREPARE st_pref_idx;

END $$
```

Optimiser ses requêtes

- Comprendre l'optimiseur
  - Élimine le plus d'enregistrements possible
  - Utilise l'index quand cela est possible
  - Évite (autant que possible), les full table scans
  - Cherche l'ordre de jointure des tables optimal
  - Cherche à éviter les accès disque
  - Préfère les accès index aux accès données

Optimiser ses requêtes

- Identifier les requêtes candidates à l'optimisation
  - Slow log (log-slow-queries, log-queries-not-using-indexes, long_query_time)
  - Log general
  - SHOW PROCESSLIST
- Écrire des requêtes simples (si possible)
  - Une requête simple verrouillera moins longtemps les ressources
- Simplifier les expressions
  - SELECT name,population FROM city WHERE population*1.5 > 10000000; => pas index
  - SELECT name,population FROM city WHERE population > 10000000/1.5 ; => index

Optimiser ses requêtes

- Utiliser des tables temporaires (temporary tables)
  - CREATE TEMPORARY TABLE tmp_ma_table
- Utiliser l' extension MySQL REPLACE
  - Correspond selon le contexte, à une clause **INSERT** ou à la suite de clauses **DELETE** + **INSERT**
  - L'enregistrement est supprimé si et seulement si, la clause **INSERT** entraîne une violation d'unicité sur la clé primaire
  - Pas identique à la clause **`UPDATE`** les champs omis prendront comme valeur, la valeur par défaut
- Insertion multi-enregistrements
  - INSERT INTO ma_table VALUES (...),(...),(...);

Optimiser ses requêtes

- Regrouper les requêtes dans des transactions
  - START TRANSACTION;
    ...
    COMMIT;
- Minimiser la tailles des index (surtout la PK pour InnoDB)
- LOAD DATA INFILE (mysqlimport)
- Bannir le SELECT * FROM ...
- Utiliser des index (pertinents)
- Utiliser la commande EXPLAIN

Optimiser ses requêtes: EXPLAIN

- EXPLAIN explique comment MySQL va traiter la commande SELECT
- Donne les informations suivantes:
  - L'ordre de traitement des tables
  - Le(s) index vu par l'optimiseur
  - Le(s) index utilisé(s)
  - Nombre de lignes à analyser
- EXPLAIN SELECT ...

Optimiser ses requêtes: EXPLAIN

- type 1/3:
  - All
    - Lecture entière du fichier de données
    - Le cas à éviter
  - Index
    - Lecture entière du fichier d'index
    - A priori mieux que all

Optimiser ses requêtes: EXPLAIN

- type 2/3:
  - range
    - Recherche par intervalle
    - <, <=, >, >=, ...
  - Index_subquery, unique_subquery
    - Sous requête utilisant un index
  - index_merge (MySQL 5.0 +)
    - Combine différents index

Optimiser ses requêtes: EXPLAIN

- type 3/3:
  - ref_or_null / ref / eq_ref
    - Index non unique « nullable »
    - Index non unique
    - Index unique
  - Const / system
    - Référence à un enregistrement

Optimiser ses requêtes: EXPLAIN

- possible_keys:
  - Index vu par MySQL
- Key
  - Index choisis par MySQL
- Ref
  - Colonne utilisée pour sélectionner les lignes de la table
- Rows
  - Nombre de ligne à analyser (estimation)

Optimiser ses requêtes: EXPLAIN

- Extra (Informations complémentaire sur la stratégie employée par l'optimiseur)
  - Using filesort
  - Using temporary
  - ...
  - Using index

Optimiser ses requêtes

- L'optimiseur est performant, mais il peut se tromper
- Possibilité de le contraindre
  - USE INDEX
  - FORCE INDEX
  - IGNORE INDEX
- STRAIGHT_JOIN

Les requêtes préparées

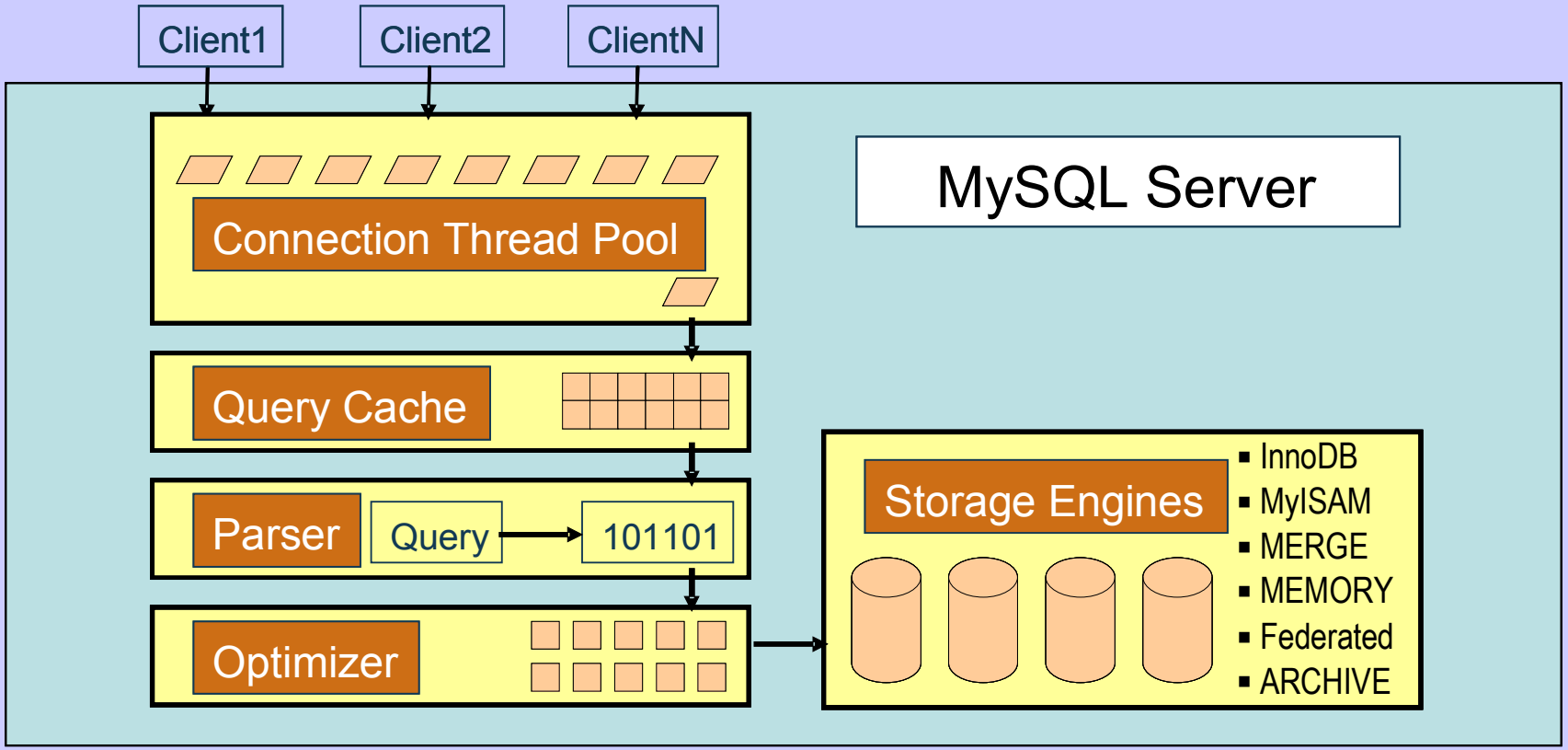
- Préparer une requête en vue de l'exécuter plusieurs fois avec des paramètres différents
- SELECT, INSERT, REPLACE, UPDATE, DELETE, CREATE TABLE, SET, ANALYSE TABLE, OPTIMIZE TABLE, REPAIR TABLE, SHOW...
- L'analyse de la requête n'est faite qu'une seule fois, lors de la préparation de la requête
- La requête préparée n'est visible que pour la session en cours, et sa durée de vie max est celle de la session
- Compatible avec le cache de requête depuis MySQL 5.1.17

Les requêtes préparées

- PREPARE ma_rp FROM 'SELECT name
  FROM city WHERE countrycode = ? AND population > ?';
- SET @cc='fra', @pop=400000;
- EXECUTE ma_rp USING @cc, @pop;

- SET @cc='USA', @pop=800000;
- EXECUTE ma_rp USING @cc, @pop;

Le cache de requêtes



Le cache de requêtes

- Mise en cache des requêtes SELECT et de leur résultat
- Les requêtes doivent être strictement identiques (casse, espace,...)
- Le cache est toujours à jour, car lorsqu'une table est modifiée, les requêtes qui lui font références sont invalidées
- Certaines fonctions empêchent la mise en cache(NOW(), RAND()...)
- Désactivé par défaut, car query_cache_size=0
  - SHOW VARIABLES LIKE 'query_cache%';
    - query_cache_size | 0
    - query_cache_type | ON

Le cache de requêtes

- query_cache_type
  - ON
    - Met en cache les requêtes select
    - Sauf si SELECT SQL_NO_CACHE
  - DEMAND
    - SELECT SQL_CACHE
  - OFF
- query_cache_size
  - Taille du cache de requêtes (en octets)

Le cache de requêtes

- SHOW STATUS LIKE 'qcache%';
- Qcache_hits
  - Nombre de fois que le cache de requête a servi
- Qcache_inserts
  - Nombre de requêtes insérées dans le cache
- Qcache_lowmem_prunes
  - Nombre de requêtes effacées du cache pour laisser place a de nouvelles requêtes (cache trop petit)
- Qcache_not_cached
  - Nombre de requêtes non ''cachable''
- Qcache_queries_in_cache
  - Nombre de requêtes dans le cache

Le cache de requêtes

- FLUSH QUERY CACHE
  - Défragmente le cache de requête
- RESET QUERY CACHE
  - Réinitialise le cache de requêtes
- FLUSH TABLES
  - Réinitialise le cache de requêtes

Optimisation du serveur MySQL

- Rassembler l'information pour savoir quoi optimiser
  - Log général, slow log, log binaire
  - SHOW PROCESSLIST
  - SHOW VARIABLES
  - SHOW STATUS
  - STATUS
  - MySQL Administrator

Optimisation du serveur MySQL: MyISAM

- **Key_buffer_size**
  - Tampon qui stock les index des tables MyISAM
  - 25 à 30% de la RAM, pour un serveur dédié MySQL, en full MyISAM
  - Pas trop grand car risque de swap !
  - Ratio:
    - key_reads / key_read_requests < 0.03 (0.01 encore mieux)
      sinon l'augmenter

Optimisation du serveur MySQL: MyISAM

- **myisam_sort_buffer_size**
  - Tampon pour la création d'index pour les requêtes de maintenance: ALTER TABLE, REPAIR TABLE, LOAD DATA INFILE
  - SET SESSION myisam_sort_buffer_size = 800*1024*1024;
    ALTER TABLE ma_table ADD INDEX ...;
- **bulk_insert_buffer_size**
  - Tampon pour les insertions massives
    - INSERT ...SELECT
    - INSERT VALUES(),(),(),....
    - LOAD DATA INFILE

Optimisation du serveur MySQL: InnoDB

- **innodb_buffer_pool_size**
  - Tampon pour stocker les index et des données des tables InnoDB
  - Jusqu'à 80% de la RAM, pour un serveur dédié MySQL, en full InnoDB
- **innodb_flush_logs_at_trx_commit** (1 par défaut)
  - 0: Risque de pertes de transactions validées en cas de crash d 'InnoDB
  - 1: transactions flushées après chaque commit. Pas de pertes de transactions validées (ACID)
  - 2: Risque de pertes de transactions validées uniquement en cas de crash de l'O.S.

Optimisation du serveur MySQL: InnoDB

- **innodb_log_buffer_size**
  - Taille du tampon des logs d'InnoDB
  - Vidé environ toutes les secondes (checkpoint)
  - En général entre 8Mo & 16Mo

- **innodb_log_file_size**
  - Taille des fichiers de log d'InnoDB (2 par défaut)
  - Une grande valeur améliore les performances Mais augmente le temps de restauration
  - Valeurs courantes: 64Mo à 512Mo

Optimisation du serveur MySQL: Variables globales

- **table_cache**
  - Cache des descripteurs de fichier
  - Chaque table ouverte nécessite un descripteur de fichiers (par connection), plus un pour le .MYI (MyISAM)
  - Augmenter votre table_cache si opened_tables croit rapidement
- **thread_cache**
  - Cache des threads
  - Chaque session prend un thread à la connexion et le rend à la déconnexion
  - Augmenter si threads_created croit rapidement
  - Taux de succès du cache de thread: threads_created/connections

Optimisation du serveur MySQL: Variables spécifiques à la session

- read_buffer_size
  - Tampon d'enregistrements pour les full table scans
- sort_buffer_size
  - Tampon pour GROUP BY / ORDER BY
- tmp_table_size
  - Tampon pour les tables temporaires stockées en mémoire (memory). Au delà, elles sont copiées sur disque (MyISAM)

Optimisation du serveur MySQL: Autres variables à surveiller...

- Indicateur de hautes charges
  - max_used_connections
  - threads_created
  - opened_tables
- Indicateur de réponses lentes
  - slow_queries
  - Slow_launch_threads (Nombre de threads qui ont pris plus de slow_launch_time secondes pour être créés)

Optimisation du serveur MySQL: Autres variables à surveiller...

- Indicateurs de comportements
  - Table_locks_immediate
  - Table_locks_waited
- Les handler
  - Handler_read_first=>lecture d'index
  - Handler_read_key=>lecture d'index
  - Handler_read_next=>lecture d'index
  - Handler_read_prev=>lecture d'index
  - **Handler_read_rnd=>lecture des données**
  - **Handler_read_rnd_next=>lecture des données**

Optimisation du serveur MySQL: Autres variables à surveiller...

- Indicateurs de requêtes:
  - **Tables temporaires**
    - Created_tmp_disk_tables => augmenter tmp_table_size
    - Created_tmp_tables => order by, group by
  - **Requêtes pas optimisées**
    - Select_full_join, Select_full_range_join**:** problèmes d'index dans la jointure
    - Select_scan: full scan sur la premiere table de la jointure
  - **Requêtes coûteuse**
    - sort_merge_passes => augmenter le sort_buffer_size

Optimisation de MySQL...

- Les procédures stockées
- Le partitionnement (MySQL 5.1)
- La réplication
- Le cluster

Aller (encore) plus loin

- Comme JC Vandamme, soyez « **aware** »
- http://www.mysql.com
- http://www.mysqlperformanceblog.com
- http://jpipes.com/index.php
- **http://dasini.net/blog/**

Questions ?