

Audit et optimisation MySQL 5

**Bonnes pratiques
pour l'administrateur**

**Pascal Borghino
Olivier Dasini
Arnaud Gadal**

© Groupe Eyrolles, 2010, ISBN : 978-2-212-12634-1

EYROLLES



1

Gérer une situation d'urgence avec MySQL

Problème majeur ou incident moins dramatique, l'un comme l'autre méritent la plus haute attention. En effet, tous deux sont susceptibles d'être résolus rapidement ou au contraire d'impacter gravement la production. Voici nos conseils pour minimiser les conséquences de ces aléas qui ponctuent la vie d'une base de données.

Il serait illusoire de recenser ici l'ensemble des problèmes et des solutions associées que vous pourriez rencontrer sur un serveur MySQL. En revanche, il est possible de lister quelques bonnes pratiques en fonction du temps dont vous disposez face à un incident en production. Inutile en effet de se lancer dans un audit poussé du système si vous n'avez que quelques minutes pour rétablir une situation compromise. Cependant, avec un peu plus de temps, une heure voire une journée, le mode opératoire diffère complètement. Il est alors possible de rechercher plus longuement les causes du problème et d'activer si possible d'autres ressources en interne ou en externe, par exemple via du support. Bref, analyser le problème afin qu'il ne se reproduise plus.

À chaque degré d'urgence sa panoplie d'outils

Dix minutes, une heure et une journée sont les trois différents degrés d'urgence (arbitraires) que nous vous proposons dans ce chapitre. Cette distinction peut bien sûr être discutée, l'idée étant néanmoins de classer les problèmes par ordre de priorité. À bien y réfléchir, ces ordres de grandeur ne sont finalement pas si éloignés de la réalité.

Les incidents gravissimes (serveur MySQL qui ne redémarre pas, connexion à la base impossible sur un serveur en fonctionnement...) nécessitent une réponse immédiate de votre part car ils ont une incidence directe sur la production. Un délai de résolution de 10 minutes est souhaitable mais laissera malgré tout des traces dans vos tableaux de statistiques mesurant les temps de disponibilité de vos applications.

Légèrement moins urgents, les problèmes que l'on souhaite voir régler dans l'heure ne sont pas à prendre à la légère non plus. En effet, il peut s'agir de difficultés de réplication, de droits absents ou incomplets pénalisant des utilisateurs ou des scripts, de crash de tables, etc.

Enfin, les problèmes de performance sont susceptibles d'appartenir à la catégorie d'incidents dont la résolution, ou tout du moins le diagnostic, ainsi qu'un éventail de solutions adaptées, relève de la journée.

Temps de résolution : dix minutes

Bienvenue à ceux qui savent gérer leur temps ! Joueurs d'échecs, un plus. Voici une annonce qui sortirait sûrement du lot pour une entreprise souhaitant attirer dans ses filets un administrateur de bases de données (DBA). En effet, tel un joueur d'échecs confronté à une partie ultrarapide (5 minutes), le DBA en situation de crise dispose de très peu de temps pour tenter de renverser la situation. En cela, dix minutes constituent à la fois un laps de temps très court à l'échelle d'une journée, par exemple, mais on peut également considérer que c'est amplement suffisant pour détecter l'origine du problème. Nous parlons ici de détection, pas encore de résolution.

Étape 0 : informez et communiquez !

Identifiez la ou les personnes auxquelles vous devez référer en cas de problème. Celle-ci relayera le message si besoin. L'informatique est plus qu'une question de technique : un accident grave sur les bases passe rarement inaperçu et vous oblige à communiquer. Un incident, en particulier s'il est majeur, sera mieux vécu si les utilisateurs concernés par cet incident sont mis au courant plutôt que s'ils sont obligés de contacter eux-mêmes les services techniques pour tenter de comprendre pourquoi leur application ne répond plus. Prenez les devants et annoncez les conséquences : une réplication hors service implique au mieux un état figé des données sur certaines applications et au pire un arrêt de certaines d'entre elles, tout dépend de la robustesse du code sous-jacent.

De plus, indiquer aux utilisateurs qu'il existe un problème leur permet d'attendre votre feu vert avant de renouveler leurs opérations plutôt que de tenter des rafraîchissements ou validations supplémentaires qui n'arrangent rien.

Si vous ne pouvez d'ores et déjà annoncer un délai de rétablissement, dites-le. Outre le fait que les utilisateurs soient avertis au plus tôt, communiquer en amont vous permet également de rester concentré en réduisant le nombre de personnes cherchant à se renseigner auprès de vous (téléphone, messagerie interne, de vive voix...).

Ne restez pas seul et discutez avec d'autres administrateurs

En cas d'extrême urgence, partager ses observations, avec un administrateur système par exemple, est une bonne chose. Si le DBA pense en termes de bases (réplication, droits, tables corrompues), l'administrateur système a souvent une vue complémentaire (place disque, réseau, connectivité aux disques, scripts programmés) ; c'est souvent lui qui a installé, ou tout du moins participé à l'installation, de la machine sur laquelle le serveur MySQL s'exécute. Ses connaissances purement techniques ajoutées à la visibilité qu'il possède sur les machines utilisées par votre département en font un allié de choix en cas de crise.

Consultez les informations système : journal d'erreurs, activités disques et processeur...

Lorsqu'une alerte critique liée à la base de données est levée, une des premières actions à mener, si l'on ne dispose que d'informations vagues du type « système indisponible », est de consulter le journal d'erreurs de MySQL... À condition de pouvoir le faire bien sûr. Si l'un des disques système n'est plus accessible, l'incident bascule tout d'un coup du côté des administrateurs système. Vous êtes seul et occupez les deux fonctions ? Échangez votre casquette de DBA contre celle de l'administrateur système le temps que le problème matériel et/ou système soit résolu.

Les journaux d'erreurs sont un passage absolument obligé lorsqu'un incident relatif à la base de données survient. Sa simple lecture peut faire gagner un temps précieux ! Problème de réseau ou de réplication, arrêt du serveur, table corrompue, syntaxe incorrecte du fichier de configuration, impossibilité de démarrer un moteur de stockage, etc., l'éventail couvert par le fichier d'erreurs est vaste et bien souvent explicite.

Si sa lecture ne vous a pas suffi, ou s'il n'y a aucune information pertinente à l'intérieur, ce qui arrive, quelques commandes système basiques sont à exécuter :

- Tout d'abord `df -kh` pour s'assurer qu'aucune de vos partitions n'est pleine à 100 % ; cela dit le log d'erreurs vous l'aurait signalé.
- `iostat -dx 5` pour obtenir une vue sur l'activité des disques.
- `htop` ou `top` permettent quant à eux de repérer si vos processeurs, ou cœurs pour être plus précis, sont mobilisés et dans quelle proportion.

Espace disque disponible, activité processeur et activité du système disque sont quelques investigations qui doivent normalement contribuer à vous éclairer.

ATTENTION Précautions à prendre avec une table MyISAM corrompue

Le maître mot est ici : sauvegardez ! En effet, la documentation le stipule elle-même : il est possible que les procédures de réparation entraînent une perte de données :

▶ <http://dev.mysql.com/doc/refman/5.1/en/repair-table.html>

Si possible, veuillez donc sauvegarder les tables MyISAM concernées. En cas d'impossibilité de passer par un outil tel que *mysqldump* ou *mysqlhotcopy*, copiez directement les fichiers *.MYD*, *.MYI*, et *.frm* de la table concernée si le serveur est arrêté. Si ce dernier est en marche, la commande `FLUSH TABLES WITH READ LOCK` vous permet d'effectuer l'opération mais cela peut être coûteux en termes de performance.

Le plus important ici est de récupérer le fichier qui contient la base *.MYD*. En effet, si vous connaissez la définition de la table concernée, il sera possible de recréer un *.frm* lui correspondant. Le fichier *.MYI*, quant à lui, peut également être reconstruit : `REPAIR TABLE` utilisé avec l'option `USE_FRM` repartira justement du *.frm* et les index seront totalement créés à nouveau.

Pensez également aux logs binaires. Cette remarque est aussi valable pour un moteur tel que InnoDB. Si la partition sur laquelle se trouvaient vos tables MyISAM est perdue, récupérez vos sauvegardes et les logs binaires, puis réinjectez le tout dans MySQL et vous en serez quitte pour une bonne frayeur.

Attention toutefois à exclure de vos logs binaires l'éventuelle instruction SQL dévastatrice responsable du carnage... À ce sujet, reportez-vous au chapitre 8, consacré à la réplication.

Tentez de vous connecter à la base

Au-delà des outils système, pouvez-vous vous connecter à la base ? Pouvoir effectuer la commande `SHOW FULL PROCESSLIST` est très utile dans ce type de situation. Lorsque la base est chargée, il est efficace de passer par une ligne de commandes pour écrire dans un fichier texte le contenu de cette commande, exemple :

```
shell> mysql -uuser -ppassword -e "SHOW FULL PROCESSLIST;" > /tmp/sfp.txt
```

Il sera alors possible d'éditer facilement l'ensemble des requêtes SQL et de les conserver pour étude, le cas échéant, une fois la tempête passée.

L'utilisation de la commande `SHOW FULL PROCESSLIST` n'est pas forcément chose facile dans tous les cas : encore faut-il pouvoir se connecter !

À SAVOIR Éviter l'empilement des requêtes et décrypter le SHOW PROCESSLIST

Chaque serveur MySQL possède une limite maximale de connexions simultanées. Celle-ci est définie par la variable système `max_connections`. Certains scripts ou applications ne ferment pas correctement leur connexion à MySQL une fois le résultat de celle-ci exploité ; il en résulte un empilement des connexions. Celles-ci arborent alors le statut `sleep` mais occupent malgré tout un emplacement qui ne sera pas utilisable par d'autres clients souhaitant se connecter, si la limite de `max_connection` est atteinte. Pour éviter cela, il est possible d'ajuster la variable `wait_timeout`. Sa valeur par défaut est de 8 heures, ce qui est énorme. En réduisant cette valeur vous indiquerez au serveur MySQL une nouvelle limite au-delà de laquelle il coupera cette connexion si celle-ci est inactive.

Concernant les statuts des requêtes issues d'un `SHOW PROCESSLIST` (*Sending data, Writing to net, Copying to tmp table, etc.*), si certains sont évocateurs, d'autres le sont beaucoup moins. La documentation de MySQL en dresse une liste exhaustive :

▶ <http://dev.mysql.com/doc/refman/5.1/en/general-thread-states.html>

Cependant nous vous conseillons de consulter également la liste, peut-être plus lisible, de Domas Mituzas sur son blog :

▶ <http://mituzas.lt/2009/09/27/mysql-processlist-phrase-book/>

Les statuts y sont expliqués de façon beaucoup plus claire.

Si MySQL vous refuse la connexion en indiquant explicitement *Too many connections*, c'est tout simplement que le nombre défini par `max_connections` a été atteint. Il reste une chance cependant : MySQL autorise bien `max_connections` seulement à se connecter simultanément, mais il existe une connexion supplémentaire, notamment en cas d'urgence.

Cette connexion n'est disponible que pour des utilisateurs disposant du droit `SUPER` ou l'utilisateur `root`. Cet utilisateur avisé pourra alors se connecter à la base même si celle-ci a atteint son nombre maximal théorique de connexions. Rappelons quand même qu'il est déconseillé de laisser une application se connecter à la base sur le compte `root`.

ASTUCE Défilement page par page pour SHOW FULL PROCESSLIST

L'utilisation de la commande `SHOW FULL PROCESSLIST` sur un serveur chargé provoque un défilement illisible, car trop rapide, des nombreuses requêtes exécutées à cet instant sur le serveur MySQL. Il est possible de simuler le comportement du défilement page par page (le `| more` sous Linux, par exemple) grâce à la commande suivante :

```
mysql> pager more;
```

Ainsi, c'est vous qui ferez défiler à votre guise les requêtes affichées par le `SHOW FULL PROCESSLIST`. Pour retrouver le comportement par défaut du client MySQL, saisissez :

```
mysql> nopager;
```

En cas d'échec de ce dernier recours, il ne vous reste plus qu'à redémarrer le serveur MySQL pour récupérer des connexions de libres afin de vous connecter. Si le flux de connexions à la base est tel que vous craignez que la situation ne se renouvelle aussitôt une fois le serveur redémarré, sachez qu'il est possible de restreindre les connexions entrantes à celles qui sont émises localement. L'ajout de la commande `bind-address=127.0.0.1` dans le fichier de configuration `my.cnf`, permet ce type de limitation. `skip-networking` désactive l'écoute des connexions TCP/IP et n'autorise que les connexions via socket sous Unix.

REMARQUE **Se référer aux chapitres concernés**

Nous détaillons le matériel au chapitre 2 et les logs MySQL au chapitre 5 ; c'est pourquoi nous n'aborderons pas à nouveau ici le fonctionnement du RAID ou du log d'erreurs, par exemple. Il en va de même pour d'autres thèmes évoqués dans ce chapitre tels que la réplication ou la mise en place et l'exploitation d'outils de surveillance pour son système.

Supprimer les requêtes les plus lourdes

Que vous soyez l'administrateur des bases de données ou non, difficile de connaître par cœur l'intégralité des utilisateurs MySQL en activité sur les différentes bases.

La dénomination des utilisateurs MySQL rend parfois difficile leur rôle au sein du SI. De plus, certains comptes utilisateurs sont susceptibles d'être utilisés par plusieurs projets qui n'ont pas forcément de liens. Il est du coup difficile de mesurer l'importance d'une requête en la rapportant seulement au nom de l'utilisateur MySQL qui l'exécute.

En cas d'extrême urgence, il est du devoir du DBA de sacrifier certaines requêtes hautement pénalisantes pour l'ensemble du serveur MySQL afin que le site Internet reste disponible. Les internautes sont préservés au détriment de scripts internes qui devront alors être rejoués.

DANS LA VRAIE VIE

Dans un monde idéal, ces deux types d'activités ne devraient pas s'exécuter en même temps ou en tout cas pas sur le même serveur, mais la réalité économique l'emporte parfois sur les règles de sécurité.

Aussi, posséder une liste des utilisateurs dont l'activité est vitale pour l'entreprise et à ne couper sous aucun prétexte (facturation, livraison...) vous permettra de trier le moment venu. Fort de cette liste d'intouchables, les requêtes susceptibles d'être supprimées pour soulager le serveur seront plus facilement identifiables.

PRATIQUE Supprimer des requêtes rapidement

En cas d'urgence, exploiter l'affichage de la commande `SHOW FULL PROCESSLIST` en ligne de commandes via un terminal de type `putty` peut s'avérer délicat. Si le serveur est chargé votre terminal sera submergé de requêtes pour la plupart peu lisibles. Un outil tel que `mytop` permet notamment d'afficher les requêtes par couleur afin de différencier leur temps d'exécution. Il est également possible, à partir de l'interface de cet outil, de supprimer les requêtes de votre choix en indiquant leur identifiant.

MySQL Administrator est une alternative à ce type de script. Son interface graphique permet de naviguer facilement sur l'ensemble des requêtes exécutées sur le serveur. Un tri par utilisateurs MySQL est possible, le nombre d'occurrences du même utilisateur est également indiqué et un bouton permet de supprimer simplement la requête de votre choix ou toutes celles appartenant à un utilisateur.

Si vous n'utilisez pas MySQL Administrator, une alternative est de générer vous-même le fichier des identifiants à supprimer, en utilisant la base `information_schema` :

```
mysql> SELECT concat('KILL ', id, ';') FROM information_schema.processlist
        WHERE user='user_cible' INTO OUTFILE '/tmp/req2del.txt';
```

MySQL génère ainsi un fichier comportant une ligne par requête appartenant à l'utilisateur visé, comme :

```
KILL 272;
KILL 284;
[...]
```

Il ne reste plus qu'à l'exécuter :

```
mysql> source /tmp/req2del.txt;
```

ou

```
shell> mysql -uuser -p < /tmp/req2del.txt
```

À noter : même détruite, une requête peut prendre un long moment avant de rendre la main. En effet, un `ROLLBACK` d'une grosse opération sous InnoDB peut être très long, il n'y a pas de solution idéale dans ce cas là... Si vous redémarrez le serveur vous vous exposez à une récupération de crash d'InnoDB qui peut également prendre un certain temps.

Autre solution extrême à réserver à de gros `SELECT` bloquants sous MyISAM, par exemple : récupérer l'identifiant du thread correspondant à la connexion qui exécute la requête et effectuer un `kill -9`. À vous de mesurer les conséquences d'une telle coupure ; avec MyISAM, il n'y a pas de notions de transactions et la cohérence de la base est donc peut-être compromise à ce niveau.

Encore une fois, testez ces outils par vous-même avant qu'une situation d'urgence ne se déclare. Nos conseils sur la surveillance de vos serveurs occupent le chapitre 4.

Il suffit parfois de supprimer une requête pour débloquer un empilement de connexions préjudiciable pour le serveur MySQL. C'est le cas si cette requête n'est pas optimisée et génère un trop grand nombre de verrous, créant ainsi une file d'attente importante. MyISAM et son verrouillage par table est plus impacté qu'InnoDB mais aucun de ces moteurs de stockage n'est à l'abri de tels phénomènes.

Éviter que l'authentification des utilisateurs repose sur un DNS : l'erreur `unauthenticated user`

Difficile de supprimer toutes les requêtes d'un utilisateur particulier si même MySQL ne sait pas l'identifier... Ce type de message « `unauthenticated user` » est visible en cas de serveur MySQL très chargé ou en proie à un problème de DNS. Nous vous recommandons de ne pas les utiliser lorsque vous définissez un utilisateur MySQL.

Définition d'un utilisateur MySQL basé sur un DNS

```
mysql> GRANT SELECT on *.* TO 'me_dns'@'dev.example.org' IDENTIFIED BY 'password';
```

Le même exemple sans l'utilisation de DNS

```
mysql> GRANT SELECT on *.* TO 'me'@'192.168.10.11' IDENTIFIED BY 'password';
```

En cas de problème du serveur DNS, MySQL ne pourrait identifier le premier utilisateur `'me_dns'`, le qualifiant de `unauthenticated user`. Si vous rencontrez ce problème, il est probable que vous en venez à bout en effectuant ces deux actions :

- bannir les DNS de la définition de vos utilisateurs ;
- démarrer le serveur avec l'option `--skip-name-resolve` (nous vous conseillons d'activer cette option).

Consulter son système de surveillance

Une situation d'urgence ? Voici le bon moment pour capitaliser sur vos efforts précédents et consulter vos logs ou graphiques d'activité liés à la base.

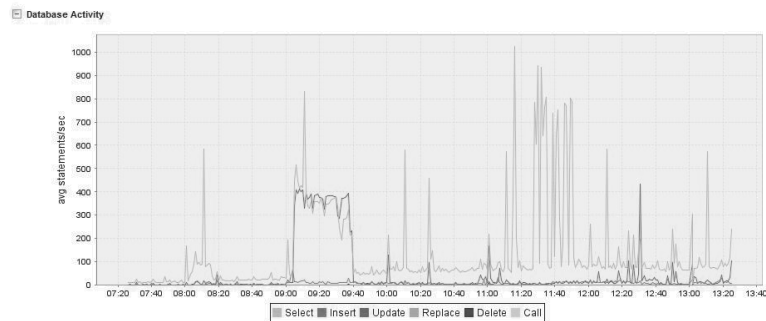
Un graphique est souvent bien plus explicite que différentes métriques à analyser. De plus, selon le degré d'historisation que vous utilisez, il est sûrement possible de remonter (pourquoi pas) à l'heure d'origine du problème. Cette dernière information est cruciale pour incriminer un script programmé dont les impacts ne sont pas forcément simultanés avec son heure de lancement. Pensez par exemple au résultat d'une jointure complexe qui serait stocké par le langage de script afin de générer une boucle de mises à jour mal optimisée, le tout sur une table très volumineuse... L'effet différé est garanti et potentiellement dramatique.

Afin de déterminer si le problème rencontré sur le serveur provient d'une requête ponctuelle passée directement via un client MySQL en production (gestion des droits à revoir !) ou est issu d'un script programmé, un ordonnanceur permet d'identifier rapidement quel script tourne à quelle heure. Son utilisation est moins fastidieuse que le parcours des crontab mais encore faut-il en posséder un. Une fois déter-

minée l'heure probable de l'origine du problème, comparez celle-ci avec celles présentées par l'ordonnanceur.

Figure 1-1

Posséder un historique sur l'activité des bases est d'une grande utilité, exemple avec MySQL Enterprise Manager.



Tranche de vie d'une campagne marketing improvisée

Votre système de surveillance est en alerte, les diodes rouges remplacent frénétiquement leurs paisibles ancêtres de couleur verte bien plus rassurant. Pourquoi ? Les tâches programmées ne sont pas incriminées : les crontab ont été passées au peigne fin. Les développeurs vous jurent qu'ils n'y sont pour rien. Finalement, une fois la tempête passée, quelqu'un a déclenché le lancement d'une campagne marketing reposant sur des pop-ups dynamiques aux requêtes non optimisées. Outre le problème de communication entre les départements technique et marketing, l'incident révèle qu'un seul et même utilisateur peut à lui seul occuper 100 % des connexions disponibles sur votre serveur MySQL.

Afin d'empêcher qu'un seul utilisateur consomme toutes les ressources, il est possible d'ajouter certaines restrictions par heure à cet utilisateur. Ainsi, le nombre de requêtes, de mises à jour ou de connexions, sont des limites qu'on peut fixer. De plus, le nombre maximal de connexions simultanées pour un utilisateur est lui aussi configurable.

Restriction à 200 connexions simultanées maximum pour un utilisateur existant :

```
mysql> GRANT USAGE ON *.* TO 'user_pub'@'192.168.200.20'  
      with MAX_USER_CONNECTIONS 200;  
mysql> SHOW GRANTS FOR 'user_pub'@'192.168.200.20';  
Grants for user_pub@192.168.200.20: GRANT SELECT ON *.* TO  
'user_pub'@'192.168.200.20' IDENTIFIED BY PASSWORD  
'*1270C0C06DEE42FD1618BB99005ADCA2EC9D1E19'  
      WITH MAX_USER_CONNECTIONS 200
```

Une augmentation brutale du nombre de requêtes peut provenir d'un problème du cache : soit celui-ci a été brutalement désactivé ou invalidé (query cache) soit votre système basé sur memcached est défectueux.

Temps de résolution : une heure

Estimer que l'on a au moins une heure devant soi signifie probablement que le problème ou ses conséquences est en partie identifié. C'est souvent le cas d'une surcharge soudaine sur le serveur MySQL. Attribuée à un script programmé, il est possible de retrouver une situation normale en coupant le programme concerné. Le but est ici de ne pas couper brutalement le responsable mais plutôt de l'optimiser.

Si nous disposons d'une heure pour résoudre le problème, cela laisse le temps d'optimiser une requête complexe et coûteuse, de diffuser le correctif aux équipes concernées et d'assister peut-être à une mise en production dans l'heure si le problème survient en heures ouvrables.

La chasse aux requêtes lentes

Dans l'hypothèse où notre problème soit issu d'une ou plusieurs requêtes trop coûteuses, le but est tout d'abord de les identifier rapidement puis de les analyser afin de les optimiser. Si votre outil de surveillance préféré (MySQL Administrator, mytop, etc.) est suffisant pour repérer les requêtes les plus lourdes s'exécutant en temps réel sur le serveur, ils ne sont d'aucune utilité pour analyser le passé.

En effet, afin de retrouver les requêtes qui ont pu impacter votre serveur et qui sont désormais peut-être terminées (mais appelées à s'exécuter de nouveau), des logs sont nécessaires. MySQL possède un fichier de logs dédiés aux requêtes lentes : il s'agit du `slow query log`. Celui-ci peut par exemple être exploité par `mysqldumpslow` ou `mysqlsa`.

Ne recherchez pas uniquement les requêtes les plus lentes mais également les plus fréquentes. En effet, votre charge MySQL est susceptible de provenir d'une grande quantité de petites requêtes n'apparaissant pas dans les logs de requêtes lentes selon le critère utilisé le plus souvent : le temps d'exécution. 100 000 requêtes de 0,9 seconde chacune n'apparaissent pas si le `long_query_time` est défini à une seconde ; pourtant, celles-ci sont peut-être responsables d'une surcharge serveur.

À LIRE **Un chapitre consacré à l'étude des journaux**

Au sujet des logs, lire ou relire notre chapitre 5 consacré aux logs MySQL. Il est possible d'agir sur la façon dont le serveur MySQL génère les journaux des erreurs, de requêtes lentes ou généraux. Il est précieux, tout particulièrement en situation d'urgence, de connaître leur fonctionnement.

Réécrire les requêtes trop coûteuses

Une heure, cela laisse le temps d'agir sur un nombre restreint de requêtes un peu trop gourmandes. Cependant le temps passe vite : certaines requêtes nécessitent à elles

seules ce laps de temps pour améliorer les choses. Il est parfois difficile d'analyser le bien-fondé fonctionnel d'une requête complexe multipliant les jointures et les sous-requêtes. Aussi, si cela est possible, proposez au développeur de se joindre à vous pour une première analyse de celles-ci. Les critères de jointures sont-ils toujours pertinents ?

Certains développeurs sont capables de créer du code SQL complexe et valide mais c'est également à l'administrateur de bases de données de leur signaler le fonctionnement interne de MySQL et d'évoquer certaines faiblesses propres à la branche 5.x : sous-requêtes imbriquées et/ou corrélées aux requêtes externes. Désormais au courant de nouvelles bonnes pratiques, le développeur est susceptible de corriger lui-même certaines des requêtes remontées lors de votre recherche de requêtes lentes, vous laissant ainsi du temps disponible pour en corriger d'autres. Une fois la correction du développeur effectuée, comparez les `EXPLAIN` avant et après et classez l'affaire si tout va bien.

Nous attirons votre attention sur le fait que mettre en production une requête sans passer par une phase de test conforme à celle que vous effectuez d'habitude est une source de régression potentielle. Une situation urgente autorise certains raccourcis susceptibles de vous sortir d'affaire, mais il n'est pas pour autant conseillé d'oublier toute prudence ; ayez au moins à l'esprit le risque lié à cette pratique.

Nos conseils pour optimiser vos requêtes se situent dans le chapitre 6, consacré à l'optimisation de la base de données.

ASTUCE Les tables statiques à la rescousse

Si vous estimez qu'il est impossible en une heure de venir à bout de toutes les requêtes à corriger, pensez aux tables statiques ou *summarized tables*. Cela ne fonctionne pas dans tous les cas mais cette technique est très efficace si elle convient. L'idée est de générer une fois pour toutes les résultats d'une requête complète et de les insérer dans une autre table. Plutôt que d'exécuter n fois une requête complexe et lente, elle est jouée une seule fois et ses résultats archivés. L'inconvénient, bien sûr, est que le contenu de la table statique ne sera pas mis à jour. À vous donc de définir éventuellement la périodicité de cette opération (vidage de la table et nouvelles insertions à partir de la requête complexe), un script programmé (`cron`) peut s'en charger. Il est également possible d'utiliser les `event` au sein même de MySQL, vous en trouverez une description sur l'un de nos blogs :

▶ <http://dasini.net/blog/2009/06/16/>

Cette technique a un sens lorsqu'une requête ne peut être mise en cache très longtemps (résultats invalidés par MySQL si par exemple l'une des tables impliquées par la requête est mise à jour). Il est également nécessaire que cette fameuse requête complexe ne soit pas contextuelle à la session d'un internaute. Dans ce dernier cas, les tables temporaires peuvent constituer une bonne solution à condition que leur taille maximale ne soit pas un problème (il s'agit de la plus petite valeur entre ces deux variables système : `max_heap_table_size` et `tmp_table_size`). Dans ce cas la mise en cache dans une table statique n'aurait pas de sens, le contenu doit servir au plus grand nombre. Il peut donc s'agir d'un contenu dynamique dont on peut se permettre une fréquence de rafraîchissement inférieure au temps réel. Un exemple : statistiques et calculs autour des points d'un championnat quelconque. Les résultats ne seront pas obsolètes avant la prochaine épreuve.

Les problèmes de réplication

Le chapitre 8, consacré à la réplication, détaille les mécanismes, la configuration et les commandes nécessaires pour gérer la réplication MySQL. Il est important que son fonctionnement soit bien compris.

En conséquence, nous n'expliquerons pas ici à nouveau les arcanes de la réplication. En revanche, le conseil à retenir dans le cadre d'une situation d'urgence concernant une réplication est de ne pas agir à la légère. En effet, en cas d'erreur de votre part, par exemple lors du redémarrage d'une réplication arrêtée, vous pouvez rendre l'esclave inutilisable avec pour seule issue de recharger complètement les données qu'il contient. L'importance d'un tel incident (esclave hors service) est inversement proportionnelle au nombre d'esclaves possédés. C'est une situation dramatique si vous n'en possédez qu'un seul (faire pointer les applications vers le maître si celui-ci tient la charge...) mais moins grave si vous en possédez un grand nombre. L'incident pourrait passer inaperçu si votre système est correctement dimensionné avec une bonne répartition de charge.

Les messages d'erreurs liés à la réplication sont assez variés. Ne pas comprendre l'origine du problème vous expose à de graves difficultés. Il serait inconsidéré d'appliquer à la légère le fameux `SET GLOBAL sql_slave_skip_counter=1` indiquant à l'esclave de passer outre le problème qui a généré l'erreur et de sauter à l'instruction suivante du `relay-log`. Le risque de désynchronisation entre l'esclave et le maître serait alors important et d'autres erreurs analogues ne manqueraient pas de se produire à plus ou moins courte échéance.

Pire encore, ignorer systématiquement un certain type d'erreurs (`slave-skip-errors`) sans maîtriser réellement les conséquences de ce réglage, est une grosse erreur.

Selon le type de moteur de stockage que vous utilisez (MyISAM ou InnoDB), et le type de réplication (`row based` ou `statement based`) une requête stoppée manuellement sur le maître peut avoir des conséquences différentes sur l'esclave. Avec MyISAM et en mode `statement based`, par exemple, une requête interrompue sur le maître provoque :

```
Query partially completed on the master (error on master: 1053) and was
aborted. There is a chance that your master is inconsistent at this
point. If you are sure that your master is ok, run this query manually
on the slave and then restart the slave with SET GLOBAL
SQL_SLAVE_SKIP_COUNTER=1; START SLAVE;
```

L'esclave détecte que la requête a été interrompue sur le maître (cette information circule dans le log binaire) et décide de stopper lui aussi la réplication. À vous de vérifier que la ou les tables concernées sur le maître et l'esclave sont bien symétriques : il n'y a

pas de `ROLLBACK` sur `MyISAM`. Si c'est le cas, vous pouvez alors utiliser le conseil prodigué par le message d'erreur `SET GLOBAL...` et redémarrer la réplication.

La réplication `MySQL` est asynchrone : une information enregistrée sur le maître n'est pas immédiatement disponible pour l'instruction suivante sur tous les esclaves. En conséquence, certaines applications ou scripts rencontrent des problèmes. C'est le cas typique cité précédemment : une insertion est jouée sur le maître et le script recherche immédiatement son identifiant correspondant sur l'esclave ; s'il ne le trouve pas, son comportement peut s'en trouver perturbé.

Une façon simple de gérer ce souci est d'effectuer les requêtes de ce type (récupération d'une clé liée à l'insertion d'un enregistrement) sur le maître.

Temps de résolution : une journée

À l'échelle de la journée nous sommes pour ainsi dire sortis de l'urgence. La survie de la base n'est plus une question de minutes mais plutôt de jours. Cependant, le scénario que nous suivons (rappelez-vous du découpage arbitraire en trois parties) implique que la situation est préoccupante et demande des restructurations parfois importantes.

Une journée ne représente que quelques heures durant lesquelles il va falloir trouver des solutions. À ce stade le problème est en effet détecté et il s'agit probablement des prémices d'une montée en charge difficile. Peut-être rencontrez-vous de façon récurrente des ralentissements à horaires réguliers, des requêtes de plus en plus lentes, des tables sur lesquelles il est difficile d'effectuer des opérations de maintenance sans bloquer la production.

LIRE Chapitre 8 consacré à la réplication

Les indisponibilités de la base liées aux changements de structures `ALTER`, `DROP`, ou de versions peuvent être résolus en partie grâce à la réplication.

Bref, il est temps de prendre le problème à bras-le-corps et d'y consacrer du temps à l'avenir puisque c'est finalement cela dont il s'agit : anticiper les problèmes de demain.

Face à des problématiques d'une telle envergure, les réponses techniques ne sont pas forcément immédiates, tout du moins pas applicables en pleine journée. Certaines phases sont en effet bloquantes : modification de tables (`ALTER TABLE`), refonte d'une partie de l'application (phase de mise en production), remaniement des crons... tout cela mérite quelques tests. Cependant si une opération coup de poing est décrétée sur un thème précis, comme résoudre les problèmes récurrents de performance du serveur xxx, il s'agit d'agir vite et bien. Toutes les difficultés ne seront probablement pas

réglées le soir même mais les solutions à appliquer et les premiers correctifs sont applicables pour le lendemain (les premiers scripts correctifs sont susceptibles de s'exécuter pendant la nuit si votre activité s'y prête).

PRATIQUE **Un problème peut en cacher un autre**

Sur une telle opération, il est souhaitable d'impliquer les utilisateurs concernés par les modifications de base. Il est fréquent qu'en discutant d'un problème de performance les utilisateurs fassent part au DBA qu'ils ont par ailleurs des retours de blocage intermittents à différents horaires, des pertes de connexion (*MySQL Server has gone away*), etc. Plutôt que d'effectuer plusieurs réunions pour différents problèmes sur un même serveur, autant tous les évoquer au moins une fois et tenter de les résoudre en même temps, si possible. Un passage en InnoDB peut résoudre les problèmes de verrous évoqués par les utilisateurs ; un index contribuera à accélérer certaines requêtes ainsi que le partitionnement d'une table volumineuse. De plus, si la réunion n'est pas purement orientée bases de données, un administrateur système ou réseau sera le bienvenu pour régler certains problèmes de connexion ou de perte de paquets, constatés éventuellement sur certaines machines.

Afin de dessiner une solution face à une montée en charge avérée ou à venir, un passage par le chapitre 4, consacré à la surveillance de votre serveur, est tout indiqué. En fonction des conclusions de cet audit, il sera possible d'émettre un planning et de classer les actions à effectuer par priorité ou faisabilité.

Voici un échéancier tel que l'administrateur de bases de données pourrait l'écrire à l'issue d'une telle réunion :

- lundi :
 - mise à jour matérielle nécessaire (+8 Go RAM et 2 quad-cœurs, attente des composants) ;
 - optimisation des requêtes facturation ;
 - partitionnement de la table « factures » sur l'environnement de développement ;
 - benchmarks de la version partitionnée (mysqslap) ;
- nuit lundi/mardi :
 - ALTER TABLE factures (partitions) si tests concluants ;
 - passer les tables factures_client_xx en InnoDB ;
- mardi :
 - consulter logs des actions ALTER partitions/passage en InnoDB ;
 - surveiller cron 10h45 et comparer situation de la veille.

Conseils généraux face à l'urgence

Tirer profit du passé

Documenter les commandes ou procédures clés à appliquer en cas de problème au travers d'un wiki, d'un fichier texte, etc., permet le jour *J*, quelle que soit l'heure, d'être opérationnel. Pensez à votre état de fraîcheur si l'on vous réveille lors d'une astreinte à 3h du matin... Ne pas perdre de temps à se rappeler comment éteindre ou rallumer certains services importants en plus des procédures liées à l'activité MySQL n'est pas négligeable dans pareilles situations.

Si ce n'est pour vous, faites-le pour les personnes susceptibles de prendre le relais en votre absence. Ce document doit être enrichi régulièrement, au fil des problèmes rencontrés, des changements de serveurs, de DNS, etc.

Anticiper les problèmes

C'est évident mais tellement vrai : mieux vaut prévenir que guérir. Lorsque les problèmes surviennent il est parfois déjà trop tard. Outre les urgences quotidiennes, l'administrateur de bases de données doit veiller à se ménager du temps, chaque jour dans l'idéal, pour améliorer l'existant, effectuer quelques opérations de maintenance (`OPTIMIZE TABLE`, `ANALYZE TABLE`, `CHECK TABLE` ou sa version client bien pratique `mysqlcheck`), repérer les tables volumineuses susceptibles de poser problème (partitionnement nécessaire ?), suivre de près les serveurs du moment, ceux dont l'activité est particulièrement forte ou importante à un instant précis.

ATTENTION Modifications à chaud, en production

Soyez extrêmement vigilant à l'égard des modifications qui sont susceptibles d'être exécutées à chaud sur les machines de production. Certaines variables système sont en effet modifiables sans redémarrage du serveur MySQL. Il est important d'en mesurer tous les impacts possibles en termes de performance et de stabilité du système. Il est possible de mettre un serveur à plat en modifiant une seule ligne dans un fichier de configuration `my.cnf` ; par exemple, passer `sync_binlog` de 0 à 1 peut entraîner une chute dramatique des performances de tout le système disque (au sujet de cette variable, lire également le chapitre 8, consacré à la réplication, et le 2, dédié au matériel). De manière générale, il faut reporter dans le fichier de configuration de MySQL toute modification effectuée (et validée) à chaud. Sans cela, cette nouvelle valeur serait perdue au prochain redémarrage du serveur.

Quelques bonnes pratiques pour terminer :

1. archivez dans un document toute modification effectuée ainsi, afin qu'en cas de problème vous puissiez remonter le fil de vos actions. Notez les horaires ;
2. reportez ces modifications dans le `my.cnf` ;
3. envoyez par e-mail à votre équipe les modifications effectuées. L'équipe d'astreinte ou les administrateurs système pourront ainsi éventuellement mettre en relation un problème et votre modification, accélérant la résolution du cas.

L'entraînement à l'urgence

Il ne suffit pas d'installer un maître et deux esclaves pour se croire à l'abri de tout problème... Que se passe-t-il en cas de perte du master ? Dans un autre registre, les sauvegardes sont-elles fiables ? On lit souvent qu'il faut tester ses sauvegardes, c'est-à-dire les recharger sur un serveur et s'assurer que les données soient bien présentes et correctes ; mais combien d'entre nous l'effectuons réellement ? Si cela devait se produire en production, en combien de temps seriez-vous capable de tout remonter ? Ces types de scénarios catastrophes se produiront un jour ou l'autre et il faut y être préparé. Si les machines vous manquent pour tester tout cela, il est possible d'utiliser des solutions telles que MySQL Sandbox pour se construire une configuration analogue à celle de la production à moindre coût. Autre test intéressant à effectuer sur des serveurs de test : débrancher physiquement le courant et rallumer le tout. Où en est la réplication ? Des tables sont-elles corrompues ? En combien de temps le serveur MySQL redémarre-t-il ? Difficile de le deviner : il faut tester. L'utilisation de machines virtuelles permet également de se constituer un parc de machines avec lesquelles s'entraîner.

Enregistrer les données de l'incident

Quel que soit le degré d'urgence dans lequel vous vous trouvez, à des fins immédiates ou pour plus tard, tentez de sauvegarder des informations concernant l'état du serveur au moment de l'incident. Le chapitre 4, consacré aux solutions de surveillance, est un bon point d'entrée.

En quelques mots, archiver les résultats de commandes systèmes telles que :

```
iostat -dx 5
vmstat 1 10
top -b -n 1
cat /proc/meminfo
```

et du côté de MySQL (si cela est possible) :

```
SHOW FULL PROCESSLIST;
SHOW INNODB STATUS\G
SHOW GLOBAL STATUS;
```

L'état d'esprit à adopter dans l'urgence

Terminons ce chapitre sur quelques règles élémentaires en cas de coup dur.

- Ne paniquez pas, restez concentré.

- Tentez de mesurer rapidement l'impact du problème sur le reste du système/des applications.
- Attention aux conclusions hâtives.
- Soyez méthodique.
- Connaître le fonctionnement des applications est un plus. Sans autres ressources humaines que vous (en pleine nuit par exemple), vous pourrez agir sur certains scripts ou identifier plus rapidement la source du problème.
- Dans l'urgence, allez au plus simple.
- Si possible, effectuez une sauvegarde de ce que vous manipulez, un service dégradé vaut mieux que pas de service du tout.

Trouver de l'aide

Vous êtes dépassé par le problème ? Cela arrive. Dans l'intérêt de tous, ne restez pas seul à tenter de résoudre l'affaire si vous n'y arrivez pas et n'avez aucune piste. Si aucune aide n'est disponible en interne et que vous tournez en rond, il est temps de solliciter une aide extérieure. Lisez le chapitre 8 consacré à ce sujet. Il existe des services de support joignables par Internet ou par téléphone.