Réplication MySQL

Amélioration de l'évolutivité et de la disponibilité avec MySQL 5.5

Livre blanc technique MySQL® par Oracle

Octobre 2010

TABLE DES MATIÈRES

1 Introduction

La réplication MySQL a été largement déployée par certains sites internet leader sur le Web et dans les entreprises pour fournir des niveaux extrêmes d'évolutivité de base de données. Les utilisateurs peuvent créer aisément et rapidement plusieurs répliques de leur base de données afin d'effectuer un déploiement horizontal souple et s'émanciper des contraintes en capacité d'une instance unique. Ils peuvent ainsi répondre rapidement aux charges de travail croissantes de base de données.

La réplication sert également de point de départ pour offrir des services de base de données hautement disponibles, en offrant un moyen de mettre en miroir les données sur plusieurs hôtes pour répondre aux défaillances de systèmes individuels. De plus, de nombreux utilisateurs répliquent des données dans des systèmes dédiés aux fonctions de sauvegarde ou d'analyse de données afin d'utiliser les ressources plus efficacement en déchargeant ces tâches de leurs serveurs de production.

Avec la publication de MySQL 5.5, plusieurs améliorations ont été apportées à la réplication MySQL, qui offrent des niveaux supérieurs d'intégrité des données, de performance et de flexibilité dans les applications. Nous présentons ces améliorations dans le livre blanc (au moment de la rédaction de ces lignes, MySQL 5.5 est disponible uniquement en version Candidate).

Nous examinons également les avantages du déploiement de la réplication MySQL d'un point de vue commercial et technique, nous présentons les principes fondamentaux de la technologie sur laquelle repose la réplication et nous fournissons un simple guide pas-à-pas d'installation et de configuration d'une topologie maître/esclave, et de gestion des événements de reprise après échec.

En conclusion, ce livre souligne les principales essentielles de l'utilisation de la réplication avec la base de données MySQL Cluster.

2 Principes fondamentaux de la réplication

Pour les besoins de ce livre, nous définissons « réplication » comme la duplication des données vers un ou plusieurs emplacements. Dans les sections suivantes, nous présentons ce qui différencie les différents types de réplication les plus populaires.

La réplication permet à une base de données de copier ou de dupliquer des changements d'un emplacement physique ou d'un système vers un autre (généralement depuis un système « maître » vers un système « esclave »). Elle permet généralement d'accroître la disponibilité et l'évolutivité d'une base de données, bien que souvent les utilisateurs exécutent aussi des opérations de sauvegarde ou des requêtes d'analyse sur les systèmes esclaves, déchargeant ainsi ces fonctions des systèmes maîtres.

MySQL prend en charge en natif la réplication en tant que fonctionnalité standard de la base de données. Selon la configuration, vous pouvez répliquer toutes les bases de données, une sélection de bases de données ou même des tables sélectionnées dans une base de données.

La réplication MySQL fonctionne simplement avec un serveur agissant en tant que maître et un ou plusieurs serveurs agissant en tant qu'esclaves. Le serveur maître consigne les changements dans la base de données. Une fois ces modifications consignées, elles sont envoyées vers le ou les esclaves, puis appliquées immédiatement ou après un délai défini.

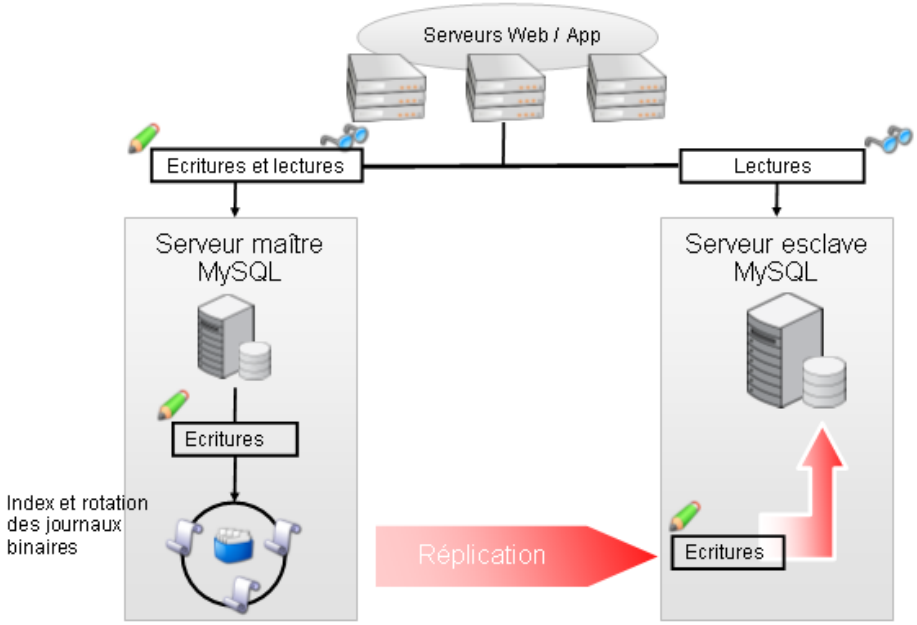

Figure 1 La réplication MySQL prend en charge la haute disponibilité et l'évolutivité en lecture « prêtes à l'emploi »

L'utilisation de la réplication MySQL permet de déployer horizontalement d'importantes fermes de serveurs Web dans lesquelles les lectures (SELECTs) représentent la majeure partie des opérations effectuées dans la base de données. Les esclaves représentent une charge de travail infime sur les serveurs maîtres (généralement 1 % de charge de travail par esclave). Il n'est pas rare d'avoir 30 esclaves déployés par maître dans les sites Web importants[1].

Réplication asynchrone

MySQL prend en charge en natif la réplication à sens unique, asynchrone. La réplication asynchrone signifie que les données sont copiées d'une machine à une autre, entraînant un délai dans la copie des modifications de données. Les données risquent surtout de ne pas être copiées vers / appliquées à l'esclave, lorsque la validation de la transaction est confirmée à l'application. Ce délai dépend souvent de la bande passante réseau, de la disponibilité des ressources et de la charge du système. Cependant, avec les composants et un réglage appropriés, la réplication elle-même peut sembler presque instantanée pour de nombreuses applications.

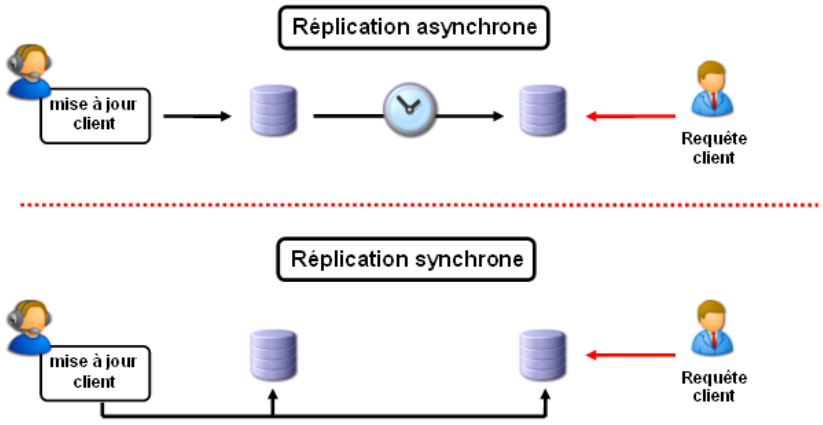

Figure 2 Réplication asynchrone et synchrone

[1] Pour consulter un exemple de topologies de réplication MySQL complexes, reportez-vous à la documentation relative à Ticketmaster dans la section Ressources

Réplication synchrone

La réplication synchrone correspond à la validation simultanément des données sur une ou plusieurs machines, généralement connue sous le nom « validation à deux phases ». Cette réplication permet d'optimiser la cohérence entre plusieurs systèmes, mais l'augmentation des messages pénalise les performances.

Avec les moteurs de stockage InnoDB ou MyISAM, MySQL ne prend pas en charge en natif la réplication synchrone. Il existe toutefois des technologies, telles que Distributed Replicated Block Device ou DRBD, qui fournissent une réplication synchrone du système de fichiers sous-jacent permettant à un deuxième serveur MySQL de prendre le relais (à l'aide de cette deuxième copie) en cas de perte de la copie/du serveur initial. Pour plus d'informations, reportez-vous à : http://www.drbd.org/

Si vous utilisez MySQL Cluster, les données sont répliquées de façon synchrone entre les nœuds de données dans le Cluster (site), puis de façon asynchrone entre des Clusters géographiquement séparés.

Réplication semi-synchrone

La réplication semi-synchrone est une nouvelle fonctionnalité de MySQL 5.5 (au moment de la rédaction de ces lignes, MySQL 5.5 est une version Candidate). Cela signifie que si la réplication semi-synchrone est activée sur le maître et qu'un esclave semi-synchrone au moins est configuré, un thread exécute une validation de transaction sur les blocs maîtres une fois la validation appliquée localement et attend jusqu'à ce qu'un esclave semi-synchrone au moins renvoie au maître la confirmation de réception de tous les événements associés à la transaction, ou jusqu'à une expiration. En cas d'expiration, le maître valide transaction mais rétablit le mode asynchrone.

En utilisant la réplication asynchrone, si le maître tombe en panne, il n'est pas possible de savoir immédiatement si les transactions validées par le maître ont été répliquées sur l'esclave. Par conséquent, une reprise d'un maître vers un esclave peut entraîner une reprise à partir d'un serveur qui ne contient pas toutes les transactions relatives au maître. La réplication semi-synchrone réduit le risque de transactions « orphelines » sur le serveur maître, car toutes les transactions validées ont été reçues par l'esclave. En d'autres termes, pour tout client thread, les modifications effectuées sur le maître lors de sa transaction « en cours » sont perdues (le client doit réessayer), mais toutes les modifications de transactions, dont la validation a été confirmée, sont préservées.

La réplication semi-synchrone a un impact sur les performances car les validations sont plus lentes en raison de la nécessité d'attendre les esclaves. Il s'agit de la contrepartie de l'augmentation de l'intégrité des données. Le ralentissement correspond au moins au délai TCP/IP aller-retour d'envoi de la validation à l'esclave, d'enregistrement par l'esclave dans son journal de relais et d'attente de l'accusé de réception de l'esclave. Par conséquent, la réplication semi-synchrone est plus efficace avec des serveurs physiquement colocalisés qui communiquent via des réseaux rapides.

La réplication semi-synchrone n'est pas disponible actuellement pour les tables utilisant le moteur de stockage MySQL Cluster.

Réplication par instruction

Par défaut, MySQL exploite la réplication par instruction lorsque les instructions SQL (pas les modifications de données) sont répliquées à partir du maître vers le ou les esclaves. La réplication par instruction est incluse dans MySQL server depuis la version 3.23.

L'un des avantages de la réplication par instruction est la réduction, dans certains cas, de la quantité des données écrites dans les fichiers journaux, par exemple lorsque des mises à jour ou

des suppressions affectent de nombreuses lignes. Pour des instructions simples qui affectent quelques lignes seulement, la réplication par ligne peut occuper moins d'espace.

La réplication par instruction présente toutefois quelques inconvénients, notamment l'absence de prise en charge des instructions au comportement non déterministe, par exemple une fonction d'heure actuelle.

Réplication par ligne

Introduite par la version MySQL 5.1, la réplication par ligne consigne les modifications dans des lignes de table individuelles, contrairement aux instructions. Avec la réplication par ligne, le maître écrit des messages, également connus en tant qu'événements dans le journal binaire, qui indiquent comment les lignes de table individuelles ont été modifiées. Elle s'apparente aux formes plus traditionnelles de réplication incluses dans d'autres RDMS. Généralement, la réplication par ligne nécessite moins de verrous sur le maître et l'esclave, ce qui permet d'atteindre un niveau de concurrence plus élevé. L'un des inconvénients de la réplication par ligne est qu'elle génère habituellement un volume de données à consigner plus important. Par exemple, une instruction qui modifie 100 lignes dans une table représente 100 changements à consigner avec la réplication par ligne, alors qu'avec la réplication par instruction seule l'instruction SQL doit être répliquée.

Avant la version MySQL 5.5, les types de colonne devaient être identiques sur le maître et l'esclave lors de l'utilisation de la réplication par ligne. Depuis la version 5.5, vous pouvez autoriser ou non le type promotion et/ou rétrogradation, ou imposer le contrôle de type strict.

Si MySQL Cluster est utilisé, la réplication par ligne doit être utilisée.

Réplication en format mixte

Depuis la version MySQL 5.1.8, le format de journalisation binaire peut être modifié en temps réel, selon l'événement à consigner, en utilisant la journalisation en format mixte. Avec le format mixte activé, la réplication par instruction est utilisée par défaut, mais elle bascule automatiquement vers une réplication par ligne dans certaines conditions, par exemple :

- Une instruction DML met à jour une table MySQL Cluster
- Une instruction contient un objet UUID()
- Deux tables ou plus avec des colonnes AUTO_INCREMENT sont mises à jour
- Un INSERT DELAYED est exécuté, quel qu'il soit
- Lorsque le contenu d'une vue nécessite la réplication par ligne, l'instruction qui crée la vue l'utilise également, par exemple lorsque l'instruction créant une vue utilise la fonction UUID()
- Un appel à une fonction définie par l'utilisateur (UDF) est effectué

Pour consulter la liste complète des conditions, visitez :
http://dev.mysql.com/doc/refman/5,1/en/binary-log-mixed.html

3 Liste des modifications apportées à la réplication dans MySQL 5.5

MySQL 5.5 apporte plusieurs améliorations à la réplication, répertoriées ci-après dans cette section. Certaines améliorations sont développées plus loin dans ce document.

- **Réplication semi-synchrone** : amélioration de la résilience, avec le maître qui attend que l'esclave conserve des événements.
- **Réglage fsync de l'esclave et récupération automatique des journaux de relais** : option permettant de gérer l'écriture des journaux de relais sur le disque, sans dépendre du comportement par défaut du système d'exploitation. Définissez `sync_relay_log=1` pour vous assurer qu'une instruction ou transaction au maximum est manquante dans le journal de relais après un incident. L'esclave peut désormais reprendre à partir de

journaux de relais corrompus en demandant au maître de renvoyer les entrées corrompues. Trois nouvelles options sont introduites (sync-master-info, sync-relay-log et sync-relay-log-info). Leur utilisation est présentée dans le manuel de référence MySQL à l'adresse http://dev.mysql.com/doc/refman/5.5/en/replication-options-slave.html#sysvar_sync_master_info.

- **Heartbeat de réplication** : vérifie automatiquement l'état de la connexion entre le maître et le ou les esclave(s) grâce à un dispositif de détection des défaillances plus précis. Elle peut détecter une perte de connexion en millisecondes (configurable) et évite toute rotation inutile de journal de relais lorsque le maître est inactif.
- **Filtrage de réplication par serveur** : lorsqu'un serveur est supprimé d'un anneau de réplication, un serveur restant peut être sélectionné pour supprimer ses messages de réplication en attente, une fois appliqués par tous les serveurs.
- **Conversions de type esclave précises** : permet d'utiliser différents types sur le master et sur l'esclave, avec un type de promotion et de rétrogradation automatique lors de l'utilisation de la réplication par ligne (déjà possible avec la réplication par instruction).
- **Vidage de journal individuel** : vidage sélectif des journaux de serveurs lors de l'utilisation de 'FLUSH LOGS' pour un contrôle accru.
- **Journalisation sécurisée des transactions mixtes** : réplication des transactions contenant les modifications InnoDB et MyISAM

4 Cas d'utilisation de la réplication

Vous pouvez répliquer vos données d'un serveur MySQL à un autre pour de nombreuses raisons, tant techniques que commerciales. Dans cette section, nous explorons divers cas d'utilisation et scénarios d'application dans lesquels la réplication MySQL peut être utile.

Déploiement horizontal

Il s'agit de la situation la plus fréquente justifiant la mise en œuvre d'une réplication pour les utilisateurs. Dans une topologie de déploiement horizontal, l'objectif principal est de distribuer la charge de travail sur un ou plusieurs serveurs esclaves afin d'améliorer les performances.

Les utilisateurs peuvent créer aisément et rapidement plusieurs répliques de leur base de données afin d'effectuer un déploiement horizontal souple et s'émanciper des contraintes en capacité d'une instance unique. Ils peuvent ainsi répondre rapidement aux charges de travail croissantes de base de données.

Il s'oppose au déploiement vertical, qui consiste à augmenter les ressources (généralement la mémoire vide et le processeur) sur la machine existante. Le déploiement vertical peut être associé à une approche verticale de type « chariot élévateur » visant à répondre à des besoins accrus en capacité.

Dans une architecture de déploiement horizontal, les lectures et les écritures sont partagées entre le serveur maître et le ou les serveurs esclaves.

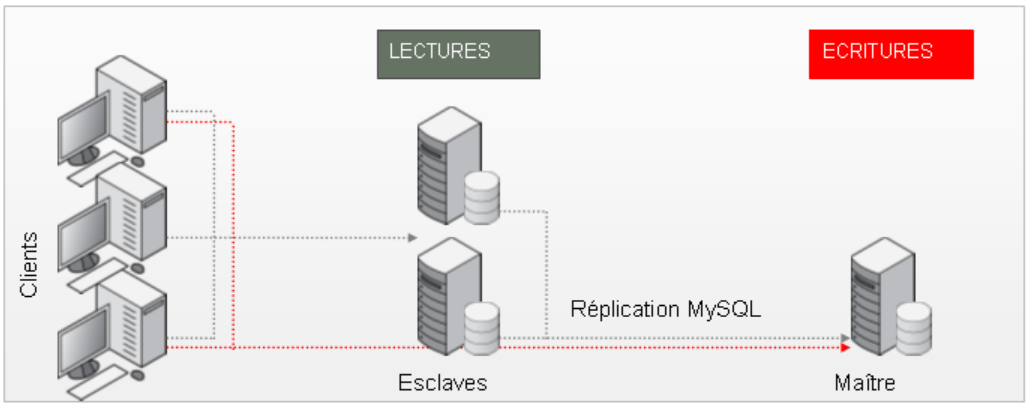

Figure 3 Déploiement horizontal avec la réplication MySQL

Plus précisément, toutes les écritures (UPDATE, INSERT, DELETE) sont envoyées au maître pour exécution, et les lectures (SELECT) sont dirigées vers un ou plusieurs esclaves. La charge de travail de lecture, précédemment exécutée sur le serveur maître, peut ainsi exploiter les ressources disponibles sur le ou les serveurs esclaves. Le déploiement horizontal permet une utilisation plus efficace des ressources en distribuant la charge de travail entre plusieurs serveurs.

La séparation des lectures et des écritures peut être gérée dans différentes couches du système, par exemple au sein de l'application (elle gère les connexions à tous les serveurs et choisit d'utiliser une connexion vers le maître ou vers les esclaves) ou dans le connecteur de base de données.

Par exemple, si vous utilisez un connecteur JDBC de MySQL (Connector/J), lors de la connexion à la base de données, vous pouvez indiquer plusieurs serveurs dans la chaîne de connexion, en commençant par le maître suivi des esclaves. Si nous prenons la configuration illustrée dans la Figure 10 (« noir » est le maître, « bleu » et « vert » sont les esclaves), l'application peut se connecter à la base de données « clusterdb » en utilisant la chaîne de connexion « `jdbc:mysql:replication://black,blue,green/clusterdb` ». Une fois connecté de cette façon, Connector/J achemine les transactions vers le serveur approprié (maître/esclave) en fonction de la valeur de l'attribut « `ReadOnly`» de la connexion (demandé à l'aide de `connection.getReadOnly()` et écrit à l'aide de `connection.setReadOnly(bool)`).

Comme la réplication est asynchrone, si l'application nécessite une opération en lecture seule pour s'assurer d'utiliser les toutes dernières valeurs de la base de données, elle doit l'envoyer au maître plutôt qu'aux esclaves. Dans le cas de Connector/J, cela implique l'exécution de `connection.setReadOnly(false)`.

Haute disponibilité

Dans ce scénario, l'idée est de répliquer les modifications depuis un serveur maître vers un esclave afin de basculer sur le serveur esclave en cas de déconnexion du maître suite à une erreur, un incident ou pour effectuer une maintenance.

Comme pour le déploiement horizontal, la sélection du serveur approprié peut être mise en œuvre de différentes façons. Par exemple, si vous utilisez Connector/J avec la configuration illustrée dans la Figure 9 (« noir » est le maître et « bleu » est l'esclave), l'application peut utiliser « `jdbc:mysql://black,blue/clusterdb` » comme chaîne de connexion. Connector/J envoie ensuite toutes les opérations vers « noir » tant qu'il est disponible, puis bascule vers « bleu » lorsqu'il n'est pas disponible.

Réplication géographique

Avec la réplication géographique, l'objectif est de répliquer des données entre deux emplacements géographiquement distincts, généralement très éloignés. La réplication asynchrone est la solution la plus adaptée dans ce scénario basé sur l'impact potentiel de la

latence du réseau. Par exemple, les données d'une agence centrale à Paris sont répliquées vers une agence régionale à Marseille, autorisant ainsi des requêtes d'exécution à partir d'un magasin de données local. Cette approche est également idéale pour mettre en œuvre la récupération d'urgence en cas de catastrophe sur l'un des sites (par exemple une panne électrique ou une catastrophe naturelle).

Figure 4 Réplication pour redondance géographique

Sauvegarde de base de données

Pour éviter toute détérioration des performances ou verrouillage qu'une sauvegarde sur le maître peut engendrer, vous pouvez exécuter à la place la sauvegarde sur le serveur esclave. Notez que les sauvegardes en ligne peuvent être exécutées sur la base de données maître en utilisant MySQL Cluster.

Analyses

De nombreuses requêtes d'informatique décisionnelle ou d'analyse peuvent être exigeantes en ressources et nécessiter des délais d'exécution importants. Pour répondre à ces requêtes d'analyses, il est possible de créer des esclaves. Dans cette configuration, l'exécution des requêtes n'a aucun impact sur les performances du maître.

Cette méthode peut se révéler très utile avec MySQL Cluster qui est idéal pour les applications utilisant principalement une Clé Primaire basée sur les lectures et écritures, mais dont l'exécution peut être lente avec des requêtes très complexes sur de vastes ensembles de données. Il suffit de répliquer les données MySQL Cluster vers un deuxième moteur de stockage (généralement MyISAM ou InnoDB), puis de générer vos rapports à cet emplacement. Cette action peut être exécutée tout en répliquant des données vers un site MySQL Cluster distant pour une redondance géographique, comme illustrée dans la Figure 5 ci-dessous.

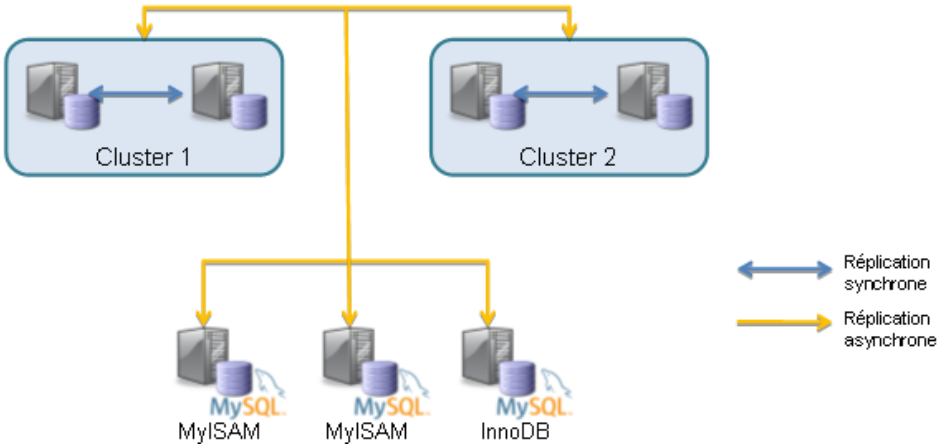

Figure 5 Cas d'utilisation d'association de réplications

5 Topologies de réplication

MySQL prend en charge diverses topologies de réplication. Nous présentons ci-dessous certaines de ces topologies et d'autres prises en charge avec des restrictions.

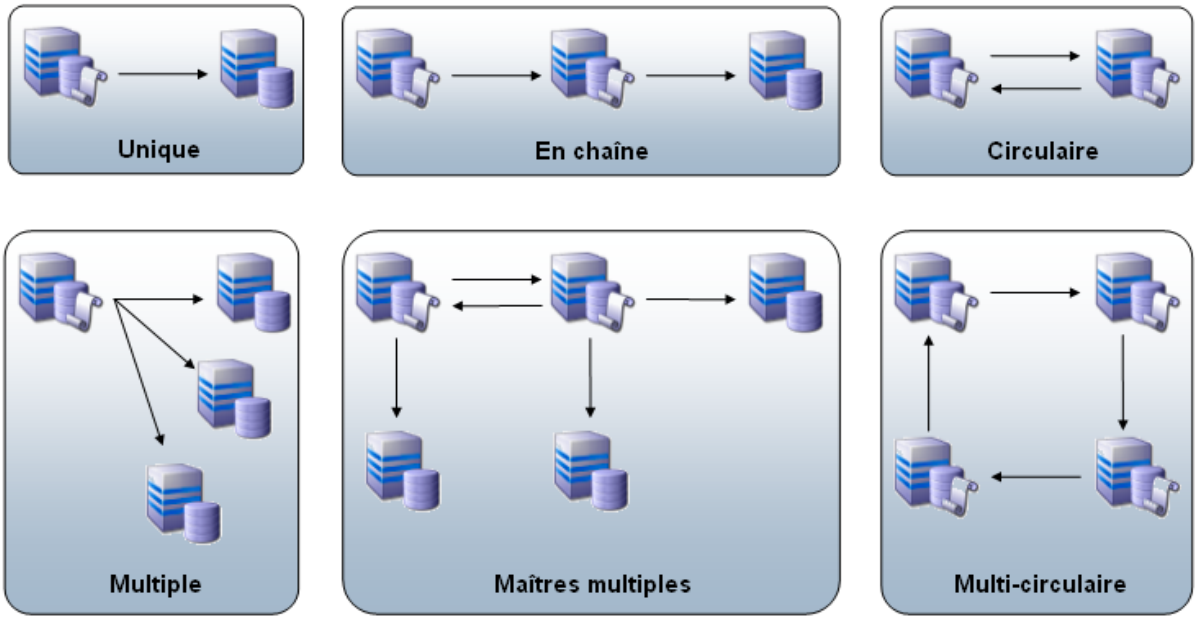

Figure 6 Topologies de réplication MySQL courantes

Maître vers esclave

Cette topologie est la plus populaire et la plus facile en termes de configuration et d'administration. Elle implique deux serveurs, un maître et un esclave. Toutes les écritures sont exécutées sur le maître et les lectures peuvent être partagées entre le maître et l'esclave.

Maître vers plusieurs esclaves

Dans ce scénario, plusieurs esclaves sont associés à un seul maître. Cette topologie offre un degré de déploiement horizontal supérieur contre une administration accrue.

Maître vers esclave(s) vers esclave(s)

Cette configuration est une extension d'une configuration maître/esclave ou maître/esclaves. Dans ce cas, un ou plusieurs esclaves supplémentaires sont associés à l'esclave déjà associé au serveur maître d'origine. En réalité, le ou les esclaves au centre de cette configuration agissent à la fois comme maître et comme esclave. Dans cette configuration, toutes les écritures sont exécutées sur le maître principal.

Maître vers maître (plusieurs maîtres)

Dans une configuration maître/maître, deux serveurs sont associés dans une chaîne et sont réciproquement maîtres et esclaves. Cette configuration permet d'écrire indifféremment sur l'un ou l'autre système, mais comme la modification va être répliquée, le degré de complexité dans l'installation, la configuration et l'administration augmente de façon importante. De plus, sauf si vous utilisez MySQL Cluster, il n'existe pas de détection ni de résolution des conflits. Par conséquent, l'application doit s'assurer de ne pas mettre à jour une ligne sur un serveur alors qu'une modification apportée à la même ligne sur l'autre serveur n'est pas encore répliquée.

Il est également possible d'organiser plusieurs serveurs MySQL en anneau pour augmenter les niveaux d'évolutivité et de performance (à condition que l'application évite d'envoyer des mises à jour conflictuelles vers la même ligne sur différents serveurs). MySQL 5.5 présente une nouvelle fonctionnalité de filtrage qui permet de mieux gérer les échecs sur un serveur alors que la réplication de ses mises à jour est en cours vers les autres serveurs de l'anneau.

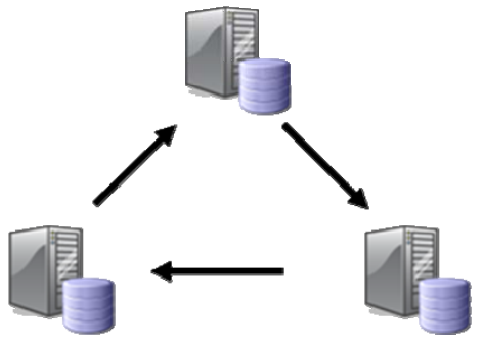

Figure 7 Anneau à maîtres multiples

Lors de l'utilisation de la réplication à maîtres multiples avec des colonnes d'auto-incrémentation, vous devez utiliser les paramètres auto_increment_offset et auto_increment_increment sur chaque serveur pour vous assurer qu'aucune valeur en double n'est affectée. Un exemple pour 2 serveurs (noir et bleu) est illustré dans la Table 1 ci-dessous.

Serveur	auto_increment_increment	auto_increment_offset	Valeurs
noir	2	1	1,3,5…
bleu	2	2	2,4,6…

Table 1 Eviter les valeurs d'auto incrémentation conflictuelles

Maîtres multiples vers esclave (source multiple)

Cette topologie de réplication n'est *pas* prise en charge actuellement par MySQL. Dans une configuration à maîtres multiples, un esclave « sert deux maîtres » principalement, en d'autres termes il réplique les modifications à partir de plusieurs maîtres.

MySQL Cluster permet à plusieurs serveurs MySQL d'écrire sur le même Cluster. Par conséquent, il est possible de configurer cette fonctionnalité (alors que chaque serveur MySQL est l'esclave d'un seul maître) si elle est requise pour une application appropriée.

6 Réplication de flux de travail interne

Avec la réplication MySQL, le maître écrit des mises à jour dans ses fichiers journaux binaires et gère un index de ces fichiers afin de conserver une trace de la rotation des journaux. Les fichiers journaux binaires servent de registre de mises à jour à envoyer aux serveurs esclaves. Lorsqu'un esclave se connecte à son maître, il détermine la dernière position lue dans les journaux en fonction de sa dernière mise à jour réussie. L'esclave reçoit ensuite toutes les mises à jour qui ont eu lieu depuis cette date. L'esclave se bloque par la suite et attend que le maître l'informe de nouvelles mises à jour. Une illustration simple de ces concepts est présentée dans la Figure 8 ci-dessous.

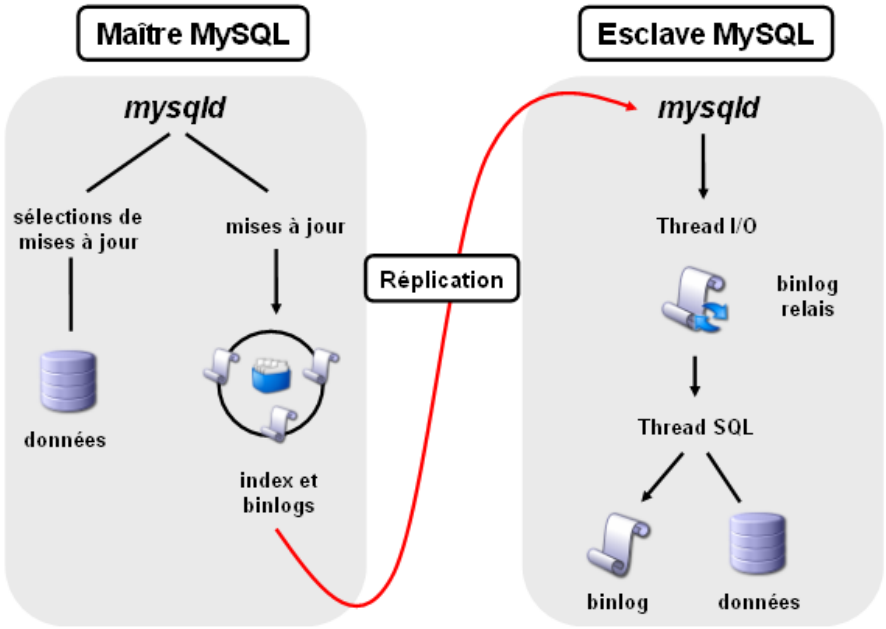

Figure 8 Mise en œuvre de la réplication MySQL

Threads de réplication

Plusieurs threads sont utilisés pour mettre en œuvre la réplication de mises à jour depuis le maître vers le ou les esclaves. Chaque thread est décrit dans cette rubrique. Si vous utilisez MySQL Cluster, un thread supplémentaire est impliqué. Pour plus d'informations, reportez-vous à la section 11.

Thread Binlog Dump

Le maître crée ce thread pour envoyer le contenu du journal binaire à l'esclave. Le thread Binlog Dump acquiert un verrou dans le journal binaire du maître pour lire chaque événement à envoyer à l'esclave. Dès que l'événement est lu, le verrou est annulé, même avant que l'événement ne soit envoyé à l'esclave. Un maître auquel plusieurs esclaves sont « raccordés » crée un thread de vidage binlog pour chaque esclave actuellement connecté, chaque esclave disposant de ses propres threads I/O et SQL.

Thread I/O esclave

Lorsqu'une instruction START SLAVE est émise sur l'esclave, elle crée un thread I/O qui se connecte au maître et lui demande d'envoyer les mises à jour enregistrées dans ses journaux binaires. Le thread I/O esclave lit ensuite les mises à jour que le thread de vidage binlog du maître envoie, puis les copie localement sur l'esclave en tant que journaux de relais dans le répertoire de données de l'esclave.

Thread SQL esclave

L'esclave crée ce thread pour lire les journaux de relais écrits par le thread I/O esclave, et exécute les mises à jour contenues dans les journaux de relais.

Réplication de fichiers journaux

Durant la réplication, le serveur MySQL crée plusieurs fichiers qui sont utilisés pour conserver le journal binaire relayé depuis le maître, et enregistre les informations relatives à l'état et l'emplacement actuel dans le journal relayé.

Trois types de fichiers sont utilisés dans le processus par l'esclave :

le journal de relais

Le journal de relais sur l'esclave contient des événements qui ont été lus sur le journal binaire du maître. Les événements dans le journal binaire sont finalement exécutés sur l'esclave par son thread SQL.

master.info

Les informations relatives à l'état et à la configuration actuelle de l'esclave se situent dans le fichier master.info. Ce fichier contient les informations de connectivité de réplication de l'esclave, y compris le nom d'hôte du maître, les identifiants de connexion utilisés et la position actuelle de l'esclave dans le journal binaire du maître.

relay-log.info

Les informations d'état relatives au point d'exécution dans le journal de relais de l'esclave se situent dans le fichier relay-log.info.

Le journal binaire et le fichier d'index associé, pour le suivi de toutes les mises à jour à répliquer, se situent sur le maître.

7 Configuration de la réplication MySQL

Dans cette section, nous présentons une méthode de configuration de la réplication MySQL. Cette procédure est écrite pour la configuration d'un esclave unique, mais peut être répétée pour configurer plusieurs esclaves. Pour les besoins de ce guide, nous supposons que vous avez téléchargé et installé avec succès au moins deux serveurs MySQL.

Remarque au sujet de l'exemple

Pour les besoins de ce document, nous avons configuré deux serveurs MySQL 5.5 qui jouent les rôles suivants :

Maître
Nom d'hôte : noir
IP : 192.168.0.31

Esclave
Nom d'hôte : bleu
IP : 192.168.0.34

Figure 9 Configuration utilisée pour l'exemple de procédure

Nous supposons que vous utilisez le moteur de stockage InnoDB. Pour comprendre les différences avec le moteur de stockage MySQL Cluster (NDB), doté de fonctions supplémentaires, reportez-vous au chapitre 11.

Pour présenter les cas les plus complexes dans cet exemple, nous supposons que le serveur MySQL à utiliser en tant que maître est déjà exécuté et qu'il contient des données à répliquer vers le nouvel esclave. Si vous commencez avec des données vides, vous pouvez ignorer les étapes 3 et 4.

Compatibilité

Si vous tentez de configurer une réplication entre deux serveurs MySQL déjà installés, assurez-vous que les versions MySQL installées sur le maître et l'esclave sont compatibles. Pour consulter la liste de versions compatibles actuelle, visitez :
http://dev.mysql.com/doc/refman/5.5/en/replication-compatibility.html

Étape 1: Configurer les fichiers cnf du maître et de l'esclave

La première étape de configuration de la réplication permet de modifier le fichier « my.cnf » sur les serveurs utilisés en tant que maître et esclave. Une valeur par défaut est fournie avec l'installation MySQL. Nous fournissons toutefois des fichiers locaux de configuration « master.cnf » et « slave.cnf », à utiliser au démarrage des serveurs MySQL si une base de données MySQL de production est déjà exécutée sur ces serveurs.

Au minimum, nous souhaitons ajouter deux options à la section [mysqld] du fichier master.cnf :

- log-bin : dans cet exemple, nous choisissons black-bin.log
- server-id : dans cet exemple, nous choisissons 1. Le serveur ne peut pas agir en tant que maître si la journalisation binaire n'est pas activée. La variable server_id doit être un nombre entier positif comprise entre 1 to 2^{32}

master.cnf :

```
[mysqld]
server-id=1
log-bin=black-bin.log
datadir=/home/billy/mysql/master/data
innodb_flush_log_at_trx_commit=1
sync_binlog=1
```

Remarque : Pour optimiser la durabilité et la cohérence dans la configuration d'une réplication utilisant InnoDB avec les transactions, vous devez également spécifier les options innodb_flush_log_at_trx_commit=1, sync_binlog=1.

Vous devez ensuite ajouter l'option server-id à la section [mysqld] du fichier slave.cnf de l'esclave. La valeur server-id, comme la valeur master_id, doit être un nombre entier positif compris entre 1 et 2^{32} - 1. Il est également nécessaire que l'ID de l'esclave soit différent de celui du maître. Si vous

configurez plusieurs esclaves, chacun doit avoir une valeur server-id unique, distincte de celle du maître et des autres esclaves. Les valeurs server-id peuvent être comparées à des adresses IP : ces ID identifient de manière unique chaque instance de serveur dans la communauté des serveurs de réplication.

Vous pouvez également définir les noms de fichiers à utiliser par l'esclave en définissant relay-log-index et relay-log.

slave.cnf :

```
[mysqld]
server-id=2
relay-log-index=slave-relay-bin.index
relay-log=slave-relay-bin
datadir=/home/billy/mysql/slave/data
```

A présent, redémarrez les serveurs MySQL à l'aide du gestionnaire de services ou directement à partir de la ligne de commande s'ils ne sont pas exécutés en tant que services :

```
[billy@black ~]$ mysqladmin -u root shutdown # only needed if MySQL already running
[billy@black ~]$ mysqld --defaults-file=/home/billy/mysql/master/master.cnf &

[billy@blue ~]$ mysqladmin -u root shutdown # only needed if MySQL already running
[billy@blue ~]$ mysqld --defaults-file=/home/billy/mysql/slave/slave.cnf&
```

Remarque : Si l'esclave a déjà effectué une réplication précédemment, démarrez le serveur esclave avec l'option i--skip-slave-start pour qu'il ne tente pas de se connecter immédiatement au maître. Vous pouvez également démarrer le serveur esclave avec l'option --log-warnings pour recevoir dans le journal d'erreurs davantage de messages concernant les problèmes (par exemple, des problèmes réseau ou de connexion). Cette option est activée par défaut, mais les connexions abandonnées sont consignées dans le journal d'erreurs uniquement si la valeur de l'option est supérieure à 1.

Étape 2: Créer un utilisateur de réplication

L'étape suivante de la configuration de la réplication permet de créer un compte sur le maître utilisé exclusivement pour la réplication. Pour plus de sécurité, il est vivement recommandé de créer un utilisateur de réplication dédié et d'éviter ainsi d'accorder des privilèges supplémentaires, en plus des permissions de réplication. Créez un compte sur le serveur maître que le serveur esclave peut utiliser pour se connecter. Comme indiqué, le privilège REPLICATION SLAVE doit être accordé à ce compte. Vous pouvez exécuter GRANT dans le client MySQL ou dans votre outil d'administration MySQL favori :

```
[billy@black ~]$ mysql -u root --prompt='master> '
master> CREATE USER repl_user@192.168.0.34;
master> GRANT REPLICATION SLAVE ON *.* TO repl_user@192.168.0.34 IDENTIFIED BY 'billy';
```

Étape 3: Verrouiller le maître, noter la position du Binlog et sauvegarder la base de données maître

Sur le maître, videz toutes les tables et instructions d'écriture de blocs en exécutant une instruction FLUSH TABLES WITH READ LOCK.

```
master> FLUSH TABLES WITH READ LOCK;
```

Lorsque le verrou de lecture placé par FLUSH TABLES WITH READ LOCK est actif, lisez la valeur du nom de fichier du journal binaire actuel, puis décalez sur le maître avec la commande suivante :

```
master> SHOW MASTER STATUS;
+------------------+----------+--------------+------------------+
```

```
| File             | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+------------------+----------+--------------+------------------+
| black-bin.000001 |     1790 |              |                  |
+------------------+----------+--------------+------------------+
```

La colonne « File » affiche le nom du journal et la colonne « Position » affiche le décalage dans le fichier. Dans cet exemple, le fichier journal binaire est « black-bin.000001 » et la position est « 1790 ». Notez ces valeurs qui seront utiles plus loin lors de la configuration de l'esclave. Elle représentent les coordonnées de réplication auxquelles l'esclave doit démarrer le traitement des nouvelles mises à jour depuis le maître.

Remarque : Si le maître a été exécuté précédemment sans journalisation binaire activée, le nom du journal et les valeurs de position affichés par SHOW MASTER STATUS sont vides. Dans ce cas, les valeurs requises ultérieurement pour spécifier le fichier journal et la position de l'esclave correspondent à la chaîne vide ('') et à 4.

Ensuite, en laissant ouverte la fenêtre du client MySQL utilisée pour exécuter le FLUSH TABLES WITH READ LOCK, vous devez vider le contenu des bases de données sur le maître que vous souhaitez répliquer sur l'esclave. Dans notre exemple, nous allons vider le contenu de la base de données « clusterdb ».

Remarque : Vous pouvez ne pas répliquer la base de données système « mysql » si le serveur esclave possède un ensemble de comptes d'utilisateur différent de ceux existant sur le maître. Dans ce cas, vous devez l'exclure du processus de vidage.

Vous pouvez exécuter mysqldump dans le client MySQL ou de façon graphique avec MySQL Workbench.

```
[billy@black ~]$ mysqldump -u root clusterdb > /home/billy/mysql/master/clusterdb.sql
```

Vous pouvez réactiver l'activité d'écriture sur le maître avec la commande suivante :

```
master> UNLOCK TABLES;
```

Étape 4: Charger le fichier de vidage sur l'esclave

Ensuite, vous devez charger le fichier mysqldump depuis le maître vers l'esclave (bien entendu, cette action nécessite de copier le fichier clusterdb.sql du serveur « noir » vers le serveur « bleu »). Cette action peut être exécutée à l'aide de la ligne de commande ou de façon graphique avec MySQL Workbench. Étant donné que la base de données « clusterdb » n'existe pas sur l'esclave, elle doit être créée avant le chargement dans les tables et les données :

```
[billy@blue ~]$ mysql -u root –e ‘create database clusterdb;’
[billy@blue ~]$ mysql -u root clusterdb < /home/billy/mysql/slave/clusterdb.sql
```

Étape 5: Initialiser la réplication

Nous sommes désormais prêt à initialiser la réplication sur l'esclave. Exécutez la commande suivante sur l'esclave :

```
slave> SLAVE STOP;
```

Vous devez ensuite émettre une commande CHANGE MASTER :

```
slave> CHANGE MASTER TO MASTER_HOST='192.168.0.31',
    -> MASTER_USER='repl_user',
    -> MASTER_PASSWORD='billy',
    -> MASTER_LOG_FILE='black-bin.000001',
    -> MASTER_LOG_POS=1790;
```

Où :

- **MASTER_HOST** : est l'IP ou le nom d'hôte du serveur maître, dans cet exemple noir ou 192.168.0.31
- **MASTER_USER :** est l'utilisateur auquel nous avons octroyé le privilège REPLICATION SLAVE à l'étape 2 de cet exemple « repl_user »
- **MASTER_PASSWORD :** est le mot de passe que nous avons affecté à « repl_user » à l'étape 2
- **MASTER_LOG_FILE :** est le nom de fichier que nous avons déterminé à l'étape 3
- **MASTER_LOG_POS :** est la position que nous avons déterminée à l'étape 3

Pour terminer, démarrez la réplication sur l'esclave :

```
slave> start slave;
```

Étape 6: Vérifications de base

Nous sommes désormais prêts à exécuter une vérification de base pour garantir le fonctionnement de la réplication. Dans cet exemple, nous insérons une ligne de données dans la table « simples » sur le serveur maître, puis nous vérifions que ces nouvelles lignes se matérialisent sur le serveur esclave :

```
master> insert into clusterdb.simples values (999);

slave> select * from clusterdb.simples;
+-----+
| id  |
+-----+
|   1 |
|   2 |
|   3 |
| 999 |
+-----+
```

8 Migration vers une réplication semi-synchrone

Cette section suppose que la réplication asynchrone est installée mais que vous souhaitez migrer vers la réplication semi-synchrone, en d'autres termes, cet exemple fait suite à la section 7. Vous pouvez bien entendu commencer à utiliser immédiatement la réplication semi-synchrone si vous n'utilisez pas encore la réplication asynchrone.

Remarque : La réplication semi-synchrone est disponible uniquement dans MySQL 5.5 et ultérieur.

Étape 1: Installer les plug-ins sur le maître et l'esclave

Installez les plug-ins appropriés pour chaque serveur MySQL :

```
master> INSTALL PLUGIN rpl_semi_sync_master SONAME 'semisync_master.so';

slave> INSTALL PLUGIN rpl_semi_sync_slave SONAME 'semisync_slave.so';
```

Étape 2: Activer la réplication semi-synchrone

Pour fonctionner, la réplication semi-synchrone doit être activée sur le maître et sur l'un des esclaves au moins (dans notre cas, un seul esclave existe). Pour cela, vous pouvez définir rpl_semi_sync_master_enabled sur « ON » dans master.cnf et rpl_semi_sync_slave_enabled sur « ON » dans slave.cnf, ou à l'aide de la ligne de commande MySQL :

```
master> SET GLOBAL rpl_semi_sync_master_enabled = on;

slave> SET GLOBAL rpl_semi_sync_slave_enabled = on;
slave> STOP SLAVE IO_THREAD; START SLAVE IO_THREAD;
```

Si la réplication n'est pas en cours d'exécution, il n'est pas nécessaire d'arrêter puis de redémarrer le thread I/O esclave. Vous pouvez simplement émettre START SLAVE.

Vérifiez que la réplication semi-synchrone est active sur le maître et qu'un esclave au moins est connecté en mode semi-synchrone :

```
master> SHOW STATUS LIKE 'Rpl_semi_sync_master_status';
+------------------------------+-------+
| Variable_name                | Value |
+------------------------------+-------+
| Rpl_semi_sync_master_status | ON    |
+------------------------------+-------+
master> SHOW STATUS LIKE 'Rpl_semi_sync_master_clients';
+------------------------------+-------+
| Variable_name                | Value |
+------------------------------+-------+
| Rpl_semi_sync_master_clients | 1     |
+------------------------------+-------+
```

a. Confirmer que la réplication est exécutée en mode semi-synchrone

Nous ajoutons une nouvelle ligne sur le maître, puis nous vérifions les variables d'état pour nous assurer qu'il a été répliqué de façon semi-synchrone (en d'autres termes, l'esclave a accusé réception de la modification avant le délai d'expiration du maître et avant qu'il confirme l'insertion du client de façon asynchrone).

```
master> insert into clusterdb.simples values (100);
master> SHOW STATUS LIKE 'Rpl_semi_sync_master_yes_tx';
+------------------------------+-------+
| Variable_name                | Value |
+------------------------------+-------+
| Rpl_semi_sync_master_yes_tx | 1     |
+------------------------------+-------+

slave> select * from clusterdb.simples;
+-----+
| id  |
+-----+
|   1 |
|   2 |
|   3 |
| 100 |
| 999 |
+-----+
```

9 Administration et dépannage de la réplication

Dans cette section, nous présentons l'exécution de quelques tâches administratives et de dépannage de base sur la réplication MySQL.

Vérification de l'état de la réplication

Les tâches les plus courantes dans la gestion d'un processus de réplication consistent à garantir que la réplication est exécutée et qu'aucune erreur ne se produit entre l'esclave et le maître.

La commande principale à utiliser pour cette tâche est SHOW SLAVE STATUS qui doit être exécutée sur l'esclave. Par exemple :

```
slave> SHOW SLAVE STATUS\G
*************************** 1. row ***************************
               Slave_IO_State:
                  Master_Host: 192.168.0.31
                  Master_User: repl_user
                  Master_Port: 3306
                Connect_Retry: 60
              Master_Log_File: black-bin.000003
          Read_Master_Log_Pos: 790
```

```
                Relay_Log_File: slave-relay-bin.000013
                 Relay_Log_Pos: 936
         Relay_Master_Log_File: black-bin.000003
              Slave_IO_Running: No
             Slave_SQL_Running: No
               Replicate_Do_DB:
           Replicate_Ignore_DB:
            Replicate_Do_Table:
        Replicate_Ignore_Table:
       Replicate_Wild_Do_Table:
   Replicate_Wild_Ignore_Table:
                    Last_Errno: 0
                    Last_Error:
                  Skip_Counter: 0
           Exec_Master_Log_Pos: 790
               Relay_Log_Space: 1238
               Until_Condition: None
                Until_Log_File:
                 Until_Log_Pos: 0
            Master_SSL_Allowed: No
            Master_SSL_CA_File:
            Master_SSL_CA_Path:
               Master_SSL_Cert:
             Master_SSL_Cipher:
                Master_SSL_Key:
         Seconds_Behind_Master: NULL
Master_SSL_Verify_Server_Cert: No
                Last_IO_Errno: 0
                Last_IO_Error:
               Last_SQL_Errno: 0
               Last_SQL_Error:
  Replicate_Ignore_Server_Ids:
             Master_Server_Id: 1
```

Les conseils suivants peuvent vous être utiles pour interpréter les résultats du document généré :

- **Slave_IO_State -** indique l'état actuel de l'esclave
- **Slave_IO_Running -** indique si le thread I/O pour la lecture du journal binaire du maître est en cours d'exécution
- **Slave_SQL_Running -** indique si le thread SQL pour l'exécution d'événements dans le journal de relais est en cours d'exécution
- **Last_Error -** indique la dernière erreur enregistrée lors du traitement du journal de relais. L'idéal est qu'il soit vide et n'indique aucune erreur.
- **Seconds_Behind_Master -** indique le nombre de secondes de retard du thread SQL esclave sur le traitement du journal binaire du maître. Une valeur élevée (ou croissante) peut indiquer que l'esclave ne parvient pas à gérer une grande quantité d'instructions provenant du maître. Une valeur 0 pour Seconds_Behind_Master signifie *généralement* que l'esclave a rattrapé le maître, mais ce n'est pas toujours exact. Par exemple, la connexion réseau entre maître et esclave peut être interrompue sans que le thread I/O esclave ne l'ait encore détecté, c.-à-d. que le délai slave_net_timeout n'est pas encore écoulé.

Sur le maître, vous pouvez vérifier l'état des esclaves en examinant la liste des processus en cours d'exécution sur le serveur.

```
master> SHOW PROCESSLIST \G
```

L'esclave étant au cœur du processus de réplication, la quantité d'informations disponibles dans ce rapport est faible.

Suspension de la réplication

Vous pouvez arrêter et démarrer la réplication des instructions sur l'esclave à l'aide des instructions STOP SLAVE et START SLAVE.

Pour arrêter l'exécution du journal binaire à partir de l'esclave, utilisez STOP SLAVE :

```
slave> STOP SLAVE;
```

Lorsque l'exécution est interrompue, l'esclave ne lit pas le journal binaire à partir du maître via IO_THREAD et arrête le traitement des événements du journal de relais qui n'ont pas encore été exécutés via SQL_THREAD. Vous pouvez interrompre les threads IO ou SQL individuellement en spécifiant le type de thread. Par exemple :

```
slave> STOP SLAVE IO_THREAD;
```

L'arrêt du thread SQL peut être utile si vous souhaitez exécuter une sauvegarde ou une autre tâche sur un esclave qui traite uniquement des événements à partir du maître. Le thread continue d'être lu à partir du maître mais les modifications ne s'appliquent pas encore, ce qui permet à l'esclave de rattraper son retard lorsque vous redémarrez les opérations sur l'esclave.

L'arrêt du thread IO permet aux instructions du journal de relais d'être exécutées jusqu'à ce que le journal de relais cesse de recevoir de nouveaux événements. Cette option peut s'avérer utile pour permettre à l'esclave de rattraper son retard sur les événements du maître, pour exécuter des tâches administratives sur l'esclave, mais également pour vous assurer de disposer des dernières mises à jour jusqu'à un point spécifique. Cette méthode permet également d'interrompre l'exécution sur l'esclave pendant que vous effectuez des tâches administratives sur le maître, tout en garantissant l'absence d'une quantité importante d'événements en attente d'exécution lors du redémarrage de la réplication.

Pour redémarrer l'exécution, utilisez l'instruction START SLAVE :

```
slave> START SLAVE;
```

Si nécessaire, vous pouvez démarrer les threads IO_THREAD ou SQL_THREAD individuellement.

Affichage des journaux binaires

Comme indiqué plus haut, le journal binaire du serveur inclut des fichiers contenant des « événements » qui décrivent les modifications apportées au contenu de la base de données. Le serveur écrit ces fichiers sous un format binaire. Pour afficher leur contenu sous un format texte, utilisez l'utilitaire `mysqlbinlog`. Vous pouvez également utiliser `mysqlbinlog` pour afficher le contenu des fichiers journaux de relais écrits par un serveur esclave dans une configuration de réplication, car les journaux de relais utilisent le même format que les journaux binaires.

Pour plus d'informations sur l'utilitaire `mysqlbinlog`, visitez :
http://dev.mysql.com/doc/refman/5.1/en/mysqlbinlog.html

10 Reprise après échec et récupération

Le maître peut tomber en panne ou être arrêté pour une opération de maintenance. Cette section présente les étapes requises.

Étape 1: Conditions préalables

Dans les sections 7 et 8, nous avons configuré une réplication pour deux serveurs MySQL dans une configuration maître-esclave. Dans cette section, nous allons la développer avec les deux méthodes suivantes :

1. Autoriser le changement dans les relations : un serveur agissant en tant qu'esclave peut devenir maître (par exemple, pour arrêter le maître d'origine à des fins de maintenance) et le maître d'origine devenir esclave
2. Démarrer une configuration plus complexe, comme illustré dans la Figure 10 : un maître unique (noir) et 2 esclaves (bleu et vert)

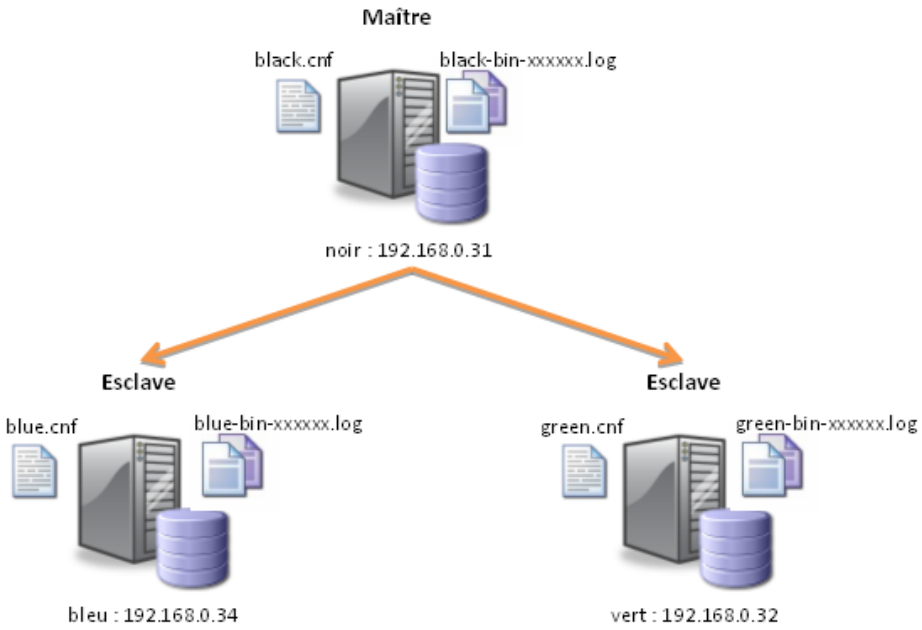

Figure 10 Un maître et deux esclaves

Etant donné que tous ces serveurs peuvent devenir maître, les serveurs esclaves doivent aussi enregistrer les modifications dans un journal binaire. De même, le serveur noir peut devenir esclave et doit par conséquent disposer des données de configuration appropriées. Les fichiers de configuration peuvent se présenter comme suit :

black.cnf :

```
[mysqld]
datadir=/home/billy/mysql/master/data
server-id=1

# Replication Master
log-bin=black-bin.log
innodb_flush_log_at_trx_commit=1
sync_binlog=1

# Replication Slave
relay-log-index=slave-relay-bin.index
relay-log=slave-relay-bin
```

blue.cnf

```
[mysqld]
datadir=/home/billy/mysql/slave/data
server-id=2

# Replication Master
log-bin=blue-bin.log
innodb_flush_log_at_trx_commit=1
sync_binlog=1

# Replication Slave
relay-log-index=slave-relay-bin.index
relay-log=slave-relay-bin
```

green.cnf

```
[mysqld]
datadir=/home/billy/mysql/slave/data
server-id=3

# Replication Master
log-bin=green-bin.log
```

```
innodb_flush_log_at_trx_commit=1
sync_binlog=1

# Replication Slave
relay-log-index=slave-relay-bin.index
relay-log=slave-relay-bin
```

Puisque chaque serveur peut agir en tant que maître et les deux autres en tant qu'esclaves, l'utilisateur de réplication doit être créé sur chaque serveur avec des permissions de connexion depuis l'un des deux autres serveurs :

```
black> CREATE USER repl_user@192.168.0.34;
black> GRANT REPLICATION SLAVE ON *.* TO repl_user@192.168.0.34 IDENTIFIED BY 'billy';
black> GRANT REPLICATION SLAVE ON *.* TO repl_user@192.168.0.32 IDENTIFIED BY 'billy';

blue> CREATE USER repl_user@192.168.0.31;
blue> GRANT REPLICATION SLAVE ON *.* TO repl_user@192.168.0.31 IDENTIFIED BY 'billy';
blue> GRANT REPLICATION SLAVE ON *.* TO repl_user@192.168.0.32 IDENTIFIED BY 'billy';

green> CREATE USER repl_user@192.168.0.34;
green> GRANT REPLICATION SLAVE ON *.* TO repl_user@192.168.0.31 IDENTIFIED BY 'billy';
green> GRANT REPLICATION SLAVE ON *.* TO repl_user@192.168.0.34 IDENTIFIED BY 'billy';
```

Pour terminer, si la réplication semi-synchrone est utilisée, le maître et les plug-ins esclaves doivent être installés sur les trois serveurs, mais la fonctionnalité maître/esclave doit être activée uniquement sur les serveurs appropriés :

```
black> INSTALL PLUGIN rpl_semi_sync_master SONAME 'semisync_master.so';
black> INSTALL PLUGIN rpl_semi_sync_slave SONAME 'semisync_slave.so';
black> SET GLOBAL rpl_semi_sync_master_enabled = on;

blue> INSTALL PLUGIN rpl_semi_sync_master SONAME 'semisync_master.so';
blue> INSTALL PLUGIN rpl_semi_sync_slave SONAME 'semisync_slave.so';
blue> SET GLOBAL rpl_semi_sync_slave_enabled = on;
blue> STOP SLAVE IO_THREAD; START SLAVE;

green> INSTALL PLUGIN rpl_semi_sync_master SONAME 'semisync_master.so';
green> INSTALL PLUGIN rpl_semi_sync_slave SONAME 'semisync_slave.so';
green> SET GLOBAL rpl_semi_sync_slave_enabled = on;
green> STOP SLAVE IO_THREAD; START SLAVE;
```

Étape 2: Détecter si le maître a échoué

Il est fréquent de placer le maître hors ligne intentionnellement, par exemple pour effectuer une maintenance matérielle, auquel cas cette étape ne s'applique pas. Dans les autres cas, la défaillance est évidente, par exemple lors d'une panne matérielle. Il existe toutefois de nombreux types de « défaillance », plus subtils, que vous devez détecter et utiliser pour déclencher une reprise après échec.

Une approche simple consiste à surveiller la valeur « Seconds_Behind_Master » obtenue à partir d'un ou de plusieurs esclaves :

```
blue> show slave status \G
*************************** 1. row ***************************
               Slave_IO_State: Waiting for master to send event
                  Master_Host: 192.168.0.31
                  Master_User: repl_user
                  Master_Port: 3306
                Connect_Retry: 60
              Master_Log_File: black-bin.000003
          Read_Master_Log_Pos: 599
               Relay_Log_File: slave-relay-bin.000013
                Relay_Log_Pos: 745
        Relay_Master_Log_File: black-bin.000003
             Slave_IO_Running: Yes
            Slave_SQL_Running: Yes
              Replicate_Do_DB:
          Replicate_Ignore_DB:
```

```
           Replicate_Do_Table:
       Replicate_Ignore_Table:
   Replicate_Wild_Do_Table:
Replicate_Wild_Ignore_Table:
                 Last_Errno: 0
                 Last_Error:
               Skip_Counter: 0
        Exec_Master_Log_Pos: 599
            Relay_Log_Space: 1047
            Until_Condition: None
             Until_Log_File:
              Until_Log_Pos: 0
         Master_SSL_Allowed: No
         Master_SSL_CA_File:
         Master_SSL_CA_Path:
            Master_SSL_Cert:
          Master_SSL_Cipher:
             Master_SSL_Key:
      Seconds_Behind_Master: 0
Master_SSL_Verify_Server_Cert: No
             Last_IO_Errno: 0
             Last_IO_Error:
            Last_SQL_Errno: 0
            Last_SQL_Error:
Replicate_Ignore_Server_Ids:
           Master_Server_Id: 1
```

Si la valeur Seconds_Behind_Master augmente et que le niveau de mises à jour envoyé au maître n'a pas changé de façon importante, cela peut indiquer que le maître rencontre des problèmes (notamment si le même comportement est observé sur plusieurs esclaves).

Comme illustré dans la Figure 15, MySQL Enterprise Monitor permet notamment de surveiller l'état de la réplication et de signaler la nécessité d'une reprise après échec.

a. Suspendre les écritures vers le maître

Si le processus MySQL Server maître a expiré ou que le matériel sous-jacent a échoué, cette étape peut être obsolète. Cependant, pour des maintenances programmées du maître ou des défaillances plus subtiles, vous pouvez empêcher les applications clientes d'effectuer des modifications pendant la transition.

Il existe diverses méthodes pour interrompre les mises à jour : via l'application, un équilibrage de charge ou le connecteur. Une autre approche consiste à réclamer un verrou sur toutes les tables :

```
black> FLUSH TABLES WITH READ LOCK;
```

b. Promouvoir l'esclave en maître

Notre exemple comporte deux esclaves et nous devons sélectionner celui destiné à devenir maître. La sélection doit prendre en compte l'état de mise à jour, en d'autres termes, retenir le serveur le plus avancé dans le traitement des mises à jour répliquées depuis le maître d'origine. Pour le déterminer, exécutez « `show slave status \G` » sur chaque client.

Une fois sélectionné, le thread I/O (reportez-vous à la Figure 8) doit être interrompu sur le nouveau maître sélectionné (dans notre exemple « bleu » est sélectionné) :

```
blue> STOP SLAVE IO_THREAD;
```

Une fois cette étape effectuée, l'exécution du thread SQL des esclaves se poursuit et applique les mises à jours restantes à partir du journal de relais. Lorsque le délai nécessaire à l'application de toutes les modifications des journaux de relais est passé (attendez que « Exec_Master_Log_Pos » = « Read_Master_Log_Pos » dans la sortie de « `SHOW SLAVE STATUS \G` »), la réplication peut être entièrement arrêtée sur le serveur qui va devenir le nouveau maître :

```
blue> STOP SLAVE;
```

Si vous utilisez la réplication semi-synchrone, activez à présent le plug-in côté maître sur le nouveau maître :

```
blue> SET GLOBAL rpl_semi_sync_master_enabled = on;
```

Avant de démarrer une réplication depuis le nouveau maître (bleu) vers l'esclave restant (vert), vous devez déterminer la position actuelle dans le journal binaire du nouveau maître :

```
blue> show master status \G
*************************** 1. row ***************************
            File: blue-bin.000001
        Position: 807
    Binlog_Do_DB:
Binlog_Ignore_DB:
```

Sur l'esclave restant (« vert »), « redéfinissez le maître » sur le nouveau maître :

```
green> stop slave;
green> CHANGE MASTER TO MASTER_HOST='192.168.0.34',
    -> MASTER_USER='repl_user',
    -> MASTER_PASSWORD='billy',
    -> MASTER_LOG_FILE='blue-bin.000001',
    -> MASTER_LOG_POS=807;
green> start slave;
```

Si vous utilisez la réplication semi-synchrone, assurez-vous que l'esclave restant est enregistré pour la réplication asynchrone en questionnant le nouveau maître (le résultat doit être 1) :

```
blue> SHOW STATUS LIKE 'Rpl_semi_sync_master_clients';
+------------------------------+-------+
| Variable_name                | Value |
+------------------------------+-------+
| Rpl_semi_sync_master_clients | 1     |
+------------------------------+-------+
```

La réplication est de nouveau exécutée, comme illustré dans la Figure 11.

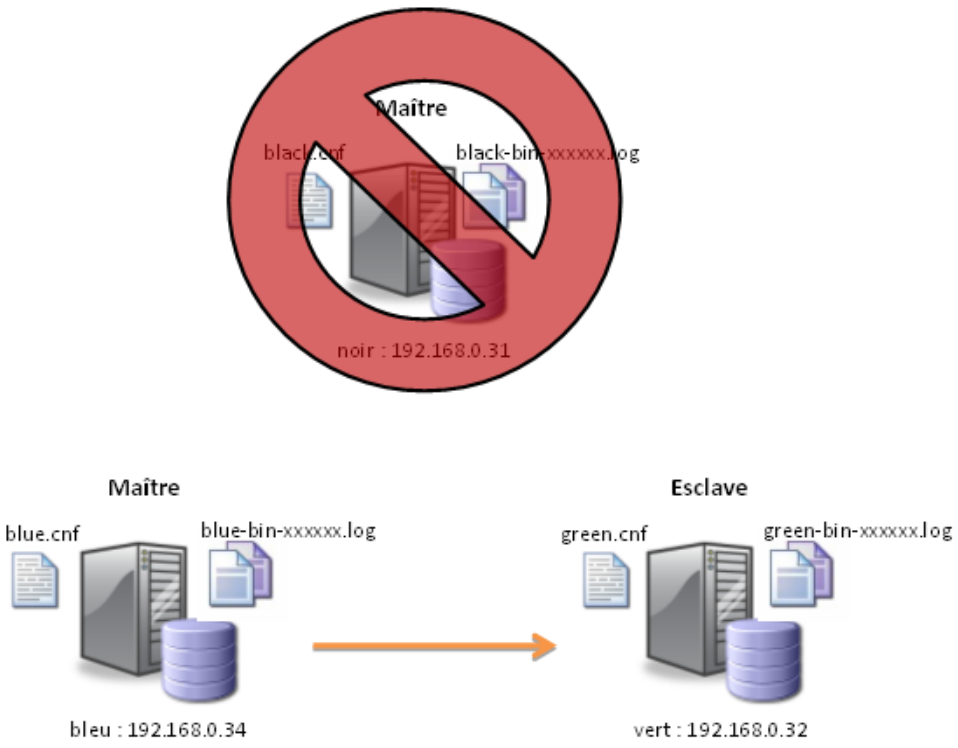

Figure 11 Réplication redémarrée avec un nouveau maître

c. Rediriger les écritures vers le nouveau maître une fois le journal de relais appliqué

L'application peut désormais commencer à envoyer des écritures vers le nouveau maître (et des lectures vers le nouveau maître et l'esclave restant). Cette action peut être activée à l'aide du connecteur, d'un équilibrage de charge ou au sein de l'application.

d. Synchroniser le maître échoué avec le nouveau maître

Par la suite, il peut être possible d'ajouter le maître d'origine (noir) en tant qu'esclave du nouveau maître (au moins initialement) dans la configuration. Pour cela, il existe deux options :

1. Traiter le maître d'origine en tant que nouvel esclave et suivre les étapes 3 à 6 de la section 7. Cette option est préférable si de nombreuses mises à jour ont été appliquées au nouveau maître avant l'ajout du maître d'origine ou si des modifications effectuées sur le maître d'origine n'ont pas été répliquées sur le nouveau maître (par exemple, lors d'une défaillance non contrôlée du maître d'origine lorsque la réplication semi-synchrone n'est pas utilisée).
2. Remettre à jour le maître d'origine en l'ajoutant en tant qu'esclave du nouveau maître, afin de l'actualiser avec toutes les modifications effectuées depuis sa promotion en tant que maître. Le reste de cette section présente cette approche.

Dans les deux cas, libérez le verrou de lecture sur le maître d'origine, si ce n'est déjà fait :

```
black> UNLOCK TABLES;
```

Si vous utilisez la réplication semi-synchrone, activez à présent ce plug-in sur le maître d'origine (nouvel esclave) :

```
black> SET GLOBAL rpl_semi_sync_slave_enabled = on;
```

Démarrez le nouvel esclave à partir du point enregistré à l'étape 4 :

```
black> CHANGE MASTER TO MASTER_HOST='192.168.0.34',
    -> MASTER_USER='repl_user',
    -> MASTER_PASSWORD='billy',
    -> MASTER_LOG_FILE='blue-bin.000001',
    -> MASTER_LOG_POS=807;
black> start slave;
```

A présent, nous exécutons de nouveau 1 maître et 2 esclaves, comme illustré dans la Figure 12.

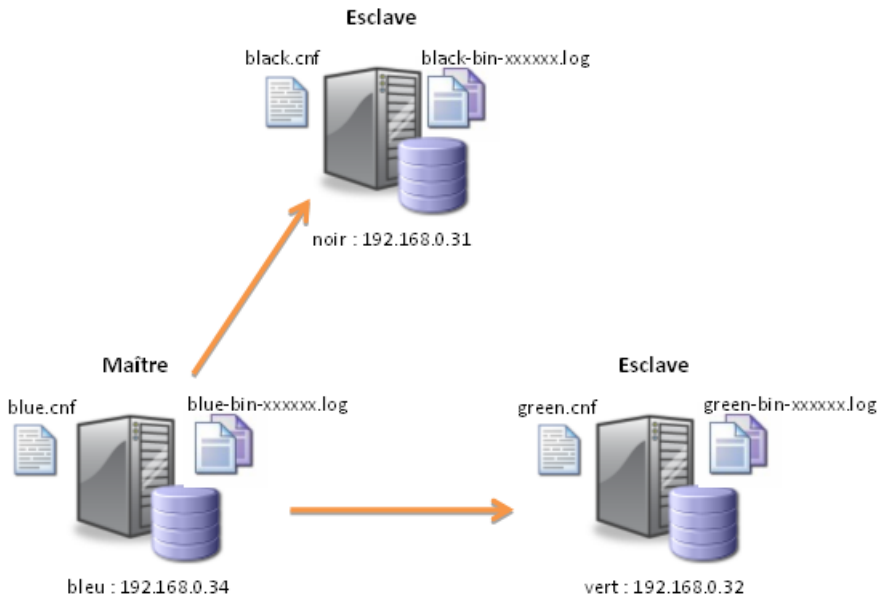

Figure 12 Retour à 1 maître et 2 esclaves

Une dernière étape (facultative) consiste à réexécuter cette procédure afin de rétablir le maître d'origine (noir).

11 Différences lors d'une réplication avec MySQL Cluster

MySQL Cluster est une base de données en cluster évolutive et hautement performante, développée initialement pour certaines applications parmi les plus exigeantes dans le secteur des télécommunications. Ces applications de télécommunication nécessitaient souvent une disponibilité de la base de données supérieure à 99,999 %.

MySQL Cluster peut être utilisé en tant que moteur de stockage enfichable, mais son architecture est fondamentalement différente de celle des autres moteurs de stockage MySQL (par exemple InnoDB et MyISAM), ce qui affecte le fonctionnement et la mise en œuvre de la réplication. L'architecture est illustrée dans la Figure 13 ci-dessous. En termes d'impact sur la réplication, sa principale caractéristique architecturale ne repose pas sur le stockage des données dans une instance MySQL Server, mais sur leur distribution vers plusieurs nœuds de données. Une réplication synchrone entre les nœuds de données dans le Cluster offre une haute disponibilité. Notez que cette réplication synchrone n'est pas associée à la réplication présentée dans ce livre blanc. Les données sont accessibles directement à partir des nœuds de données (par exemple via l'API C++ native), ou un ou plusieurs serveurs MySQL peuvent être utilisés pour fournir l'accès SQL. Chaque serveur MySQL peut lire ou écrire n'importe quelle ligne d'une table et la modification est immédiatement visible sur les autres serveurs MySQL.

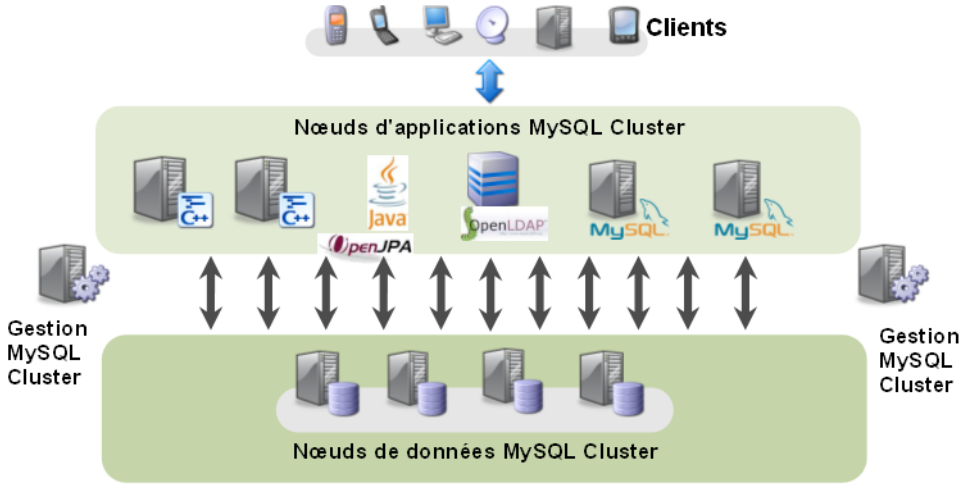

Figure 13 Architecture MySQL Cluster

La réplication MySQL est généralement utilisée avec MySQL Cluster pour fournir la redondance géographique : la réplication interne (synchrone) fournit la haute disponibilité entre les nœuds de données colocalisés au sein du datacenter et la réplication MySQL (asynchrone) vers un site distant protège contre tout échec grave sur le site.

Quel est l'impact sur la réplication MySQL ? Des modifications peuvent être effectuées à partir de tout serveur MySQL ou directement dans les nœuds de données. Un mécanisme a été mis en œuvre pour regrouper toutes les modifications dans les journaux binaires d'un ou de plusieurs serveurs MySQL désignés, qui agissent alors en tant que maîtres de la réplication MySQL. Elle est exécutée par le thread NDB Binlog Injector. Ce thread garantit que toutes les modifications, quel que soit l'emplacement de leur application dans le Cluster, sont écrites dans le journal binaire dans l'ordre approprié.

Comme illustré dans la Figure 14, elle renforce l'architecture système de réplication. Deux serveurs MySQL sont illustrés dans lesquels les journaux binaires contiennent les mêmes modifications. Chaque serveur peut être utilisé en tant que maître pour répliquer ces modifications sur un ou plusieurs déploiements MySQL Cluster esclaves et/ou sur un serveur MySQL qui stocke des données à l'aide des moteurs de stockage InnoDB ou MyISAM. Les journaux binaires de chaque maître doivent contenir les même modifications, mais ils sont indépendants. Pour faciliter la reprise après échec d'un esclave en basculant d'un maître vers un autre, l'outil « Epoch » au niveau du Cluster est utilisé pour représenter l'état d'un Cluster sur un moment donné unique. Ces techniques avancées de reprise après échec dépassent l'objet de ce livre. Pour plus d'informations, vous pouvez toutefois consulter le guide de référence de MySQL Cluster à l'adresse :
http://dev.mysql.com/doc/mysql-cluster-excerpt/5.1/en/mysql-cluster-replication-failover.html

MySQL Cluster prend en charge l'exécution de la réplication MySQL dans une configuration active-active à plusieurs maîtres ou même avec des anneaux de réplication. De plus, MySQL Cluster peut fournir une détection ou une résolution des conflits qui permet de gérer l'écriture des modifications conflictuelles sur les mêmes lignes dans ces maîtres différents. Le fonctionnement de cette détection / résolution des conflits nécessite des tâches supplémentaires au niveau de l'application, qui dépassent l'objet de ce livre blanc. Pour plus d'informations, vous pouvez toutefois consulter le guide de référence de MySQL Cluster : http://dev.mysql.com/doc/mysql-cluster-excerpt/5.1/en/mysql-cluster-replication-multi-master.html

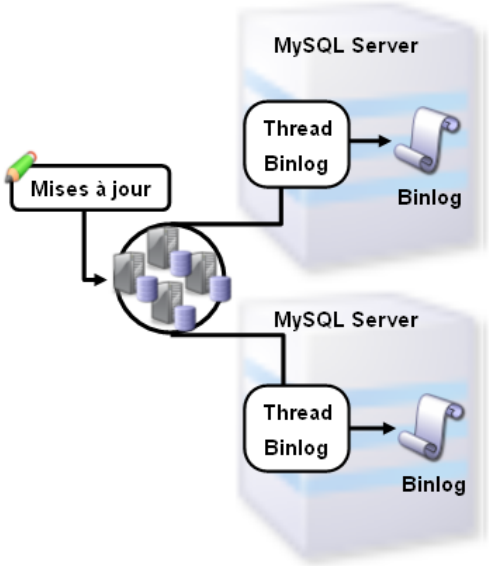

Figure 14 Plusieurs maîtres au sein d'un Cluster

Les versions MySQL Cluster sont produites selon un programme différent de celui des principales versions MySQL. Par conséquent, les fonctionnalités de réplication MySQL ne sont pas toujours disponibles dans la dernière version MySQL Cluster. Par exemple, au moment de la rédaction de ces lignes, MySQL Cluster 7.1 est la dernière version disponible, qui utilise une version modifiée de MySQL 5.1 pour les serveurs MySQL. Par conséquent, les améliorations intégrées à MySQL 5.5 ne sont pas disponibles. La fonction de promotion et de rétrogradation intégrée à MySQL Cluster 7.1.3 est une exception.

Lors de la réplication à partir de MySQL Cluster, la réplication par ligne (plutôt que la réplication par instruction) est toujours utilisée.

12 Surveillance de la réplication avec MySQL Enterprise Monitor

MySQL Enterprise Monitor avec Query Analyzer est une application Web distribuée que vous déployez en toute sécurité derrière votre pare-feu d'entreprise. MySQL Enterprise Monitor surveille en permanence l'ensemble de vos serveurs MySQL et vous informe de manière proactive des problèmes potentiels et des réglages possibles afin d'éviter toute interruption onéreuse. Il offre également les conseils avisés de MySQL sur les problèmes qu'il détecte, pour vous aider à optimiser votre temps et vos systèmes MySQL.

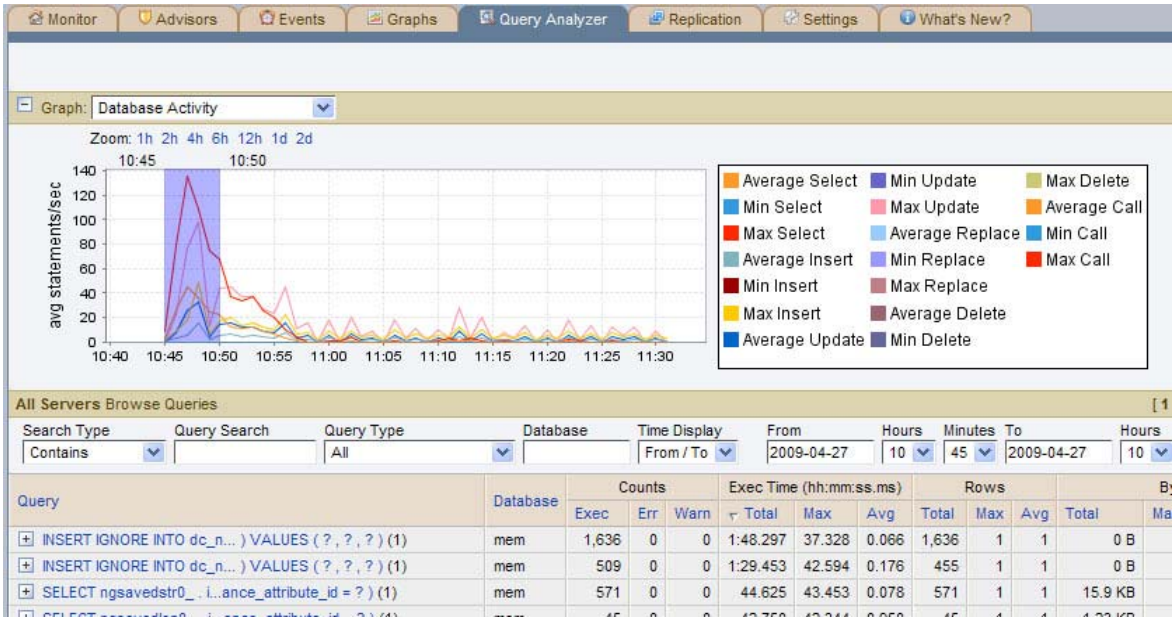

Figure 15 MySQL Enterprise Monitor

Enterprise Monitor offre les fonctionnalités spécifiques à la réplication MySQL suivantes :

- Les relations de réplication sont automatiquement détectées, regroupées et maintenues, sans installation ni configuration DBA
- Une vue consolidée et en temps réel des performance et de l'état de toutes les relations maître/esclave, comme illustrée dans la Figure 16 ci-dessous.

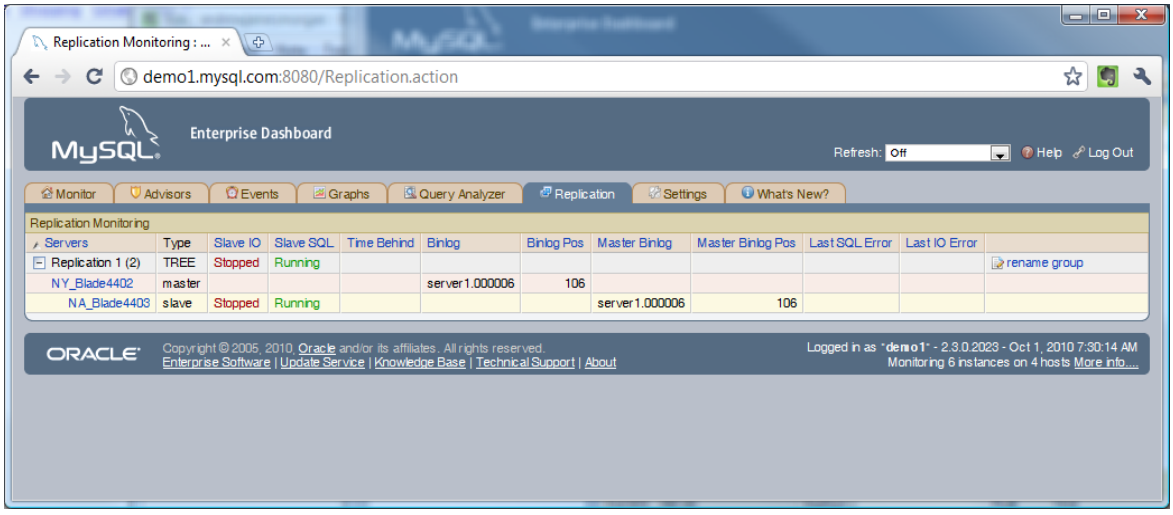

Figure 16 Statistiques de réplication dans MySQL Enterprise Monitor

MySQL Enterprise Monitor est disponible en option commerciale pour les clients MySQL.

13 Conclusion

La réplication MySQL a démontré son efficacité en tant que solution à l'extrême évolutivité des applications fondées sur une base de données dans les environnements les plus exigeants, tant sur le Web qu'au sein de l'entreprise.

Ce livre blanc vous a présenté les avantages techniques et commerciaux du déploiement de la réplication MySQL, avec des guides pas-à-pas de prise en main pratiques.

Comme le montre la version MySQL 5.5, la réplication est un secteur très actif en plein développement, qui fait l'objet d'améliorations permanentes dans divers domaines tels que l'intégrité des données, la flexibilité des performance et du déploiement.

Les ressources ci-dessous viennent compléter ce livre blanc, et peuvent accélérer votre évaluation et votre déploiement de la réplication MySQL.

14 Ressources

Télécharger MySQL 5.5 : http://dev.mysql.com/downloads/mysql/5.5.html

Guide de l'utilisateur de la réplication MySQL : http://dev.mysql.com/doc/refman/5.5/en/replication.html

Visionner le Webminaire « Delivering Scalability and High Availability with MySQL 5.5 Replication Enhancements » : http://www.mysql.com/news-and-events/on-demand-webinars/display-od-572.html

Visionner le Webminaire « MySQL Cluster Geographic Replication » : http://www.mysql.com/news-and-events/on-demand-webinars/display-od-415.html

Documentation relative à la réplication MySQL Cluster : http://dev.mysql.com/doc/mysql-cluster-excerpt/5.1/en/mysql-cluster-replication.html

MySQL chez Ticketmaster (grand utilisateur de réplication MySQL) : http://www.mysql.com/customers/view/?id=684