Introduction à la réplication de bases de données

(Linux Magazine France HS n°18 - Février 2004)

Etre capable de faire de la haute disponibilité de machines est une chose qui présente de nombreux avantages. Malheureusement elle ne résout pas à elle seule la disponibilité des services hébergés, et la solution peut parfois être fort complexe dans le cadre des bases de données. Ceci est dû au fait que, contrairement à un serveur web, un site ftp ou tout autre service, il s'agit de gérer un jeu de données qui peuvent être modifiées à fréquence élevée.

1. Définition

Une application de base de données repose sur un modèle client-serveur. Dans ce modèle, le client se connecte au SGBD pour passer des ordres. Ces ordres sont de deux natures : lecture (on parle alors de requêtes) ou mise à jour (on parle alors de transactions). Pour les transactions il y a une modification des données sur le serveur, mais cela reste des ordres de courte durée. A l'inverse, dans le cas d'une lecture, il n'y a pas de modification des données mais les traitements peuvent être longs et porter sur une grande masse de données. On comprend donc aisément que, dans le cadre d'un site web par exemple, un nombre important de requêtes peut emboliser partiellement (ou complètement) le serveur. Il existe plusieurs solutions pour palier à ce genre de problèmes et, ça tombe bien, la réplication en est une .

L'objectif principal de la réplication est de faciliter l'accès aux données en en augmentant la disponibilité. Soit parce que les données sont copiées sur différents sites permettant de répartir les requêtes, soit parce qu'un site peut prendre la relève lorsque le serveur principal s'écroule. Une autre application tout aussi importante est la synchronisation des systèmes embarqués non connectés en permanence.

Ce qui peut se résumer à l'aide des trois types de scénarii suivants :

- deux serveurs distants sur lesquels les données doivent être consistantes ;
- deux serveurs, un comme serveur principal, l'autre comme serveur de backup à chaud ;
- plusieurs serveurs en cluster utilisés pour de l'équilibrage de charge et de la tolérance à la panne.

2. Principe

Le principe de la réplication, qui met en jeu au minimum deux SGBD, est assez simple et se déroule en trois temps :

1. La base maître reçoit un ordre de mise à jour (INSERT, UPDATE ou DELETE).
2. Les modifications faites sur les données sont détectées et stockées (dans une table, un fichier, une queue) en vue de leur propagation.
3. Un processus de réplication prend en charge la propagation des modifications à faire sur une seconde base dite esclave. Il peut bien entendu y avoir plus d'une base esclave.

Bien entendu il est tout à fait possible de faire de la réplication dans les deux sens (de l'esclave vers le maître et inversement). On parlera dans ce cas-là de réplication bidirectionnelle ou symétrique. Dans le cas contraire la réplication est unidirectionnelle (seulement du maître vers l'esclave) et on parle de réplication en lecture seule ou asymétrique. De plus la réplication peut être faite de manière synchrone ou asynchrone. Dans le premier cas la résolution des conflits éventuels entre deux sites intervient avant la validation des transactions ; Dans le second cas, la résolution est faite dans des transactions séparées. Il est donc possible d'avoir quatre modèles de réplication :

- Réplication asymétrique (maître/esclave) avec propagation asynchrone ;
- Réplication asymétrique (maître/esclave) avec propagation synchrone ;

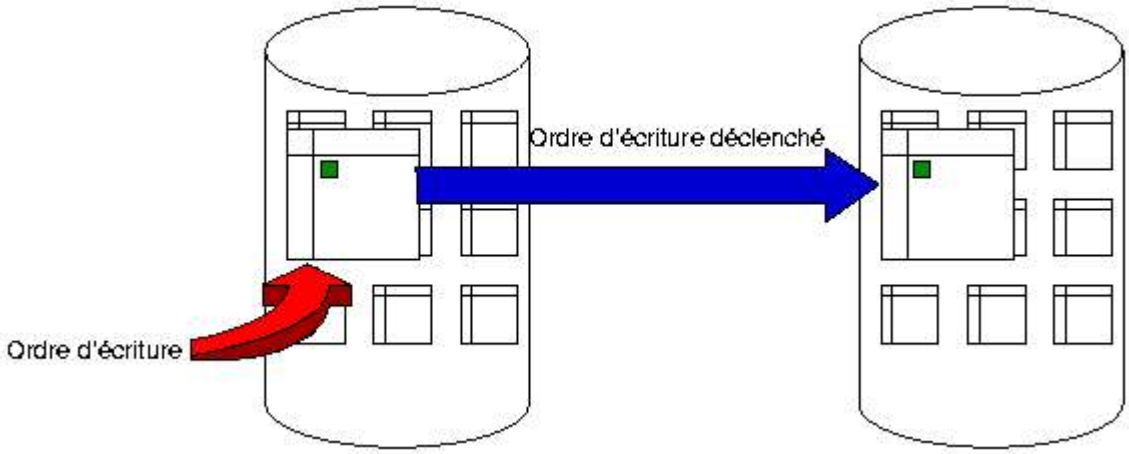
- Réplication symétrique ou Peer-to-peer (update everywhere) avec propagation asynchrone ;
- Réplication symétrique avec propagation synchrone.

3. La mise à jour synchrone

La réplication synchrone est aussi appelée “réplication en temps réel”. La synchronisation est effectuée en temps réel puisque chaque requête est déployée sur l’ensemble des bases de données avant la validation (commit) de la requête sur le serveur où la requête est exécutée. Ce type de réplication assure un haut degré d’intégrité des données mais requiert une disponibilité permanente des serveurs et de la bande passante. Ce type de réplication, fortement dépendant des pannes du systèmes, nécessite de gérer des transactions multisites coûteuses en ressources.

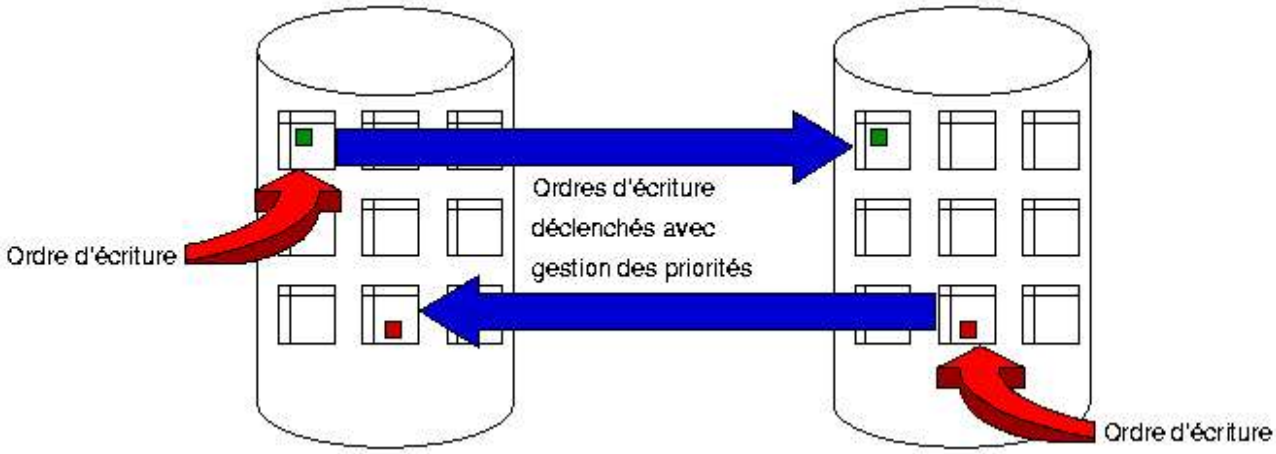
La réplication synchrone asymétrique

Une modification sur le site primaire sera propagée aux sites secondaires à l’aide par exemple d’un trigger sur la table modifiée. La table est modifiée en temps réel sur les autres sites, ces modifications faisant parties de la transaction.



La réplication synchrone symétrique

Dans ce cas, il n’y a pas de table maître, mais chaque table possède ses triggers, déclenchés lors d’une modification. Il est alors nécessaire de définir des priorités et de gérer les blocages des tables en attendant qu’une modification soit entièrement propagée. D’autre part, les triggers doivent différencier les mises à jour issues d’une réplication des mises à jour de requête directes.



4. La mise à jour asynchrone

La réplication asynchrone (aussi appelée “Store and Forward” pour << stocker et propager >>) stocke les opérations intervenues sur une base de données dans une queue locale pour les propager

plus tard à l'aide d'un processus de synchronisation.

Ce type de réplication est plus flexible que la réplication synchrone. Il permet en effet de définir les intervalles de synchronisation, ce qui permet à un site de fonctionner même si un site de réplication n'est pas accessible. L'utilisateur peut ainsi déclarer que la synchronisation sera effectuée la nuit, à une heure de faible affluence. Si le site distant est victime d'une panne, l'absence de synchronisation n'empêche pas la consistance de la base maître. De même une panne de réseau laissera les deux bases, maître et esclave dans des états de consistance. Les opérations sur les données sont plus rapides, puisqu'une requête n'est pas immédiatement déployée. Le traffic sur le réseau est de ce fait plus compact. Par contre, le planning de réplication est dans ce cas plus complexe, puisqu'il s'agit de gérer les conflits émanant d'un éventuel accès en écriture sur une base esclave entre deux mises à jour.

La réplication asynchrone asymétrique

Les mises à jour sont stockées dans une file d'attente et ne seront propagées que lors d'un déclenchement programmé.

La réplication asynchrone symétrique

Toute modification sur toute table de toute base est stockée dans une file pour être rejouée ultérieurement. De fortes incohérences des données sont à craindre.

5. En théorie et en pratique

Dans les deux articles suivant nous allons vous présenter les solutions existantes pour faire de la réplication avec PostgreSQL avant d'aborder la pratique avec MySQL. Sachez cependant que la réplication est possible avec n'importe quel base (libre ou non), mais que le prix de sa mise en place varie énormément d'un système à l'autre.

Stéphane SCHILDKNECHT
Grégoire Lejeune

• • •

Réplication avec MySQL

(Linux Magazine France HS n°18 - Février 2004)

MySQL est certainement la base de données la plus utilisée par le quidam souhaitant créer son propre si web ou simplement découvrir rapidement le monde des SGBD. Elle doit très certainement sa notoriété à sa grande simplicité d'utilisation et d'administration. Et bien que les amoureux du tunning la trouvent souvent trop simpliste, il n'en reste pas moins qu'elle permet de mettre en place la réplication avec un minimum d'efforts la ou d'autre obligent d'avoir sous la main une brochette d'experts.

Dans cet article nous allons mettre en place une réplication entre deux bases MySQL en mode unidirectionnelle/asynchrone. En fait la propagation synchrone n'est pas possible avec MySQL - tout au moins pas avec le système de réplication fournis en standard avec la base.

1. Installation

L'architecture minimale pour faire de la réplication est composée de deux machines. Il est tout à fait possible de faire de la réplication entre deux instances de bases placées sur la même machine mais l'intérêt est limité.

Dans le cas présent, nous aurons la machine "uranium" servant de master et "helium" pour le slave. C'est deux machines sont installées sous Linux Debian Sid. Nous allons faire simple pour le moment et considérer que ni la base master, ni la base slave ne sont installées sur les machines. Nous utiliserons la version 4.0.16 de MySQL.

Installation du master

Vous l'avez certainement compris, la première chose à faire consiste à installer la base. Pour cela je vous laisse faire en fonction de vos habitudes.

Une fois la base prête, nous devons créer un utilisateur spécial qui va contrôler la réplication. Il doit pouvoir accéder à la base master depuis la machine où se trouvera le slave.

```
uranium:~# mysql -uroot -p mysql
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or g.
Your MySQL connection id is 78 to server version: 4.0.16-log
Type 'help;' or 'h' for help. Type 'c' to clear the buffer.
master> grant FILE, REPLICATION SLAVE on *.* to repl@'%' IDENTIFIED BY 'repl';
Query OK, 1 rows affected (0.14 sec)
master>
```

Nous attribuons à cet utilisateur "repl" les droits FILE et REPLICATION SLAVE. Si vous utilisez une version antérieure à la 3.23.29, le droit REPLICATION SLAVE n'existe pas et FILE suffit donc. Si votre version est supérieure (ou égale) à la 3.23.29, vous pouvez ignorer le droit FILE. Pour des raisons de sécurité évidentes, nous ne donnerons pas plus de droits à cet utilisateur en le cantonnant ainsi dans son rôle de contrôleur de la réplication.

La seconde étape implique d'arrêter la base.

```
master> exit
Bye
uranium:~# /etc/init.d/mysql stop
Stopping MySQL database server: mysqld.
uranium:~#
```

Il faut maintenant sauvegarder les données de la base afin de pouvoir les transférer, plus tard, sur le slave. Pour savoir où se trouve le répertoire de données, regarder l'entrée datadir dans le fichier de configuration de la base (my.cnf).

```
uranium:~# locate my.cnf
/etc/mysql/my.cnf
uranium:~# cat /etc/mysql/my.cnf | grep datadir
datadir         = /var/lib/mysql
uranium:~# tar zcf /tmp/mysql-data-master.tgz /var/lib/mysql/
uranium:~#
```

Avant de pouvoir relancer la base, vous devez modifier le fichier de configuration. Ouvrez ce fichier avec l'éditeur de votre choix et ajoutez les deux lignes suivantes à la fin de la section [mysqld] :

/etc/mysql/my.cnf :

```
...
[mysqld]
...
server-id = 1
log-bin   = /var/log/mysql-bin.log
...
```

L'entrée server-id sert à identifier de manière unique une base dans un groupe de réplication. Vous pouvez lui attribuer la valeur de votre choix en fonction de votre logique. log-bin permet d'activer le log binaire tout en précisant l'emplacement et le nom de ce fichier de log. C'est dans ce dernier que vont être stockés les ordres de modification faite sur le master avant d'être envoyés au slave.

C'est terminé pour le master, vous pouvez redémarrer la base.

Installation du slave

Comme pour le master, il faut bien entendu commencer par installer la base. Ma remarque sur le sujet sera la même que pour l'installation de la base master. Une seule différence cependant, il est inutile de démarrer la base (ou si votre système de package ne vous laisse pas le choix, arrêtez-la).

Editez le fichier de configuration de la base et ajouter à la fin de la section [mysqld] les lignes suivantes :

/etc/mysql/my.cnf :

```
...
[mysqld]
...
server-id       = 2
log-bin         = /var/log/mysql-bin.log
master-host     = uranium
master-user     = repl
master-password = repl
master-port     = 3306
master-connect-retry = 10
...
```

Nous retrouvons le server-id. Faites bien attention de lui attribuer une valeur différente de celle du master. Les entrées master-host, master-user, master-password et master-port sont suffisamment explicites pour que je ne m'étende pas dessus. La valeur de master-connect-retry indique, en secondes, le délais entre deux tentatives de connections au master en cas d'échec. Vous pouvez vous demander pourquoi nous activons le log binaire ici. Il s'agit d'un principe de protection qui prendra tout son intérêt lorsque nous parlerons de la tolérance de panne.

Il faut ensuite récupérer l'archive de sauvegarde des donnés que nous avons fait lors de l'installation du master. Une fois ceci fait, vous pouvez supprimer le répertoire de données qui a été créé lors de l'installation de la base slave (ou en faire une copie de sauvegarde) et le remplacer par les données issues du master.

```
helium:~# cd /
helium:/# scp root@uranium:/tmp/mysql-data-master.tgz /tmp/
root@uranium's password:
```

```
mysql-data-master.tgz                          100%   3.7K   1.0MB/s   00:02
helium:/# mv /var/lib/mysql /var/lib/mysql_old
helium:/# tar zxf /tmp/mysql-data-master.tgz
helium:/#
```

C'est terminé pour le slave, vous pouvez démarrer la base.

2. Tester la réplication

Nous sommes maintenant prêts à vérifier si cela fonctionne correctement. Nous profiterons de ces tests pour voir quelques commandes utiles pour l'administration d'un tel système et pour expliquer plus précisément comment fonctionne la réplication avec MySQL.

Etant partis d'un "système vide", nous n'avons pour le moment que deux bases : mysql et test, nous allons créer une base de test "lmtest" dans laquelle nous allons créer une unique table "tbltest". Bien entendu, nous faisons tout cela depuis le master :

```
master> create database lmtest;
Query OK, 1 row affected (0.00 sec)
master> connect lmtest;
Connection id:    9
Current database: lmtest
master> CREATE TABLE tbltest (
    ->   id       INT(11) AUTO_INCREMENT PRIMARY KEY,
    ->   username VARCHAR(128) NOT NULL,
    ->   age      INT(11),
    ->   ts       TIMESTAMP DEFAULT 'CURRENT_TIMESTAMP'
    -> );
Query OK, 0 rows affected (0.00 sec)
master> show tables;
+------------------+
| Tables_in_lmtest |
+------------------+
| tbltest          |
+------------------+
1 row in set (0.00 sec)
master>
```

Remarque : pour éviter la confusion, j'utilise master> comme prompt sur le master et slave> pour le slave.

Bon, regardons maintenant ce que nous avons sur le slave. Si vous ne vous êtes pas trompé lors de l'installation, voici ce que vous devez obtenir :

```
slave> show databases;
+----------+
| Database |
+----------+
| lmtest   |
| mysql    |
| test     |
+----------+
5 rows in set (0.00 sec)
slave> connect lmtest;
Connection id:    10
Current database: lmtest
slave> show tables;
+------------------+
| Tables_in_lmtest |
+------------------+
| tbltest          |
+------------------+
1 row in set (0.00 sec)
slave> desc tbltest;
+----------+--------------+------+-----+---------+----------------+
```

```
| Field    | Type           | Null | Key | Default | Extra          |
+----------+----------------+------+-----+---------+----------------+
| id       | int(11)        |      | PRI | NULL    | auto_increment |
| username | varchar(128)   |      |     |         |                |
| age      | int(11)        | YES  |     | NULL    |                |
| ts       | timestamp(14)  | YES  |     | NULL    |                |
+----------+----------------+------+-----+---------+----------------+
4 rows in set (0.00 sec)
slave>
```

Première constatation, le système de réplication de MySQL se démarque des autres par le fait que tout est répliqué. En effet avec des bases tel que Oracle ou DB2, vous avez la possibilité de répliquer seulement une base, voir même d'être plus fin en ne sélectionnant que certaines tables. Avec MySQL c'est tout ou rien. Dans la plus part des cas on serait tenté de dire que cela n'est pas grave. Pourtant cela n'est pas neutre.

Bon, poussons un peu plus loin et insérons des données :

```
master> INSERT INTO tbltest( username, age ) VALUES ( 'greg', 30 );
Query OK, 1 row affected (0.11 sec)
master> select * from tbltest;
+----+----------+------+----------------+
| id | username | age  | ts             |
+----+----------+------+----------------+
|  1 | greg     |   30 | 20040110152747 |
+----+----------+------+----------------+
1 row in set (0.00 sec)
master>
```

Vérification sur le slave :

```
slave> select * from tbltest;
+----+----------+------+----------------+
| id | username | age  | ts             |
+----+----------+------+----------------+
|  1 | greg     |   30 | 20040110152747 |
+----+----------+------+----------------+
1 row in set (0.01 sec)
slave>
```

Tout fonctionne sans problèmes.

3. Ecrire sur le slave

Pour aller plus loin et comprendre comment fonctionne la réplication, vérifions ce qui se passe si nous insérons une ligne sur la slave puis quand nous faisons une modification sur le master.

```
slave> INSERT INTO tbltest( username, age ) VALUES ( 'arthur', 5 );
Query OK, 1 row affected (0.00 sec)
slave> select * from tbltest;
+----+----------+------+----------------+
| id | username | age  | ts             |
+----+----------+------+----------------+
|  1 | greg     |   30 | 20040110152747 |
|  2 | arthur   |    5 | 20040110180618 |
+----+----------+------+----------------+
2 rows in set (0.00 sec)
slave>
```

Jusque la tout semble normal  Par contre si vous faite un select sur le master, vous pouvez attendre aussi longtemps que vous le voulez, jamais vous ne verrez s'afficher les nouvelles données. Pire, ajoutons une ligne sur le master :

```
master> INSERT INTO tbltest( username, age ) VALUES ( 'colyne', 2 );
```

```
Query OK, 1 row affected (0.00 sec)
master> select * from tbltest;
+----+----------+------+----------------+
| id | username | age  | ts             |
+----+----------+------+----------------+
|  1 | greg     |   30 | 20040110152747 |
|  2 | colyne   |    2 | 20040110153627 |
+----+----------+------+----------------+
2 rows in set (0.00 sec)
master>
```

voici ce que nous avons ensuite sur le slave :

```
slave> select * from tbltest;
+----+----------+------+----------------+
| id | username | age  | ts             |
+----+----------+------+----------------+
|  1 | greg     |   30 | 20040110152747 |
|  2 | arthur   |    5 | 20040110180618 |
+----+----------+------+----------------+
2 rows in set (0.00 sec)
slave>
```

Harg ! La ligne n'a pas été répliquée ! Et vous pouvez ajouter autant de lignes que vous le souhaitez sur le master, jamais vous ne les verrez apparaître sur le slave. Bien entendu ce problème n'est heureusement pas insoluble. Mais encore faut-il savoir ou est le problème. Pour cela, sur le slave, tapez la commande suivant :

```
slave> SHOW SLAVE STATUS;
... | Last_errno | Last_error                                          | ...
... | 1062       | Error 'Duplicate entry '2' for key 1' on query  | ...
```

J'ai légèrement tronqué la sortie pour ne vous montrer que la partie qui nous intéresse. Comme on pouvait si attendre, le fait d'avoir insérer des données sur le slave bloque la réplication. C'est somme toute logique dans une politique de conservation des données. En effet le système n'est pas à même de choisir si c'est la slave qui à raison ou le master. Il attend donc une intervention humaine (bonne nouvelle pour les parano de SF ).

Vu la faible quantité de donnée impacté dans le cas présent, le plus simple consiste à identifier qui a raison (slave ou master) et modifier les données en conséquences. Donc si nous considérons dans le cas présent que la ligne sur le slave est mauvaise, il suffit tout simplement de la supprimer.

```
slave> delete from tbltest where id='2';
```

Cependant cela ne suffit pas. Si vous essayez, vous constaterez que la réplication ne se fera pas plus malgré la suppression de la ligne responsable du conflit. Il faut en effet relancer la réplication. Pour cela rien de plus simple il suffit d'arrêter puis de redémarrer l'esclave. Bien entendu si vous avez une base en production cela peut pauser des problèmes. En général on ne coupe pas une base de données. Et bien là aussi c'est la même chose. Nous n'allons pas arrêter la base, juste stopper le mode esclave et le relancer. Pour cela tapez les commandes suivantes :

```
slave> slave stop;
Query OK, 0 rows affected (0.00 sec)
slave> slave start;
Query OK, 0 rows affected (0.00 sec)
slave> select * from tbltest;
+----+----------+------+----------------+
| id | username | age  | ts             |
+----+----------+------+----------------+
|  1 | greg     |   30 | 20040110152747 |
|  2 | colyne   |    2 | 20040110153627 |
+----+----------+------+----------------+
2 rows in set (0.00 sec)
slave>
```

Simple non ? En fait dans le cas présent c'est effectivement très simple, mais si vous vous lancez vraiment dans la réplication attendez-vous à devoir traiter des cas beaucoup plus épineux.

4. Tolérance de panne

Maintenant que vous savez que tout cela fonctionne, vous pouvez envisager de faire de la tolérance de panne. Imaginons pour cela que vous ayez un site web avec lequel vous utilisez une base de données - rien de plus classique en fait. Maintenant imaginer que la machine hébergeant votre base de données décide de rendre l'âme. Là généralement c'est la catastrophe. Mais non puisque vous avez mis votre base en réplication ! En effet toutes vos données sont sur le slave donc il suffit de demander à vos scripts d'attaquer le slave.

Bien entendu tout ceci se prépare. En effet il ne faut pas attendre que vous ayez remarqué que le master est dans les choux pour basculer sur le slave. Donc il peut être intéressant d'écrire un script vérifiant l'état du master et d'utiliser un DNS que vous pourrez alors modifier dynamiquement grâce à nsupdate si vous utilisez bind par exemple.

Le second problème interviendra lorsque vous aurez remonté votre (ancien) master. En effet pendant un certain temps c'est le slave qui a recueilli les données. Bien entendu, vous pouvez faire une sauvegarde de vos données du slave, l'arrêter, tout remettre sur le master et refaire la configuration. Malheureusement cela vous oblige à stopper la base et donc à rendre votre service indisponible pendant un certain temps (qui sera généralement plus long que celui que vous avez planifié) ce qui ne plaira certainement pas a tout le monde  .

La "bonne" solution consisterait à "inverser" slave et master. Ceci est tout à fait possible car nous avons activé le log binaire sur le slave. Il suffit donc pour le transformer en master de lui passer la commande SLAVE STOP. Ensuite nous devons récupérer les données pour les copier vers l'ancien master (qui deviens donc la slave - et oui, faut suivre !) Pour cela vous il faut créer une archive des données comme nous l'avons vue lors de la procédure d'installation. Il est impératif que les données ne soient pas modifiées lors de cette opération. Pour cela il suffit d'utiliser la commande FLUSH TABLES WITH READ LOCK, de créer l'archine (avec tar par exemple) puis de faire un UNLOCK TABLES à la fin :

```
slave> FLUSH TABLES WITH READ LOCK;
Query OK, 0 rows affected (0.00 sec)
slave> quit
Bye
helium:~# tar zcf /tmp/mysql-data-slave.tgz /var/lib/mysql/
helium:~# mysql mysql
Welcome to the MySQL monitor.  Commands end with ; or g.
Your MySQL connection id is 24 to server version: 4.0.16-log
Type 'help;' or 'h' for help. Type 'c' to clear the buffer.
slave> SHOW MASTER STATUS;
+---------------+----------+--------------+------------------+
| File          | Position | Binlog_do_db | Binlog_ignore_db |
+---------------+----------+--------------+------------------+
| mysql-bin.012 | 1212     |              |                  |
+---------------+----------+--------------+------------------+
1 row in set (0.00 sec)
slave> UNLOCK TABLES;
Query OK, 0 rows affected (0.00 sec)
slave>
```

Il suffit maintenant de réinstaller la base sur(l'ancien) master comme nous l'avons vu plus haut (installation du slave). Il faut ensuite ajuster la position de lecture du log binaire, c'est pour cela que nous avons fait un SHOW MASTER STATUS avant de délocker les tables :

```
master> slave stop;
Query OK, 0 rows affected (0.00 sec)
master> CHANGE MASTER TO
    -> MASTER_HOST='uranium',
```

```
    -> MASTER_PORT=3306,
    -> MASTER_USER='repl',
    -> MASTER_PASSWORD='repl',
    -> MASTER_LOG_FILE='mysql-bin.012',
    -> MASTER_LOG_POS=1212;
Query OK, 0 rows affected (0.00 sec)
master> slave start;
Query OK, 0 rows affected (0.01 sec)
master>
```

A partir de la tout doit fonctionner comme avant :

```
slave> INSERT INTO tbltest( username, age ) VALUES ( 'arthur', 5 );
Query OK, 1 row affected (0.00 sec)
slave> select * from tbltest;
+----+----------+------+----------------+
| id | username | age  | ts             |
+----+----------+------+----------------+
|  1 | greg     |   30 | 20040110152747 |
|  2 | colyne   |    2 | 20040110153627 |
|  3 | morgane  |   30 | 20040110153820 |
|  4 | maguy    |   60 | 20040110215139 |
|  5 | arthur   |    5 | 20040111011340 |
+----+----------+------+----------------+
5 rows in set (0.00 sec)
slave>
```

Et sur le master devenu slave :

```
master> select * from tbltest;
+----+----------+------+----------------+
| id | username | age  | ts             |
+----+----------+------+----------------+
|  1 | greg     |   30 | 20040110152747 |
|  2 | colyne   |    2 | 20040110153627 |
|  3 | morgane  |   30 | 20040110153820 |
|  4 | maguy    |   60 | 20040110215139 |
|  5 | arthur   |    5 | 20040111011340 |
+----+----------+------+----------------+
5 rows in set (0.00 sec)
master>
```

5. Aller plus loin

Maintenant que vous avez les connaissances nécessaires pour mettre en place la réplication vous pouvez envisager des architectures plus complexes avec plusieurs esclaves. Remarquez que vous pouvez parfaitement avoir une machine esclave qui sert de maître à d'autres. Ceci permet de mettre en place une architecture sous forme d'arbre. Dans tous les cas, n'oubliez jamais que vous devez toujours écrire sur la machine racine.