Haute disponibilité des données

par clustering avec MySQL 5.0

Master Management de la sécurité des systèmes industriels et des systèmes d'information

janvier 2006

Noureddine BOUHADDAOUI
Julien VEHENT

Sommaire

Introduction

La problématique de répartition des données est un problème important majeur des infrastructures de gestion de l'information. Alors que l'approche historique consiste à dimensionner un seul serveur de façon à ce qu'il absorbe toute la charge, on observe depuis plusieurs années une tendance inverse, inspirée des techniques de RAID pour les volumes de stockage, qui consiste a prendre un très grand nombre de machines a faible coût pour former un cluster de stockage.

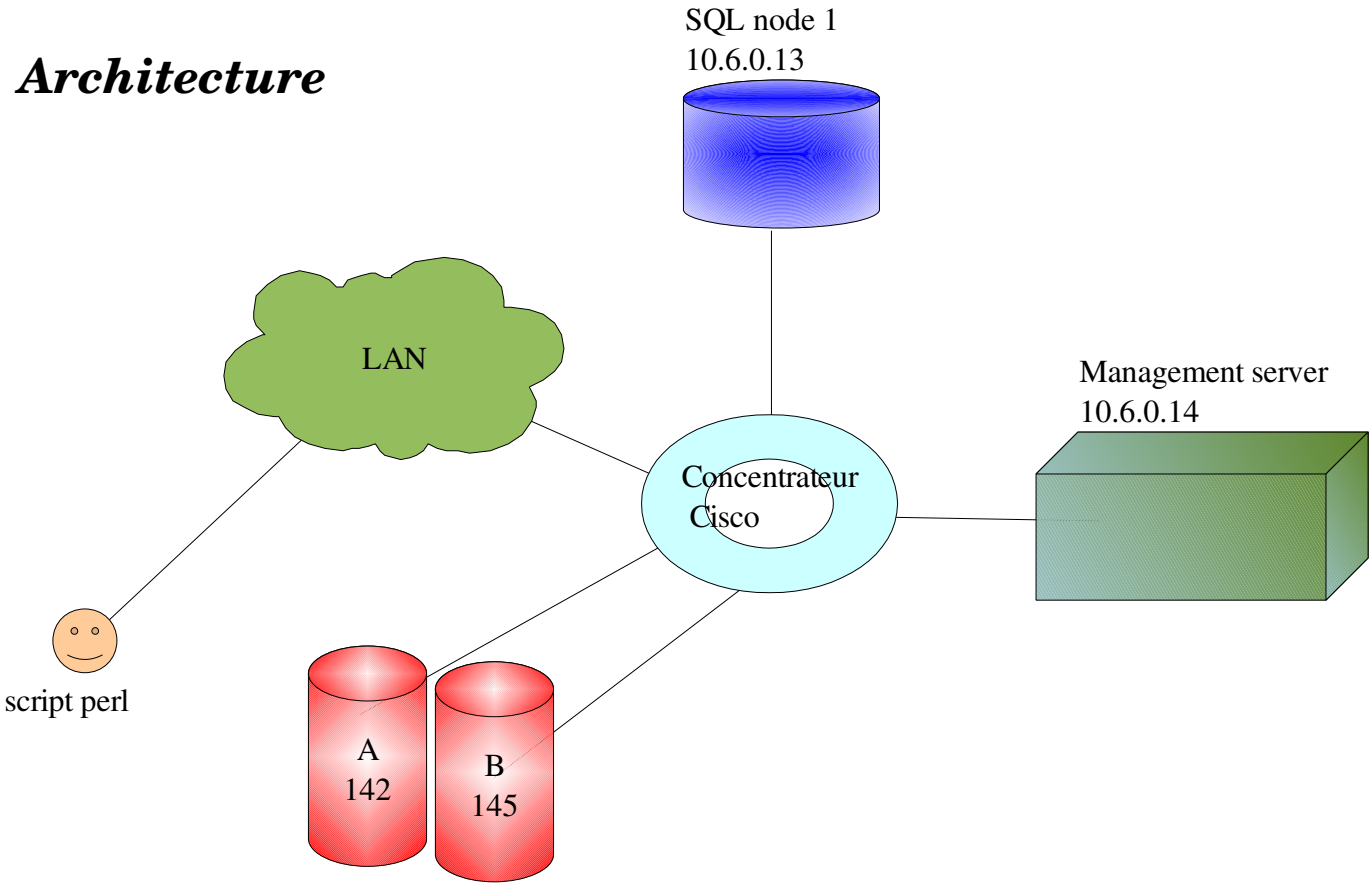
Dans le domaine des systèmes de gestion de bases de données, MySQL fournit une solution élégante pour réaliser un cluster à faible coût permettant d'atteindre un niveau de disponibilité proche des 99,999% annuel.

Un Cluster MySQL est un groupe de processus qui s'exécutent sur plusieurs serveurs MySQL, des noeuds NDBCluster, et des processus d'administration, ainsi que des processus d'accès spécialisés. Tous ces programmes fonctionnent ensemble pour former un Cluster MySQL. Lorsque les données sont stockées dans le moteur NDBCluster, les tables sont réparties sur les noeuds NDBCluster. Ces tables sont directement accessibles depuis tous les autres serveurs MySQL faisant partie du Cluster. Par conséquent, si une application met à jour une ligne, tous les autres serveurs le verront immédiatement.

Les données stockées dans le moteur de table de MySQL Cluster peuvent être dupliquées, et sont capables de gérer une indisponibilité d'un noeud sans autre impact que l'annulation des transactions qui utilisaient ces données. Cela ne devrait pas être un problème, car les applications transactionnelles sont généralement écrites pour gérer les échecs de transaction.

Ce document est une présentation de la mise en place d'un cluster MySQL entre 4 machines tournant sur la distribution Ubuntu GNU/Linux.

Architecture



Le cluster MySQL que nous présentons est constitué de trois composants principaux :

- ➔ *Les nodes A et B : ce sont des machines ignorantes qui se contente de stocker les bases de données du cluster. Elles exécutent le daemon « ndbd » et sont directement reliées au serveur de management. Initialement, les nodes de stockages ne connaissent pas l'architecture du cluster. Ils connaissent juste l'adresse IP du Manager.*

  Les nodes du cluster se partagent les bases de données en suivant une règle simple : on peut avoir de 1 à 4 répliques des bases avec un nombre de cluster multiple du nombre de réplique (exemple : pour avoir 3 répliques, il faut au moins 3, 6, 9 ou 12 nodes, etc...);

- ➔ *Le serveur SQL : c'est le point d'accès au cluster pour un utilisateur lambda. Le serveur SQL héberge l'interpréteur de commandes et exécute les requêtes en prenant les données sur les nodes que le Manager lui communique. Il possède la même configuration que les nodes mais lance le daemon « mysqld » a la place de « ndbd »;*

- ➔ *Le manager : c'est lui qui supervise le cluster. Ses missions principales sont de répartir la charge lors des transferts de données et de maintenir a jour l'état du cluster.*

Le système d'exploitation utilisé est Ubuntu avec un noyau 2.6.15-27-386 et MySQL 5.0.24a. Le serveur de Management utilise le daemon NDB_MGMD, les nodes utilisent NDBDet les serveurs SQL utilisent le daemon MySQLD classique.

Les développeurs de MySQL préconisent l'utilisation d'un réseau 100Mbits au minimum afin de garantir un bon fonctionnement du cluster. Pour des raisons de structure, une partie du réseau de test est limitée par l'utilisation d'un Hub mais la plupart des noeuds et serveurs SQL sont reliés au réseau 100Mbits.

Configuration

1. Configuration du Manager

Le Manager du cluster, comme les nodes, n'utilise pas directement le daemon MySQLD. En fait, la suite de logiciels permettant la gestion de clusters MySQL est légèrement à part : ce sont les composants « ndb ».*

ndb_config	ndb_restore
ndb_delete_all	ndb_show_tables
ndb_drop_table	ndb_waiter
ndb_mgmd	ndbd
ndb_select_count	ndb_drop_index
ndb_test_platform	ndb_mgm
ndb_cpcd	ndb_select_all
ndb_desc	ndb_size
ndb_error_reporter	

Dans un premier temps, nous supprimons toute activité liées à MySQL sur la machine Manager (y compris les scripts de lancement dans /etc/rc2.d/) et nous créons un fichier de configuration `/root/mysql-cluster/config.ini` *:*

`root@Machine-gdr-14:~#cat /root/mysql-cluster/config.ini`

```
#Options affectant les processus ndbd sur tous les noeuds
[ndbd default]
NoOfReplicas=2          # Nombre de répliques
DataMemory=200M         # Mémoire à allouer pour le stockage des données
IndexMemory=50M         # Mémoire à allouer pour le stockage des index

#Définition du manager (lui-même)
[mgm]
HostName = 10.6.0.14

# On crée une entrée NDBD par node du cluster
[ndbd]
# Nom d'hôte ou adresse IP
hostname=10.6.0.142
# Dossier pour les fichiers de données du noeud
datadir=/home/julien/Cours/Master/Systeme/mysql-cluster-data
[ndbd]
hostname=10.6.0.145
datadir=/root/mysql
```

```
#Déclaration du serveur SQL
[mysqld]
hostname=10.6.0.13
```

Le fichier est relativement simple. Une petite subtilité est qu'il ne faut pas mettre de commentaire sur la même ligne que les champs entre crochets, car cela provoque un bug lors du lancement du Manager:

```
root@Machine-gdr-14:~# ndb_mgmd -f /root/mysql-cluster/config.ini
Error line 14: Parse error
Error line 14: Could not parse name-value pair in config file.
Unable to read config file
```

La commande ci-dessus, lorsqu'elle réussi, lance deux processus qui signifie que le Manager est opérationnel. On peut se connecter à l'interface d'administration via la commande « `ndb_mgm` » *et visualiser l'état du cluster :*

```
root@Machine-gdr-14:~# ndb_mgm
-- NDB Cluster -- Management Client --
ndb_mgm> show
Connected to Management Server at: localhost:1186
Cluster Configuration
---------------------
[ndbd(NDB)]     8 node(s)
id=2 (not connected, accepting connect from 10.6.0.142)
id=3    @10.6.0.145  (Version: 5.0.24, starting, Nodegroup: 0)

[ndb_mgmd(MGM)] 1 node(s)
id=1    @10.6.0.14  (Version: 5.0.24)

[mysqld(API)]   3 node(s)
id=4 (not connected, accepting connect from 10.6.0.13)


ndb_mgm>
```

On voit ici l'état des différents nodes : les nodes de stockage, dont un seul est connecté pour l'instant, la machine de management et l'interpréteur MySQL. Cette interface est indispensable a la surveillance du cluster. Elle permet de démarrer ou d'arrêter des nodes, de connaître leurs status, de créer un backup du cluster, etc...

2. Configuration des Nodes A & B

Les nodes sont ignorants ! C'est une règle de base du fonctionnement du cluster. Leur configuration est extrèmement basique :

Pour ajouter un node dans le cluster, on édite son fichier `/etc/mysql/my.cnf` *comme suis :*

`root@Machine-gdr-14:~#cat /etc/mysql/my.cnf`

```
[client]
port            = 3306
socket          = /var/run/mysqld/mysqld.sock
[mysqld_safe]
socket          = /var/run/mysqld/mysqld.sock
nice            = 0
[mysqld]
ndbcluster                   # démarrage en mode cluster
ndb-connectstring=10.6.0.14  # adresse du serveur de management

[... fichier tronqué ...}

[mysql_cluster]
ndb-connectstring=10.6.0.14  # adresse du serveur de management
```

Les trois lignes commentées sont ajoutées à la configuration au fichier my.cnf par défault. Elle permettent de lancer le daemon « ndbd » en stand alone (sans mysqld) pour que celui-ci se connecte au manager et intègre le cluster.

Malheureusement, le daemon « ndbd » est particulièrement instable et se coupe souvent. Pour pallier à ce problème, nous avons écris un script bash très simple qui vérifie toutes les 2 secondes si « ndbd » est lancé et le relance (voir colonne de droite).

```
#! /bin/sh
UID_ROOT=0
if [ "$UID" -ne "$UID_ROOT" ]
then
        echo "Vous devez etre ROOT pour lancer ce script"
        exit 0
fi
echo -n "Lancement initial de ndbd a "
date
/usr/sbin/ndbd &
while true ;
do
        NDBDPID=$(ps ax | grep ndbd | awk {'print $1'} | head -n 1)
        if [ -z $NDBDPID ]
        then
                echo -n "relance de ndbd a "
                date
                /usr/sbin/ndbd &
        fi
        sleep 2
done
```

Il faut également s'assurer que le répertoire « datadir » spécifié dans la configuration du manager pour un node existe bel et bien sur ce node.

Une fois ce travail effectué, il faut lancer « `ndbd --initial` *» pour qu'il réplique l'intégralité des fichiers des bases de données sur son système local. Les fois suivante, la commande « ndbd » suffit a lancer le node.*

Les nodes échangent en permanence des informations entre eux car ils ont besoin de tenir leurs bases de données à jour. Dans un cluster MySQL, la notion de Master/Slave est différente des système de réplication classique car ici c'est le manager qui décide quel noeud est Master pour les autres. Ainsi, si un noeud Master est indisponible, le manager en désigne un autre comme nouveau Master. cela permet d'éliminer les problèmes de rédémarrage du Master qui sont monnaie courante dans un système Master/Slave classique.

Voici un exemple de données que s'échangent les deux nodes de notre cluster :

```
$.....................

$.....................

$.....................

$.....................

................p. Hassan al-Bolkiah                                    ..^M.........BN..$..0<.
.............T....t.g.                                  %...................p.........MYS.$..d..
...................p.....MYS.^M...Malaysia
............Southeast                          As$..d..                              ...................p.ia
.........H.............jS........B.......G.......G..

.Malaysia             $..d.. ...................p.                                .......Constitutional
Monarchy, Federation            .......Sala$..T.. ...................p.huddin Abdul Aziz Shah Alhaj
..^M.........MY..

$..0<.                .............S...).x..                %...................p.........VCT.$..d..
...................p.....VCT.^M...Saint         Vincent          and          the          Grenadines
............Caribbean                                       $..d..               ...................p.
.........C.............P..........B.......C......

.Saint Vincent and the Gr$..d.. ...................p.enadines                  .......Constitutional
Monarchy                       .......Elisabet$..P.. .....

..............p.h II                                                          ..^M.........VC..$..0<.
.............T...+E...                                  %...................p.........LBN.$..d..
...................p.....LBN.^M...Lebanon
............Middle                  East                  $..d..                    ...................p.
........."F.............P.2........B.......F.....dlF..

 Mariana Islands                                  ............Micronesia  $..d.. ...................p.
.........C.........0.........B..............

.Northern Mariana Islands    $..d.. ...................p.                  .......Commonwealth of the
US                        .......George W. Bu$..L.. .........

..........p.sh                                                              ..^M.a.......MP..$..0<.
.............S....$7..                                  %...................p.........SOM.$..d..
...................p.....SOM.^M...Somalia
............Eastern                           Afri$..d..                            ...................p.ca
.........I.............h.........8B......iD......

.Soomaaliya              $..d.. ...................p.                     ...


...... ..................

$..... .....(t..............

$.....................

...... ......t..............

$..... ......t......
```

En pratique, les nodes maintiennent en permanence une connection TCP active entre eux (essentiellement des paquets PUSH / PUSH_ACK). La quantité de données qui transite est importante pour assurer une homogénéité des données sur les nodes, ces derniers pouvant être interrogés sans distinctions.

3. Configuration de l'interpréteur SQL

Ce dernier composant est tout aussi simple à configurer que les nodes mais certaines subtilitées méritent notre attention.

La configuration se fait également dans le fichier « my.cnf » avec l'ajout des trois mêmes champs que précédemment (ndbcluster et ndb-connectstring) plus la modification de la ligne « `bind-address` » pour accepter les connections sur l'interface réseau :

```
bind-address = 10.6.0.13
```

Ces modifications empêche le lancement de MySQL par les scripts fournis dans la distribution Ubuntu, il convient donc de les supprimer dans `/etc/rc2.d` pour éviter toute erreur au démarrage du système.

Le lancement du daemon MySQLD se fait alors directement via le binaire, ce dernier allant chercher par défaut la configuration dans /etc/mysql/my.cnf :

```
Machine-gdr-13:~# mysqld &
[1] 3899
070125 13:32:27  InnoDB: Started; log sequence number 0 43655
070125 13:32:28 [Note] mysqld: ready for connections.
Version: '5.0.30-Debian_3-log'  socket: '/var/run/mysqld/mysqld.sock'  port:
3306  Debian etch distribution
Machine-gdr-13:~#
```

note : le système utilisé ici n'est pas une Ubuntu mais une Debian Etch car nous pensions avoir des problèmes avec Ubuntu et MySQL alors que c'est nous qui faisions une erreur.

On peut alors accèder au système MySQL comme sur n'importe quel autre base de donnée, le fonctionnement en cluster ne modifiant pas l'utilisation de l'interpréteur.

Pour effectuer nos tests, nous avons utilisé la base de données « world » distribuée sur le site de MySQL (http://dev.mysql.com/doc/world-setup/en/world-setup.html). Pour utiliser cette base dans notre cluster, il faut modifier le fichier source world.sql pour remplacer le moteur (engine) MyISAM par NDBCLUSTER :

exemple :

```
CREATE TABLE `City` (
  `ID` int(11) NOT NULL auto_increment,
  `Name` char(35) NOT NULL default '',
  `CountryCode` char(3) NOT NULL default '',
  `District` char(20) NOT NULL default '',
  `Population` int(11) NOT NULL default '0',
  PRIMARY KEY  (`ID`)
) ENGINE=NDBCLUSTER DEFAULT CHARSET=latin1;
```

Un problème que nous avons rencontré est que la création de la base de données est impossible si la connection entre le serveur SQL et le manager est imparfaite (d'où le besoin de connection 100Mbits

fiables). Nous avons passé beaucoup de temps à comprendre pourquoi une erreur « `Create Table : table already exist` *» survenait lors de la tentative de la création de la base et ce n'est qu'après plusieurs hypothèse et une installation de Debian Etch que nous avons compris le lien avec le manager.*

Le serveur SQL doit apparaître comme suit dans l'outil « ndb_mgm » du manager :

```
Cluster Configuration
---------------------
[ndbd(NDB)]     2 node(s)
id=2    @10.6.0.142  (Version: 5.0.30, Nodegroup: 0)
id=3    @10.6.0.145  (Version: 5.0.24, Nodegroup: 0, Master)

[ndb_mgmd(MGM)] 1 node(s)
id=1    @10.6.0.14  (Version: 5.0.24)

[mysqld(API)]   1 node(s)
id=4    @10.6.0.13  (Version: 5.0.30)
```

Ainsi, nous pouvons créer la base de données qui est immédiatement répliquée sur les nodes du cluster en suivant les régles de gestion (nombre de copies, master/slave) spécifiées par le manager.

Tests

1. Montée en charge

Hormis le test basique de l'exécution d'une requête sur le cluster, la montée en charge est testée par un ensemble de requête inclus dans un script Perl avec récupération du temps d'exécution. Le script est le suivant :

```
#! /usr/bin/perl -w

use strict;

use Mysql;

use Time::HiRes;

#source de données

my $db = Mysql->connect("10.6.0.13", "world", "julien") or die "Echec de la
connexion\n$!";

my @req;

#cherche les villes de plus de 200000 habitants ou on parle francais

$req[0] = "select distinct City.Name, Country.Name, Language from City,
Country, CountryLanguage where LifeExpectancy >=75 and
Country.Code=CountryLanguage.CountryCode and City.CountryCode=Country.Code and
CountryLanguage.Language like 'French' and City.Population >= 200000;";

# cherche les villes américaines ou on parle francais

$req[1] = "select distinct City.Name from CountryLanguage, Country, City where
Country.Code = CountryLanguage.CountryCode and City.CountryCode = Country.Code
and Country.Name like 'United States' and CountryLanguage.Language like
'French';";

# cherche les villes du monde de plus de 2000000 d'habitants ou on parle
francais et anglais et ou l'esperance de vie est sup a 70 ans

$req[2] = "select City.Name, Country.Name from CountryLanguage, Country, City
where Country.Code = CountryLanguage.CountryCode and City.CountryCode =
Country.Code and Country.LifeExpectancy >= 80 and CountryLanguage.Language in
('French','English') and City.Population>=2000000;";

# cherche les villes du monde ayant le meme nombre d'habitants et parlant la
même langue

$req[3]="select CA.Name, CB.Name from City CA, City CB, CountryLanguage LA,
CountryLanguage LB where CA.Population=CB.Population and LA.Language =
LB.Language;";

my $initTime = [Time::HiRes::gettimeofday()];

foreach my $requete (@req){

      $db->query($requete);

}

my $elapsed = Time::HiRes::tv_interval($initTime);

print "temps d'exécution : $elapsed s\n";
```

L'exécution de ce script nous a permit de comprendre que la charge de calcul était centralisée sur l'interpréteur MySQL, alors que nous pensions initialement que les nodes se répartissaient également la charge de calcul.

De plus, une requête est lancée sur un interpréteur qui accède à un node particulier. Ce groupe n'est pas modifiée tant que la requête n'est pas complètement exécutée.

Si ce node particulier est déconnecté pendant la transaction, le cluster ne bascule pas la requête sur un autre node. L'utilisateur reçoit un message d'erreur et l'exécution est annulée. Toutefois, la requête peut être immédiatement relancée sur l'interpréteur et prendra un autre node pour créer la transaction.

On peut donc résumer : TRANSACTION = MANAGER (REQUETE + INTERPRETEUR + NODE)

2. Réplication

La seconde phase de test que nous avons réalisée ici consiste a vérifier que les données sont bien réparties sur tous les nodes et que l'absence d'un node qui vient d'enregistrer une modification ne pose pas de problème au niveau de l'intégrité des données. En clair :

1. *Débrancher le node A (B est donc le Master)*

```
[ndbd(NDB)]     2 node(s)
id=2    @10.6.0.142  (Version: 5.0.30, starting, Nodegroup: 0)
id=3 (not connected, accepting connect from 10.6.0.145)
```

2. *Exécuter une requête « update » sur la base de donnée « world »*

```
mysql> update City set Population=2851956 where Name like 'Paris';
```

3. *Rebrancher le node A*

```
[ndbd(NDB)]     2 node(s)
id=2    @10.6.0.142  (Version: 5.0.30, Nodegroup: 0)
id=3    @10.6.0.145  (Version: 5.0.24, Nodegroup: 0, Master)
```

4. *Débrancher le node B (A est donc le Master)*

```
[ndbd(NDB)]     2 node(s)
id=2    @10.6.0.142  (Version: 5.0.30, Nodegroup: 0, Master)
id=3 (not connected, accepting connect from 10.6.0.145)
```

5. *Exécuter une requête « select » sur le champ précédemment mis à jour*

```
mysql> select Population from City where Name like 'Paris';
```

6. *Vérifier que la valeur est bien modifiée sur le node A*

```
+------------+
| Population |
+------------+
|    2851956 |
+------------+
1 row in set (0.04 sec)
```

Conclusion

La mise en place d'un cluster MySQL n'est pas d'un niveau de complexité très important tant que l'on a bien compris les concepts sous-jacents... et c'est là que la difficulté s'accroît car la documentation officielle (en Français comme en Anglais) explique mal le fonctionnement général du cluster. Il faut donc effectuer des suppositions et les vérifier alors même que la configuration n'est pas valide.

Une amélioration de ce travail serait de balancer l'interpréteur MySQL entre plusieurs machines de façon transparente pour l'utilisateur. La configuration de base du cluster permet l'utilisation de plusieurs interpréteur mais via des adresses IP différentes.

De plus, ce document ne couvre pas la disponibilité du manager, clé de voute du cluster et qui n'est pas sécurisé. Toutefois, la probabilité d'indisponibilité est réduite car cet équipement n'est que très peu chargé par les transactions du cluster.

Enfin, et pour finir, il pourrait être intéressant de regarder du coté de la répartition des calculs, comme le permet le « Grid Computing » d'Oracle, afin d'alléger les interpréteurs SQL.

Toutefois, les travaux réalisés sur le système Cluster de MySQL nous permettent d'affirmer que celui-ci est mûr pour une utilisation industrielle sur des bases de données de faible importance. Pour les autres, il vaut mieux toujours regarder du coté d'Oracle.