MySQL Replication

La funzionalita' di replica dati di MySQL e' semplice da configurare, non richiede praticamente manutenzione, e' molto flessibile e non aggiunge carico al database Master. E' percio' comprensibile che sia molto utilizzata. Gli impieghi sono diversi: per disporre di copie di database in HA (High Availability: alta affidabilita'), per creare una batteria di database per siti web molto acceduti, per semplificare modalita' e procedure di backup, ...

Nel seguito sono riportate alcune informazioni di interesse organizzate in paragrafi specifici: Introduzione, Configurazione, Architettura, Amministrazione (FAQ), Nuove funzionalita' (Stored Routine (5.0), Row Based Replication (5.1), Semi-synchronous (5.5), Crash safe, delayed, ... (5.6), CHANGE MASTER without stopping (5.7)), Avvertenze per l'uso, Point in Time Recovery.

Introduzione

La replicazione su MySQL e' statement based ed asincrona.

Il Master si occupa di registrare su file (bin-log file) tutti gli statement che vengono eseguiti sulla base dati e che operano una qualche modifica ai dati (DML e DDL).
Lo Slave si collega con un thread al Master, raccoglie il contenuto dei bin-log, lo trasferisce in locale e quindi si occupa di applicarlo alla base dati.
La parte svolta dal Master e' molto semplice, poco onerosa e "stupida". In pratica ospita una sessione client per ogni Slave configurato. La maggioranza delle opzioni di configurazione e la gestione ricadono sullo Slave. E' questo che si occupa della ricezione dei log, del loro allienamento e della corretta applicazione degli statement SQL sulla base dati.

Sono possibili differenti configurazioni. Un Master puo' servire piu' Slave. Uno Slave puo' essere a sua volta Master di altri Slave. Possono essere esclusi/inclusi dalla replicazione database, tabelle, ... Il Master e gli Slave possono essere differenti praticamente in tutto: Storage Engine, parametri di configurazione/tuning, versioni, struttura, ... insomma: continuate a leggere!

Configurazione

La configurazione della replication e' piuttosto semplice anche se le possibilita' sono molteplici. La documentazione completa e' presente nel sito ufficiale, quindi nel seguito riportiamo solo un esempio di configurazione molto semplice... ma che copre tutti i punti fondamentali!

Attivita' sul **Master**	Attivita' sullo **Slave**	**Note**
[mysqld] server-id=10 log-bin=mysql-bin		La configurazione del Master e' semplice: basta definire un server-id univoco ed indicare il nome dei file di log binario. Se l'utilizzo del database e' significativo e' opportuno porre il binlog su un file system separato per evitare un file system full. Altri parametri utili sono riportati nel seguito.
	[mysqld] server-id=20 read_only=1	Anche la configurazione dello Slave e' semplice: server-id univoco e, se non si vuole che vengano modificati i dati sulla replica, parametro read_only. Altri parametri utili sono riportati nel seguito.
grant replication slave on *.* to 'rep'@'%' identified by 'xyz';		Non e' obbligatorio definire un nuovo utente ma e' consigliato sia per sicurezza che per semplicita' di configurazione e controllo
Backup	Restore	Le modalita' backup/restore sono molteplici e dipendono da molti fattori. Nel caso piu' semplice le basi dati sono vuote e non c'e' nulla da fare! Se non avvengono modifiche sul Master un semplice mysqldump e' sufficiente... Altrimenti e' importante fare in modo che sia garantito l'allineamento delle basi dati (eg. **FLUSH TABLES WITH READ LOCK**; e backup oppure **mysqldump --single-transaction --all-databases --master-data**)
show master status;		Con questo comando si ottiene lo stato della replicazione. In particolare sono necessari il nome del file ed il progressivo che servono per configurare la replicazione sullo Slave. Il comando deve essere lanciato prima di modificare i dati sul Master (**UNLOCK TABLES**).
	change master to master_host='myMaster.MyDomain.it', master_user='rep', master_port=3306, master_password='xyz', master_log_file='*mysql-bin.000001*', master_log_pos=*69*;	Con questo comando lo Slave sa come connettersi al Master. Vengono creati i file **master.info** e **relay-log.info**. I parametri in italico sono quelli ottenuti con il comando show master status.
	start slave;	Per far partire i thread dello Slave. Sono utilizzati un thread remoto (sul Master per inviare il bin-log) e due thread locali (per ricevere il bin-log ed applicarlo).

	show slave status\G	Gia' fatto! Non c'e' altro da configurare, con questo comando si controlla lo stato della replicazione.

I passi riportati e le spiegazioni sono un po' semplificati ma... funzionano e rendono l'idea (spero ;-) E' ovvio che per far leggere i nuovi parametri del my.cnf e' necessario riavviare il server, che per lanciare uno statement SQL e' necessario collegarsi alla base dati...
E' inoltre fondamentale controllare l'utilizzo di spazio da parte dei binlog: possono crescere molto piu' in fretta della base dati. A seconda delle configurazioni utilizzate dovra' essere impostata una pulizia periodica dei binlog obsoleti.

I database da replicare (di default vengono tutti replicati) si indicano con il parametro **binlog-do-db** sul Master o il parametro **replicate-do-db** sullo Slave. I database da non replicare si indicano con il parametro **binlog-ignore-db** sul Master o il parametro **replicate-ignore-db** sullo Slave. Per indicare singole tabelle si utilizzano i parametri **replicate-do-table** e **binlog-do-table** che possono utilizzare wildcard.

I parametri di configurazione utilizzabili, come abbiamo visto, sono molteplici... Per riassumere riportiamo una configurazione adatta ad una semplice architettura in cui sono presenti un master ed uno slave configurati in replicazione per HA:

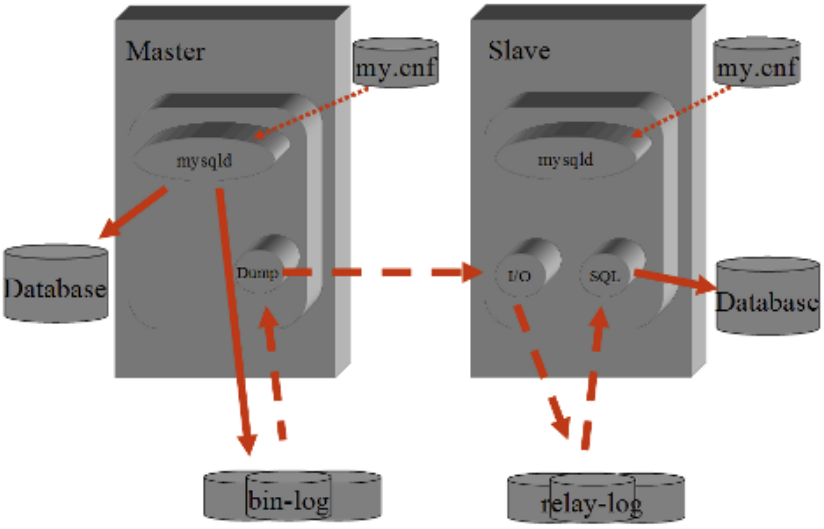
Parametri **Master**	Parametri **Slave**
[mysqld] server-id=10 log-bin=mysql-bin binlog-ignore-db=test	[mysqld] server-id=20 log-bin=mysql-bin binlog-ignore-db=test *read_only=1*

Per essere sicuri che sullo slave non avvengano accessi e' stato utilizzato il parametro read_only=1 che bisogna aver cura di rimuovere quando lo Slave diventa Master (evento chiamato promote).

Se le autorizzazioni sono differenti (eg. su uno slave utilizzato solo per backup) e' possibile evitare gli errori sugli utenti mancanti con **slave_skip_errors=1396**. Se i server sono ospitati su un cluster active-passive o comunque vi e' la possibilita' che l'hostname venga cambiato e' opportuno fissare i nomi del file di relay e del relativo indice con i parametri **relay-log=relay-bin relay-log-index=relay-bin.index**.
Nelle configurazioni su basi dati modificate in modo pesante e' necessario un opportuno dimesionamento dei file systems eventualmente dedicando un file system ai binlog. Utili sono anche i parametri **max_binlog_size** per definire la dimensione massima dei binlog sul Master, **max-relay-log-size** e **relay-log-space-limit** per limitare rispettivamente la dimensione dei relay-log e lo spazio totale occupato sullo Slave.

Architettura

La replicazione utilizza alcuni processi e file di appoggio. Il processo mysqld sul Master salva sul file **bin-log** gli statement di DML e DDL che sono eseguiti sulla base dati. Sul file **bin-log.index** e' mantenuto l'elenco dei bin-log attivi sul master.
Per mantenere allineati i dati lo Slave utilizza piu' thread. Il primo e' una connessione remota al Master ed ha il compito di raccogliere i dati dal bin-log (BinLog Dump) ed e' sempre attivo. Gli altri thread sono locali ed hanno il compito di ricevere il contenuto del bin-log (Slave I/O) sui relay-log e di applicarlo alla base dati (Slave SQL). In caso d'errore nell'inserimento dei dati il thread Slave SQL si interrompe mentre lo Slave I/O continua a raccogliere i dati dal Master. Con **show slave status\G** si ottiene l'indicazione dell'errore occorso; una volta corretto il problema la replicazione riprende dal punto in cui si era interrotta applicando il relay-log.



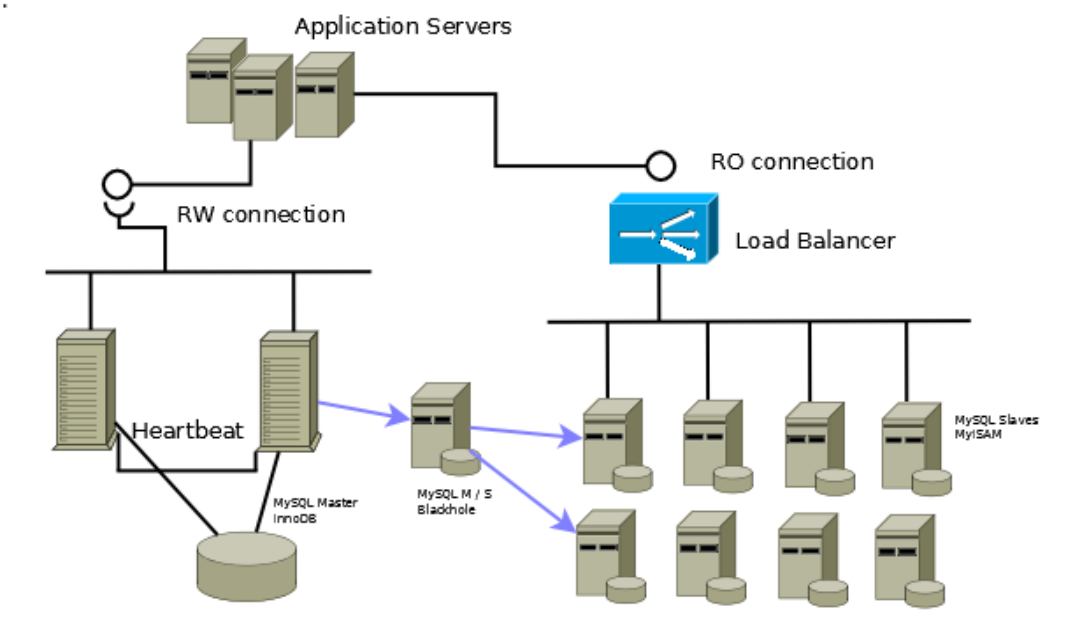
La replicazione in MySQL e' molto flessibile e sono possibili diverse alternative.
La replicazione puo' essere eseguita solo su alcuni database o su specifiche tabelle. Quali database replicare si indicano con il parametro **replicate-do-db** (*utilizza* la USE) sullo Slave o con il parametro **binlog-do-db** sul Master. Per replicare singole tabelle si utilizza il parametro **replicate-do-table** (che puo' utilizzare wildcard). E' anche possibile indicare quali DB o TABLE non replicare o applicare con parametri analoghi in cui il **do** e' sostituito con **ignore** (eg. **replicate-ignore-table**).

Molto utili sono anche: **replicate-ignore-db=mysql**, **binlog-ignore-db=test** e **slave_skip_errors=1396** per non replicare gli accessi al database delle autorizzazioni o di test e proseguire nella replicazione anche in caso di errori sulle utenze. Il nome dei relay log sullo Slave e' basato sull'hostname, se l'hostname e' soggetto a cambiamenti (eg. failover cluster) la replicazione si blocca; e' possibile impostare il nome dei relay log con i parametri **relay-log** e **relay-log-index**.
E' semplice comprendere in quale modo agiscono i parametri considerando l'architettura utilizzata da MySQL. Con i parametri binlog-* si governa quali comandi SQL vengono scritti o meno sul file bin-log da parte del Master. Con i parametri replicate-* si indica quali comandi SQL vengono applicati dallo Slave.

Gli Engine utilizzati sullo Slave possono essere diversi da quelli utilizzati sul Master. Questo consente di scegliere l'Engine piu' adatto allo scopo (eg InnoDB sul Master in cui avvengono le transazioni, MyISAM sullo slave su cui vengono effettuate selezioni complesse o backup).

Nel caso piu' semplice un Master viene replicato su uno Slave. Se gli accessi in lettura al database sono molto elevati (eg. siti web molto acceduti) e' possibile configurare piu' Slave server. Per replicare un Master su piu' Slave non e' necessario alcun cambiamento alla procedura indicata all'inizio: basta applicare i comandi previsti per lo Slave su piu' server!
Uno Slave puo' comportarsi a sua volta da Master e questo puo' essere utilizzato in cascata piu' volte. In particolare, se il numero di Slave che si vuole mantenere allineati e' molto elevato, si utilizza la catena **A -> B => C^{n}** dove n e' puo' essere grande a piacere. A e B vanno configurati come Master, su B deve essere definito il parametro **log-slave-updates**, B e C^{n} vanno configurati come Slave di A e B rispettivamente. Il server B puo' essere utilizzato in lettura dalle applicazioni oppure puo' essere dedicato alla replicazione. In questo ultimo caso si utilizza tipicamente l'Engine **Blackhole** per non utilizzare spazio in locale e non avere l'overhead del caricamento dei dati.
In questa configurazione tutte le modifiche vengono applicate sulla base dati A. I log vengono trasferiti sul sistema B con il minimo impatto su A poiche' vi e' un solo server ad accedere ai dati. Poiche' B utilizza l'Engine Blackhole non vi e' alcun rallentamento ed il sistema e' dedicato ad ospitare i demoni dei diversi server C utilizzati in lettura. Una descrizione piu' completa di questa configurazione si trova in [questo documento](#).

Come abbiamo visto utilizzando la replicazione e' possibile costruire complesse architetture di database MySQL che soddisfano ai piu' stringenti requisiti di alta affidabilita', continuita' di servizio, prestazioni, trasparenza applicativa, scalabilita', ... Riporto la figura che segue che spero abbastanza chiara:



FAQ Amministrazione

La replicazione su MySQL richiede un impegno molto limitato da parte del DBA. Ma qualcosa bisogna comunque fare...

Cosa si deve controllare quando e' attiva la replicazione e come e' possibile agire quando si verificano dei problemi? In questo capitolo vengono riportate le indicazioni relative ai casi piu' comuni.

- **Come si controlla il Master?**
  - Con il comando **SHOW MASTER STATUS;**
    Se il master e' attivo viene restituita una tabella con una riga contenente gli estremi della posizione attuale sul binlog.

    Sara' banale ma in realta' capita spesso: il file system non deve arrivare al 100%! Va sempre controllato e se necessario si deve prevedere un FS separato per i binlog.

- **Come si controllano gli Slave?**
  - Con il comando **SHOW SLAVE STATUS\G**
    Il comando riporta molteplici informazioni... Ma per far funzionare la replicazione l'unica cosa che conta e' che siano attivi i due processi di slave:

```
...
Slave_IO_Running: Yes
Slave_SQL_Running: Yes
...
```

    Utile e' spesso **Seconds_Behind_Master** che riporta il ritardo dello slave nell'applicare i log rispetto al master.
    Con il comando **SHOW PROCESSLIST;** viene riportata la lista dei processi attivi. Debbono essere attivi i thread slave di I/O ed SQL. In situazione di "riposo" debbono avere come stato rispettivamente: *Waiting for master to send event* e *Has read all relay log; ...*

```
+----+-------------+-----------+------+---------+--------+-------------------
| Id | User        | Host      | db   | Command | Time   | State
+----+-------------+-----------+------+---------+--------+-------------------
|  2 | system user |           | NULL | Connect | 696969 | Waiting for maste
|  3 | system user |           | NULL | Connect | 696969 | Has read all rela
...
```

    Mi ripeto... Sara' banale ma in realta' capita spesso: il file system non deve arrivare al 100%!

- **Il processo Slave_SQL non e' attivo, che faccio?**
  - Naturalmente si puo' far ripartire tutto con **START SLAVE**. Ma se muore nuovamente allora bisogna capire la ragione dell'errore controllando il log di MySQL. Eseguita la diagnosi vi sono diverse possibilita' (dalla migliore alla peggiore):
    - Risolvere la causa dell'errore e far riparire la replicazione con **START SLAVE**
    - Se l'errore e' dovuto ad una condizione temporanea e si vuole saltare l'istruzione che va in errore si utilizzano i comandi:

```
SET GLOBAL SQL_SLAVE_SKIP_COUNTER = 1;
START SLAVE;
```

    - Se l'errore e' dovuto ad una condizione nota (eg. sullo slave non e' volutamente presente un DB oppure un utente) e' possibile far proseguire la replicazione anche in caso d'errore impostando il parametro **slave_skip_errors=XXX,YYY,ZZZ**

- **Il processo Slave_IO non e' attivo, che faccio?**
  - Naturalmente si puo' far ripartire tutto con **START SLAVE**. Ma se muore nuovamente allora bisogna capire la ragione dell'errore controllando il log di MySQL. Si tratta quasi sicuramente di un problema di connessione verso il server master che va risolta e quindi si puo' riavvare con **START SLAVE**. In qualche caso puo' essersi persa la sincronizzazione con i log del master... per ripartire in modo pulito si utilizza il comando **RESET SLAVE**.

- **E se muore (temporaneamente) il master?**
  - Se il server che contiene il master cade per una qualsiasi ragione basta riavviarlo. Il DB master riprendera' a lavorare e gli slave a sincronizzarsi. I comandi di controllo sono gia' stati riportati...

- **E se muore (temporaneamente) lo slave?**
  - Nulla di piu' semplice: basta riavviarlo! Alla ripartenza recuperera' il tempo perso rispetto al master.

    Nel caso in cui passi troppo tempo potrebbero non essere piu' disponibili tutti i bin-log necessari alla sicronizzazione ed e' necessario un ripristino. Tra le diverse modalita' possibili questa e' una delle piu' dirette:

```
STOP SLAVE

mysqldump --single-transaction --all-databases --master-data   # Ovvi

Restore # basta un mysql < backup.out

START SLAVE
```

    Terminati i passi indicati lo slave recuperera' quanto avvenuto nel frattempo sul Master (si controlla verificando il valore di Seconds behind master di SHOW SLAVE STATUS \G)

- **E se muore il master?**
  - Non c'e' problema... morto un master se ne fa un altro!
    Nel caso in cui il master non sia piu' disponibile (eg. fault HW) e si voglia rendere master uno degli slave si utilizza una procedura di promote. Naturalmente vi possono essere molte variazioni sul tema... Nel caso di questa configurazione i comandi sono:
    - Interrompere il thread slave di IO con **STOP SLAVE IO_THREAD**
    - Attendere che il thread slave SQL abbia applicato tutte le modifiche (lo stato del processo deve essere **Has read all relay log...**)
    - Interrompere tutti i processi dello slave che diventera' master con **STOP SLAVE**
    - Attivare il *nuovo* master con **RESET MASTER**
    - Se era attivo il parametro **read_only** questo puo' essere disattivato con **SET GLOBAL read_only = FALSE;**
    - Eventuali altri slave (o il vecchio master quando rinasce) possono essere configurati con il comando di **CHANGE MASTER TO** (i parametri master_log_file e master_log_pos possono essere omessi poiche' hanno valori di default)
    - Eventuali altri slave vengono fatti partire con **START SLAVE**

- **Come si resettano i bin-log file?**
  - Con il comando **FLUSH [BINARY] LOGS;** o con # **mysqladmin flush-logs** viene effettuato un flush su disco, il log corrente viene chiuso e viene aperto il successivo.
    Per chi e' abituato su altri DB... e' l'analogo di ALTER SYSTEM SWITCH LOG FILE

- **Come si cancellano i bin-log file?**
  - Con il comando **SHOW BINARY LOGS;** vengono riportati i bin-log attivi sul Master. Prima di cancellarli e' necessario controllare che siano gia' stati caricati su tutti gli Slave!

    Per cancellarli si utilizza il comando **PURGE MASTER LOGS TO 'mysql-bin.000069';** che rimuove i binary log precedenti a quello indicato.
    La pulizia puo' anche essere schedulata a tempo effettuando un # **mysqladmin flush-logs** e **PURGE MASTER LOGS BEFORE DATE_SUB(now(), INTERVAL 15 DAY);**.
    Sicuramente meno pulito, ma funziona lo stesso, cancellare da sistema operativo i file e poi correggere il bin-log.index che e' un normale file di testo...
    Qualunque comando si utilizzi... e' necessario controllare che i bin-log siano gia' stati caricati su tutti gli Slave!

Nuove funzionalita'

La replicazione su MySQL e' disponibile dalla versione 3.23.15 (2000) e si e' dimostrata da subito semplice da configurare/amministrare, veloce e di basso impatto sulle prestazioni.
Le diverse nuove funzionalita' inserite nel tempo a livello di database hanno richesto, in qualche caso, un aggiornamento della parte di replicazione. Gli aggiornamenti piu' importanti sono riportati nei capitoli seguenti.

Stored Routines (5.0)

Per le Stored Routines (disponibili dalla 5.0) un'importante considerazione e' relativa all'utilizzo del binary-log necessario per la replication, ma anche per il recovery point-in-time. Per rendere replicabili le azioni svolte dalle stored routine e' utilizzata una sintassi specifica per indicare le modifiche svolte sui dati. Infatti sul binary log viene riportata la stored routine richiamata ed il risultato di questa deve essere replicabile per poter riapplicare il log. Per tale ragione e' importante dichiarare se il comportamento della stored routine e' deterministico (ovvero si ripete sempre uguale a fronte degli stessi dati) e se vengono modificati dati.

```
CREATE PROCEDURE test1
...
DETERMINISTIC
MODIFIES SQL DATA
...
BEGIN
...
```

Poiche' il thread di replicazione gira in stato privilegiato sugli Slave e' richiesto il privilegio **SUPER** oppure l'impostazione del parametro **--log-bin-trust-routine-creators**.

Row based replication (5.1)

La replicazione su MySQL e' storicamente Statement Based. Questo la rende particolarmente efficiente ma, in qualche caso, vi sono dei limiti. Ad esempio nell'utilizzo di trigger e stored routine, con statement SQL che utilizzano valori random, con alcuni comandi quali LOAD_FILE(), SYSDATE, con le UDF (user defined functions), ... Per questo e' stata introdotta la possibilita' di replicare i dati anche con un meccanismo Row Based.
Dalla versione 5.1 (5.1.8 per essere precisi) e' possibile utilizzare tre differenti metodi per registrare sul binary log (binlog_mode):

- **STATEMENT**: e' la modalita' classica (dalla 3.23) in cui vengono riportati sul log tutti gli statement SQL eseguiti sul Master.
- **ROW**: in questa modalita' vengono riportate le tabelle e le righe modificate da ogni statement.
- **MIXED**: con questa modalita', che e' quella di default, la maggioranza degli statement vengono riportati nel log come statement ma viene automaticamente utilizzato il row logging nelle condizioni in cui la replicazione basata sugli statement potrebbe presentare problemi (eg. quando si utilizza la funzione UUID(), con le UDF, utilizzando l'Engine NDB, ...)

La modalita' di logging e' configurata nel file **my.cnf** ma, dalla release 5.1, puo' essere modificata dinamicamente a livello di server o di singola sessione. Analogamente avviene per molti altri parametri globali... tra questi anche l'impostazione **read_only** spesso utilizzata sugli slave.

Semi-synchronous replication (5.5)

La replicazione su MySQL e' asincrona, e' quindi possibile che, per causa di un fault, le ultime transazioni eseguite sul Master non siano ancora state replicate sugli Slave. La conseguenza e' un'effettiva perdita di dati fino all'ultima transazione trasmessa agli Slave.

Dalla versione 5.5 e' possibile impostare la replicazione semi-sincrona nella quale il Master, prima di effettuare il commit locale, attende che almeno uno slave abbia ricevuto i dati della transazione. In questo modo almeno uno Slave ha ricevuto l'ultima transazione committata e quindi non si possono verificare perdite di dati.
Sebbene si tratti di un protocollo piu' semplice e veloce rispetto al Two Phase Commit, le transazioni sono comunque rallentate. Per impostare la replicazione semi-sincrona e' quindi importante che la latenza nella comunicazione tra Master e Slave sia molto bassa.
La replicazione semi-sincrona e' implementata mediante due plug-in (uno per il master ed il secondo per gli

slave). Nel caso in cui nessuno Slave abbia il plug-in attivo o quando si verificano problemi, il master ritorna nella modalita' asincrona di replicazione senza bloccare o ritardare ulteriormente le transazioni.

Crash safe, delayed replica, table logging, ... (5.6)

La versione 5.6 di MySQL [N.d.E in produzione da Febbraio 2013] contiene diverse utili nuove funzionalita' per la replicazione dei dati:

- **crash safe**: maggior controllo sulle scritture sincrone dei bin-log ora affidabili anche in caso di caduta del server
- **delayed replica**: puo' essere impostato un ritardo nell'esecuzione delle repliche sugli slave
- **logging connessioni su tabelle**: puo' essere impostato il logging su tabella delle connessioni degli slave. Questo permette di effettuare controlli e statistiche sullo stato del Master e degli Slave con query SQL
- **row image control**: per effettuare il log solo delle colonne modificate (utile con dati blob)
- **MASTER_BIND**: puo' essere impostato un nuovo parametro per indicare qualche scheda di rete utilizzare nella connessione al master
- molto altro: multithreaded slaves, checksums, server UUIDs, ...

CHANGE MASTER (5.7)

La componente replication di MySQL e' molto utilizzata e continua ad essere arricchita con nuove funzionalita'.
La versione 5.7 di MySQL, non ancora disponibile in produzione [1Q14], gia' prevede alcune novita' per la replication.

Dalla versione 5.7 sara' infatti possibile effettuare un comando di CHANGE MASTER senza avere prima eseguito il comando di STOP SLAVE.
Nelle configurazioni multi-slave in cloud questa funzionalita' semplifica la gestione nel caso di fault e sostituzione del Master (promote).

Avvertenze per l'uso

Anche se molto potente, la replicazione di MySQL ha alcuni limiti che non consentono di replicare sempre tutto... come nei bugiardini delle medicine i rischi si trovano scritti in piccolo nelle avvertenze per l'uso!

IMPORTANTE: non fate scoppiare il file system del bin log!
Se le modifiche su una base dati sono molto significative e' opportuno utilizzare un file system separato per i bin-log, in caso d'errore almeno il DB continua a funzionare...

Alcuni comandi, per le funzionalita' particolari che implementano, non vengono sempre replicati correttamente. Ad esempio: LOAD_FILE() UUID() USER() FOUND_ROWS() SYSDATE() GET_LOCK() ...
L'utilizzo di Engine differenti all'interno della stessa transazione puo' generare problemi con la replicazione... ed in effetti sarebbe da evitare in ogni caso!
Alcune applicazioni sono piu' replicabili di altre: dipende dall'uso dell'SQL. Se la vostra applicazione utilizza Stored Routine, con la gestione dei SIGNAL, scatena un trigger, che agisce su una tabella in autoincrement, con tabelle partizionate, ... e' meglio controllarla con attenzione!
E' possibile utilizzare versioni differenti, Engine differenti e (non ricordo da quando ma si puo') strutture dati differenti tra Master e Slave. Ma pensateci bene prima di crearvi dei problemi da soli!
Vi sono differenze significative tra le versioni di MySQL sia nelle funzionalita' (eg. ROW BASED) che nell'elenco di Bug presenti e risolti. Meglio convergere su una versione recente e consolidata.

Point-in-Time Recovery

Quanto visto fino ad ora sui bin-log ad uso della replication puo' essere applicato con uno scopo differente: il Point-in-time Recovery (PITR). Il PITR consente di ripristinare un database fino al momento desiderato (eg. appena prima di una DROP TABLE lanciata per errore).
Il database va semplicemente configurato con il bin-log attivo (parametro **log-bin** nel file **my.cnf**). Per il restore l'idea di base e' quella di partire da un salvataggio completo ed affidabile, su cui vengono applicati i comandi contenuti nei file di bin-log fino al momento desiderato. I bin-log sono in formato binario, il comando **mysqlbinlog** trasforma il contenuto binario nelle istruzioni SQL corrispondenti. Per ottenere le variazioni occorse il comando e':

```
mysqlbinlog binlog.000069 binlog.000070 binlog.000071 > to_apply.sql
```

Potrebbe anche essere utilizzato in pipe con mysql... ma un minimo di controllo da parte del DBA di solito e' opportuno prima di lanciare i comandi SQL sul database.

Per indicare il momento fino a cui arrivare o da cui partire si utilizzano i parametri **--start-datetime --stop-datetime**. La data va indicata con formato std ISO: **--stop-datetime="2012-04-01 10:00:00"**. E' anche possibile indicare la transazione precisa da cui partire o arrivare con i parametri **--start-position --stop-position**.
In caso di ripristino di un DB lo spazio disponibile e' spesso un problema... Nella conversione da binario ad SQL la dimensione indicativamente aumenta del 50%.

La configurazione ed i comandi di amministrazione per la gestione di un DB su cui si utilizza il binlog per il PITR sono gli stessi gia' descritti per la gestione del Master...
Il solo parametro **log-bin** nel file **my.cnf** e' sufficiente per attivare il salvataggio dei binlog. Il parametro non e' dinamico e quindi e' necessario il riavvio del server. Altro parametro utile, introdotto dalla 5.1, e' **binlog_format=mixed** che permette di utilizzare la modalita' piu' conveniente di logging tra STATEMENT e ROW.
Nel caso del PITR non serve attendere che tutti gli Slave siano sincronizzati (non ci sono Slave) per il purge dei binlog ed e' sufficiente arrivare all'ultimo full backup affidabile. Disponendo di un backup ogni sera il comando per effettuare la cancellazione dei binlog e':
PURGE MASTER LOGS BEFORE DATE_SUB(now(), INTERVAL 2 day);

---

*Titolo: **MySQL Replication***
Livello: Avanzato
Data: 1 Aprile 2008
Versione: 1.0.17 - 14 Febbraio 2014
Autore: mail [AT] meo.bogliolo.name