



PERCONA
Performance Consulting Experts

Diagnosing Failures in MySQL Replication

O'Reilly MySQL Conference
Santa Clara, CA

Devananda "Deva" van der Veen

Introduction

- About Me
 - Sr Consultant at Percona since summer 2009
 - Working with large MySQL replication clusters since 2004
 - Currently living in NW Washington, previously in SoCal
- About Percona
 - Support, Consulting, Training, Development
 - We create & maintain Percona-Server w/ XtraDB and XtraBackup
 - Many other tools too: maatkit, aspersa, etc ...

Table of Contents

- Introduce the Problems
- Replication Architecture Review
Masters, Slaves, and the many files they keep
- Different ways things break
- How to diagnose the problems
- Knowing when to skip it, when (and how) to fix it, and when to just rebuild it.
- Questions

Replication is Ubiquitous

Who *doesn't* use MySQL Replication?

Everyone else:

What do you use replication for?

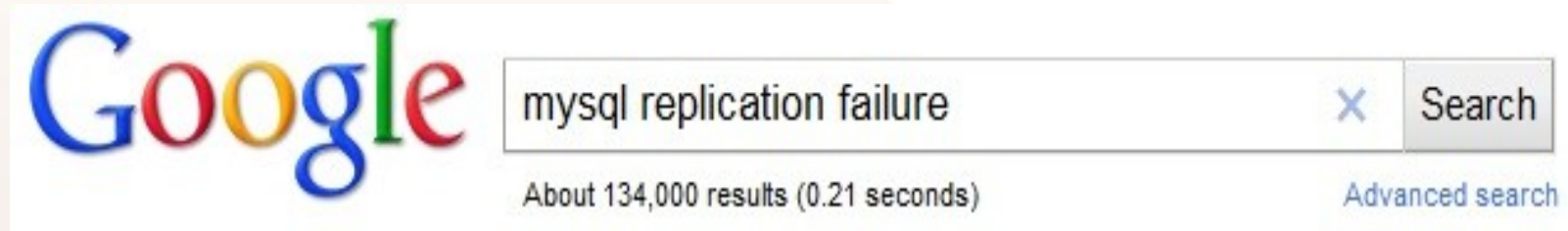
Replication is Ubiquitous

Who *doesn't* use MySQL Replication for...

- Load distribution
- Redundancy (local or geographic)
- Master-Master fail-over
- Taking backups from slave
- Archiving & Data Warehousing
- Many other things too

Ubiquitous... but not perfect

Google “mysql replication failure” → 130,000+ results



Ubiquitous... but not perfect

MySQL lists 37 Active bugs in Replication as of March 28, 2011

26945	Explicit TEMP TABLE causes replication failure if mysqld restarted
48062	Date arithmetic in Stored Procedure breaks replication
43457	replicate-ignore-db behaves differently with different binlog formats
45677 53079	Slave stops with Duplicate entry for key PRIMARY when using trigger

<http://bugs.mysql.com/search.php?cmd=display&status=Active&tags=replication>

Most problems have a Solution

Here are some specific problems that I'll discuss

- Duplicate key error on slave
- Query caused different errors on master and slave
- Temp table does not exist after slave restart
- Binary and relay log corruption
- Slave reads from wrong position after crash
- And more...

But first ...

We need to know a little about how MySQL replication works

- Master and Slave threads
- Master files
- Slave files

If you want to know more, see Lars Thalmann's presentations

<http://forge.mysql.com/w/images/5/5c/ReplicationArchitecture.pdf>

[http://assets.en.oreilly.com/1/event/2/MySQL Replication Tutorial Presentation 2.pdf](http://assets.en.oreilly.com/1/event/2/MySQL%20Replication%20Tutorial%20Presentation%202.pdf)

A Brief Review

Master Host		
Write to	<code>master-bin.xxx</code>	(binary file)
Keeps list at	<code>master-bin.index</code>	(plain-text file)

Slave Host			
Composed of two threads: IO + SQL			
IO_Thread		SQL_Thread	
Status file	<code>master.info</code>	Status file	<code>relay.info</code>
Reads from	<code>master-bin.xxx</code>	Reads from	<code>relay-bin.xxx</code>
Writes to	<code>relay-bin.xxx</code>		
Local list	<code>relay-bin.index</code>		

See <http://dev.mysql.com/doc/refman/5.1/en/slave-logs-status.html>

The Index Files

```
# cat master-bin.index  
./master-bin.000001  
./master-bin.000002
```

Master binary log index file

```
# cat relay-bin.index  
./relay-bin.000001  
./relay-bin.000002
```

Slave relay log index file

You sometimes need to edit these files manually
(but only in dire situations)

The Info Files

```
Slave:$ cat master.info
```

```
15  
master-bin.000001  
1818  
127.0.0.1  
repl_user  
repl_pass ←(plain-text!!)  
19501  
60  
0  
...
```

```
# of lines in file  
Master_Log_File  
Read_Master_Log_Pos  
Master_Host  
Master_User  
Master_Password  
Master_Port  
Connect_Retry  
Master_SSL_Allowed  
...
```

current
position
of io_thd
on master

The Info Files (2)

```
Slave:$ cat relay-log.info
```

```
./relay-bin.000002  
1392  
master-bin.000001  
1818
```

```
Relay_Log_File  
Relay_Log_Pos  
Relay_Master_Log_File  
Exec_Master_Log_Pos
```

local read position
of sql_thd

remote read position
of sql_thd

These positions correspond to the end_log_pos of the last statement executed by sql_thread

mysql> show slave status\G

Slave_IO_State: Waiting for master

Slave_IO_Running: Yes

Slave_SQL_Running: Yes

Master_Log_File: master-bin.000001

Read_Master_Log_Pos: 1818

io_thread
remote pos

Relay_Log_File: relay-bin.000001

Relay_Log_Pos: 251

sql_thread
local pos

Relay_Master_Log_File: master-bin.000001

Exec_Master_Log_Pos: 1818

sql_thread
remote pos

Fields re-ordered for display

Running Example

```
~/sandboxes$ make_rep1_sandbox --circular 2 5.1.55
```

Percona-Server-5.1.55-rel12.6-200-Linux-x86_64

Node1 = Active master | Node2 = Passive Master

```
node1 > create table foo ( id int not null
auto_increment primary key, v varchar(255) not null
default '', unique (v)) engine=innodb;
Query OK, 0 rows affected (0.01 sec)
```

```
node1 > insert into foo (v) values ('a'), ('b');
Query OK, 2 rows affected (0.01 sec)
Records: 2 Duplicates: 0 Warnings: 0
```

Using mysqlbinlog on masters

at 1610

#110318 14:20:12 server id 101 **end_log_pos 1683**

Query thread_id=8 exec_time=0 error_code=0

SET TIMESTAMP=1300483212/*!*/;

BEGIN/*!*/;

at **1683**

#110318 14:20:12 server id 101 **end_log_pos 1791**

Query thread_id=8 exec_time=0 error_code=0

SET TIMESTAMP=1300483212/*!*/;

INSERT INTO `foo` VALUES (1, 'a'), (2, 'b')/*!*/;

at **1791**

#110318 14:20:12 server id 101 end_log_pos 1818

Xid = 32

COMMIT/*!*/;

Using mysqlbinlog on relay logs

at 1184

```
#110318 14:20:12 server id 101  end_log_pos 1683
Query    thread_id=8      exec_time=0      error_code=0
SET TIMESTAMP=1300483212/*!*/;
BEGIN/*!*/;
```

at 1257

← end_log_pos is copied from master!

```
#110318 14:20:12 server id 101  end_log_pos 1791
Query    thread_id=8      exec_time=0      error_code=0
SET TIMESTAMP=1300483212/*!*/;
INSERT INTO `foo` VALUES (1, 'a'), (2, 'b')/*!*/;
```

at 1365

```
#110318 14:20:12 server id 101  end_log_pos 1818
      Xid = 32
COMMIT/*!*/;
```

Using mysqlbinlog on slave binlog

```
# at 1610
#110318 14:20:12 server id 101  end_log_pos 1674
Query    thread_id=8      exec_time=15    error_code=0
SET TIMESTAMP=1300483212/*!*/;
BEGIN/*!*/;
```

Here, **exec_time** means “slave lag”!

```
# at 1674
#110318 14:20:12 server id 101  end_log_pos 1782
Query    thread_id=8      exec_time=15    error_code=0
SET TIMESTAMP=1300483212/*!*/;
INSERT INTO `foo` VALUES (1, 'a'), (2, 'b')/*!*/;
```

```
# at 1782
#110318 14:20:12 server id 101  end_log_pos 1809
      Xid = 35
COMMIT/*!*/;
```

End of the Review

Any (short) Questions?

Finally, the meat of the talk!



My Approach

When replication stops...

- Figure out why it stopped
 - This usually tells you how to fix it
 - Generally does not take much time
- Determine extent of corruption / data loss
 - How much data is missing?
 - Were statements run more than once?
 - What is the *business impact*?
 - Don't over-analyze. Estimate is probably OK at this step.

My Approach

Now that you know *why* it stopped...

- Choose your restore method
 - Do you prioritize ***site availability*** or ***data integrity***?
- Validate slave consistency
 - If possible, before allowing traffic
 - if necessary, use tools to resync the slave
- Have a “plan B”
Sometimes you just need to rebuild

Understand What Failed

Where and **why** the failure happens matters a lot!

- Understand the basics!
- Know your own architecture!

How you solve it is...

- Learned from experience
- Common principles
- Similar even for different organizations

Two Times when Replication Fails

Server OK but replication stopped

- Writes on the slave
- File corruption in binary or relay-log
- Documented limitation in MySQL
- Un/known Bugs?

Crash caused replication to stop

- Hardware fails
- `mysqld` crash
- Kernel panic / out-of-memory killer
- `kill -9 `pidof mysqld``
- Host recently restarted and “rolled back” status file

Two Times when Replication Fails

Server OK but replication stopped

Dedicated Monitoring!!

Seconds_Behind_Master lies.

Watch IO and SQL thread state and position.

Use a heartbeat table.

Crash caused replication to stop

skip-slave-start

innodb_overwrite_relay_log_info (only in percona-server)

Common Situations

- Writes on the slave
- Documented limitations & bugs
- File Corruption
- Hardware Failures

Writes on the Slave

- (1) When it's just a slave
- (2) When it's also a master

Write on Slave (1)

(when a slave is just a slave)

```
node2 > insert into foo (v) values ('bad insert');
```

```
node2 > select * from foo;
```

1	a
2	b
3	bad insert

```
node1 > insert into foo (v) values ('c');
```

```
node2 > show slave status\G
```

```
Last_Errno: 1062
```

```
Last_Error: Error 'Duplicate entry '3' for key 'PRIMARY'' on query. Default database: 'test'.
```

```
Query: 'insert into foo (v) values ('c')'
```

Write on Slave (1)

(when a slave is just a slave)

Sample binlog from slave (aka node2)

```
#110328 13:55:52 server id 101 end_log_pos 640
  Query      thread_id=7      exec_time=0
error_code=0
INSERT INTO `foo` VALUES (1, 'a'), (2, 'b')
```

```
#110328 13:56:34 server id 102 end_log_pos 877
  Query      thread_id=7      exec_time=0
error_code=0
SET TIMESTAMP=1301345794/*!**/;
insert into foo (v) values ('bad insert')
```

Write on Slave (1)

(when a slave is just a slave)

- DELETE record from slave; optionally, insert to master
 - may preserve data integrity
 - insert to master allows you to keep the record
 - more time-consuming process than SKIP_COUNTER
 - foreign keys and triggers make this very complicated!

Write on Slave (1)

(when a slave is just a slave)

- `SET GLOBAL SQL_SLAVE_SKIP_COUNTER=1;`
 - easiest method
 - data inconsistency may propagate to other tables, eg. if you use `INSERT . . SELECT` or triggers
 - must sync table later with `mk-table-sync`

Write on Slave (1)

(when a slave is just a slave)

- `auto_increment_increment` & `_offset`
 - would have prevented failure in this example (slave insert will always create different ``id`` than master insert)
 - safe to do even if you “never” write to slave
- `master = even, slave = odd`
 - makes it trivial to identify bad writes - ``id`` will be odd

Write on Slave (2)

(when a slave is also a master)

```
node1 > insert into foo (v) values ('c');
node1 > SHOW SLAVE STATUS\G
Error 'Duplicate entry '3' for key 'PRIMARY' on
query. Default database: 'test'. Query: 'insert
into foo (v) values ('bad insert')'
-----
node2 > insert into foo (v) values ('bad insert');
node2 > SHOW SLAVE STATUS\G
Error 'Duplicate entry '3' for key 'PRIMARY' on
query. Default database: 'test'. Query: 'insert
into foo (v) values ('c')'
```

Note: I used SLAVE STOP; on both hosts to simulate simultaneous writes

Write on Slave (2)

(when a slave is also a master)

- Fixing master-master is same principle as master-slave...
- Except you might accidentally corrupt your primary master!
- So be extra careful...

For example:

- DELETE record from secondary; optionally, insert to primary
 - Use `SET SESSION SQL_LOG_BIN = 0` for DELETE
 - But allow INSERT to replicate normally

Write on Slave (2)

(when a slave is also a master)

- `SET SQL_SLAVE_SKIP_COUNTER = 1`
 - have to do on both masters
 - must sync table later with `mk-table-sync`
- `auto_increment_increment & _offset`
 - A “must have” if you use `AUTO_INC` + master-master replication + fail-over (whether manual or automatic)
 - OK to use even/odd with two masters + many slaves

Common mistake re: SKIP

<http://code.google.com/p/php-mysql-master-slave-replication-monitor/>

“will not only check on the health of your MySQL Replication setup, but it can **also automatically heal and restart the MySQL Replication**. Self-Healing MySQL Replication is not a new idea”

*This is really bad! Each skip => more corruption.
Always use mk-table-checksum/-sync after skipping a statement
... and never skip blindly!*

Documented Limitations

Documented limitations

- Statement-Based Replication (SBR) has many limitations
- Some are documented here
 - <http://dev.mysql.com/doc/refman/5.1/en/replication-features.html>
 - **33** subsections
- Row-Based Replication (RBR) avoids some limitations
- However, it has some of its own
 - All tables require a PRIMARY KEY
 - Can be more difficult to do “online alter table” than SBR
 - <http://dev.mysql.com/doc/refman/5.1/en/replication-rbr-usage.html>

Documented limitations

- Some limitations are “documented” as open bug reports
 - **37** active bugs (March 2011)

Best approach:

- Get to know the limitations & open bugs
- Avoid them like the plague
- Change your application as necessary

Examples of limitations

- Non-Deterministic functions or statements
 - UUID, CURRENT_USER, FOUND_ROWS
 - UPDATE | DELETE with LIMIT but no ORDER BY
 - Many of these solved by RBR
- TEMPORARY and MEMORY tables
- Stored Routines or Triggers that use logical expressions or dynamic SQL which could execute differently on the slave.

Examples of limitations

- Floating-point values
- Different character sets on master and slave
- Different table definition - may work in some cases, but more limited with RBR
- Mixing InnoDB and MyISAM in one transaction
 - Never a good idea
 - Behavior changed multiple times within 5.1.xx versions

UUID and SBR do not mix

```
node1 > insert into foo (v) select UUID();
node1 > show warnings;
| Note | 1592 | Statement may not be safe to log
in statement format. |
```

```
node1 > select * from foo;
| 1 | 359ac1db-5973-11e0-bca9-080027ca1802 |
node2 > select * from foo;
| 1 | 359bd78d-5973-11e0-9e6c-080027ca1802 |
```

This slide only applies to statement-based replication.

Temporary Tables (are evil)

Step 1) `CREATE TEMPORARY TABLE xyz ...;`

Step 2) `<slave restarts>`

Step 3) `INSERT ... SELECT ... FROM xyz ...;`

Last_SQL_Error: Error 'Table 'test.xyz' doesn't exist' on query. Default database: 'test'. Query: 'insert into foo (v) select * from xyz'

With many parallel temporary tables, may run into bug with `@pseudo_thread_id` - <http://bugs.mysql.com/bug.php?id=51226>

This slide only applies to statement- or mixed-mode replication.

File Corruption

File Corruption

Slave may stop if any of these files become corrupt

Binary log | Relay log | Table data | Index data

Examples of SHOW SLAVE STATUS

Last_IO_Error: 1236

Last_IO_Error: Got fatal error 1236 from master
when reading data from binary log

Last_SQL_Errno: 1594

Last_SQL_Error: Relay log read failure: Could not
parse relay log event entry.

If Master Binlog is Corrupt...

- Verify it with `mysqlbinlog [--verbose --base64]`
- Find extent of corruption / next “good” transaction.
- `CHANGE MASTER TO Master_Log_Pos = <good_pos>`
- Analyze bad section of log. Identify all affected tables.
- Use `mk-table-sync` to resync those tables once replication catches up.
- Of course, try to resolve underlying (hardware) cause of the corruption too!

If Slave Relay Log is Corrupt...

- Verify it with `mysqlbinlog <relay-log-file>`
- Check master binlog to make sure it is not corrupt.
 - If it is, see previous slide.
- Re-fetch the corrupted relay log.
`CHANGE MASTER TO`
 `Master_Log_Pos = <Exec_master_log_pos>;`
`SLAVE START;`
- If this occurs frequently, check for network problems.

If a Table is Corrupt...

- Stop replication and stop all traffic to that server
- Mount the file system read-only while you work (if possible)
- Check mysql error log - it may contain more info
- Even after “fixing” the corruption, `mk-table-sync`
- Sometimes, only option is restore from a backup

Hardware Failures

Dell PowerEdge with 8-disk RAID10
Perc 5i with 512MB BBU

HW Failure #1

Unplanned master restart → all slaves stop with error:
'Client requested master to start replication
from impossible position'

Process:

- Compare each SLAVE STATUS to master binlog
- Realize that Exec_Master_Log_Pos different on each slave, and *greater than master's log file size!*
- Panic for a minute...

HW Failure #1

Possible solutions...

- Promote slave that read the most
- Isolate “extra” events from slave binlog [and replay on master]
- Force slaves to resume from masters next (empty) binlog, then `mk-table-sync` to remove “extra” events

You can prevent this (most of the time) by using
`innodb_flush_log_at_trx_commit = 1`
`sync_binlog = 1`

HW Failure #2

Replaced a failed disk in a slave host. After restart, replication fails with duplicate key error

Process:

- Check error log... *there's no slave stop coordinates*
- Realize file system in read-only mode before shut down
- Was master `.info` rolled back by file system journal?

HW Failure #2

Possible solutions...

- Guess where slave really stopped, then `CHANGE MASTER`
But what if data files also rolled back?
- Lots of `SET SQL_SLAVE_SKIP_COUNTER` and some prayers, then use `mk-table-sync`
- Rebuild the slave

You can prevent this if you use Percona-Server by using `innodb_overwrite_relay_log_info`

The End

Any Questions?

(a few “war stories” on next slide)

War Stories

- Slave generates different auto_inc values for INSERT inside an ON_INSERT TRIGGER. This goes unnoticed for months, then starts causing problems.
- During MMM migration, a few uncommitted transactions get left behind and cause duplicate key errors on both nodes
- Major hardware failure in a geo-distributed three node replication ring. Each host is written to by local processes. There are some tables only written on single host, but also some shared tables that all write to. How do you determine which of remaining two masters is “good”?