FOSDEM 2012

Replication features of 2011

What they were
How to get them
How to use them

Sergey Petrunya
MariaDB

MySQL Replication in 2011: overview

Notable events, chronologically:

- MySQL 5.5 GA (Dec 2010)
- MariaDB 5.3 (Beta in June, RC in December)
  - Group commit
- Replication listener
- MySQL 5.6 MS releases (April, October, December)

MySQL 5.5

MySQL 5.5 —merged to→ **Percona Server 5.5**

- **Semi-synchronous replication plugin**
  (delay reporting the commit to client until it is in the relay log of some slave)
- **Slave fsync tuning** (sync_relay_log_info, sync_master_info, sync_relay_log)
- **Automatic relay log recovery** (--relay-log-recovery=1)
- **Replication Heartbeat** (CHANGE MASTER SET master_heartbeat_period=)
- **Per-server filtering** (CHANGE MASTER … IGNORE_SERVER_IDS=)
- **RBR slave type conversions** (--slave-type-conversions=ALL_{NON}_LOSSY)
- **Individual Log Flushing** (FLUSH ERROR|RELAY|BINARY|GENERAL LOGS)
- **SHOW RELAYLOG EVENTS**

MariaDB 5.3

Releases

- 5.3.1 Beta- July 2011, 5.3.2 Beta – Oct 2011
- 5.3.3 RC – Dec 2011

Replication features

- Group commit for binary log
- Annotations for RBR events
- Checksums for binlog events
- Enchancements for START TRANSACTION WITH CONSISTENT SNAPSHOT
- Optimized RBR for tables without primary key

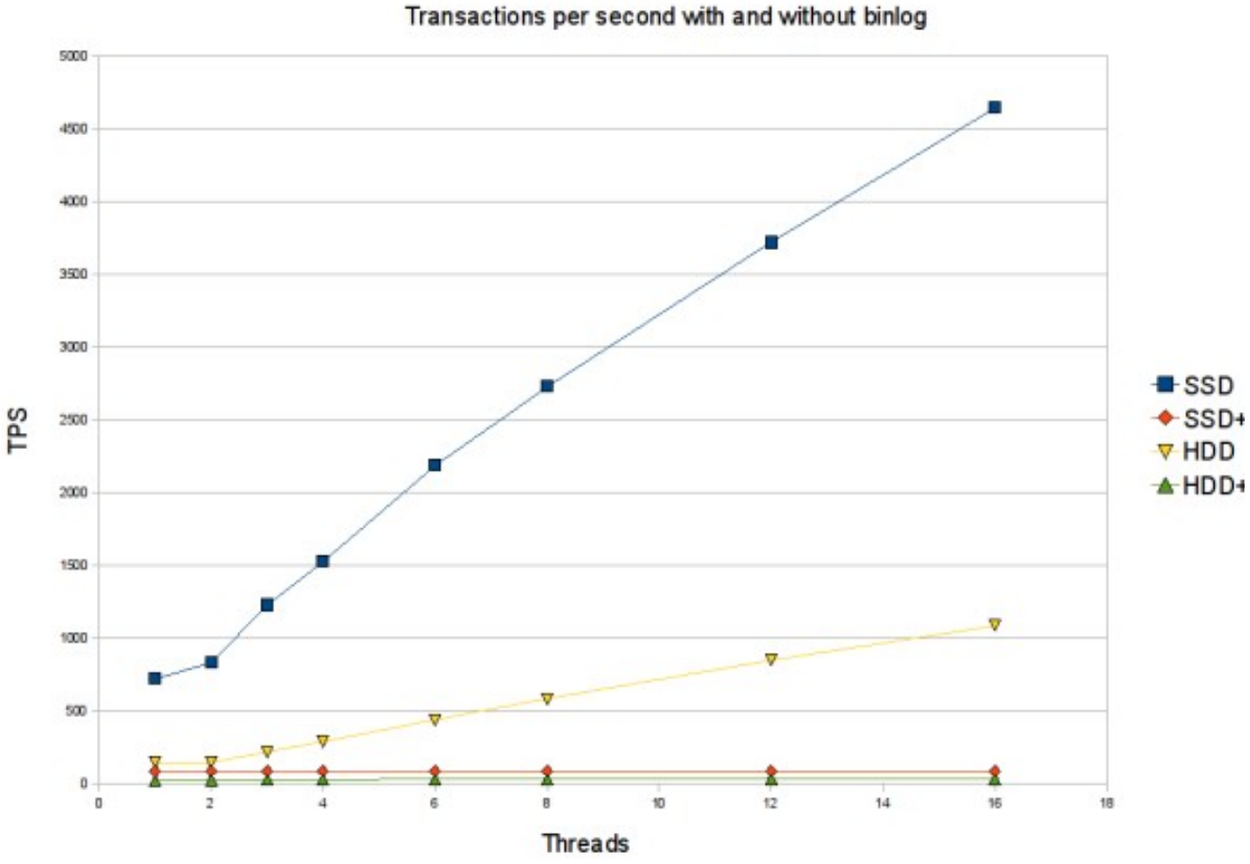
Percona Server 5.5

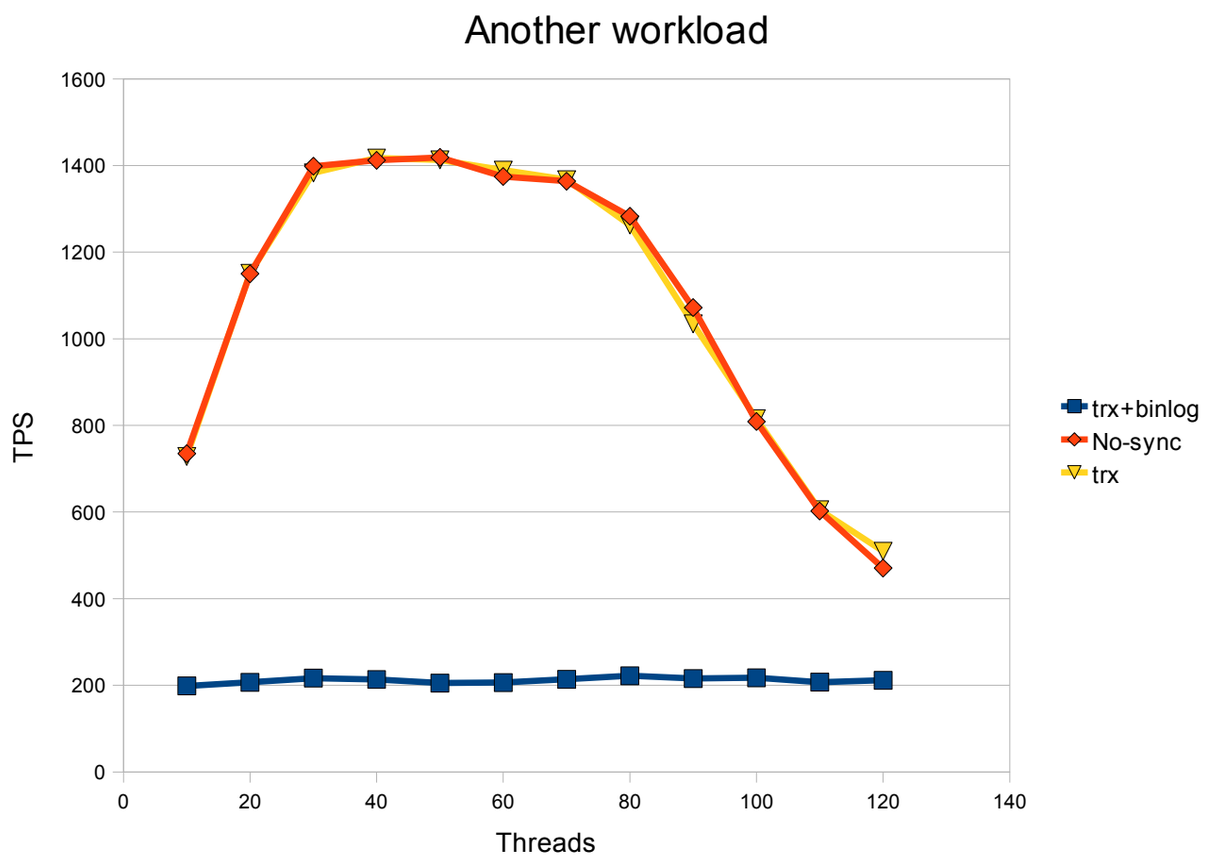
MySQL 5.6

Percona's patch

Group commit for binary log

- To get the D in ACID, one needs
  - InnoDB: **innodb_flush_log_at_trx_commit=1**
  - Binlog: **sync_binlog=1**
- When you have both ON, group commit doesn't work
  - ~200 transactions per second max

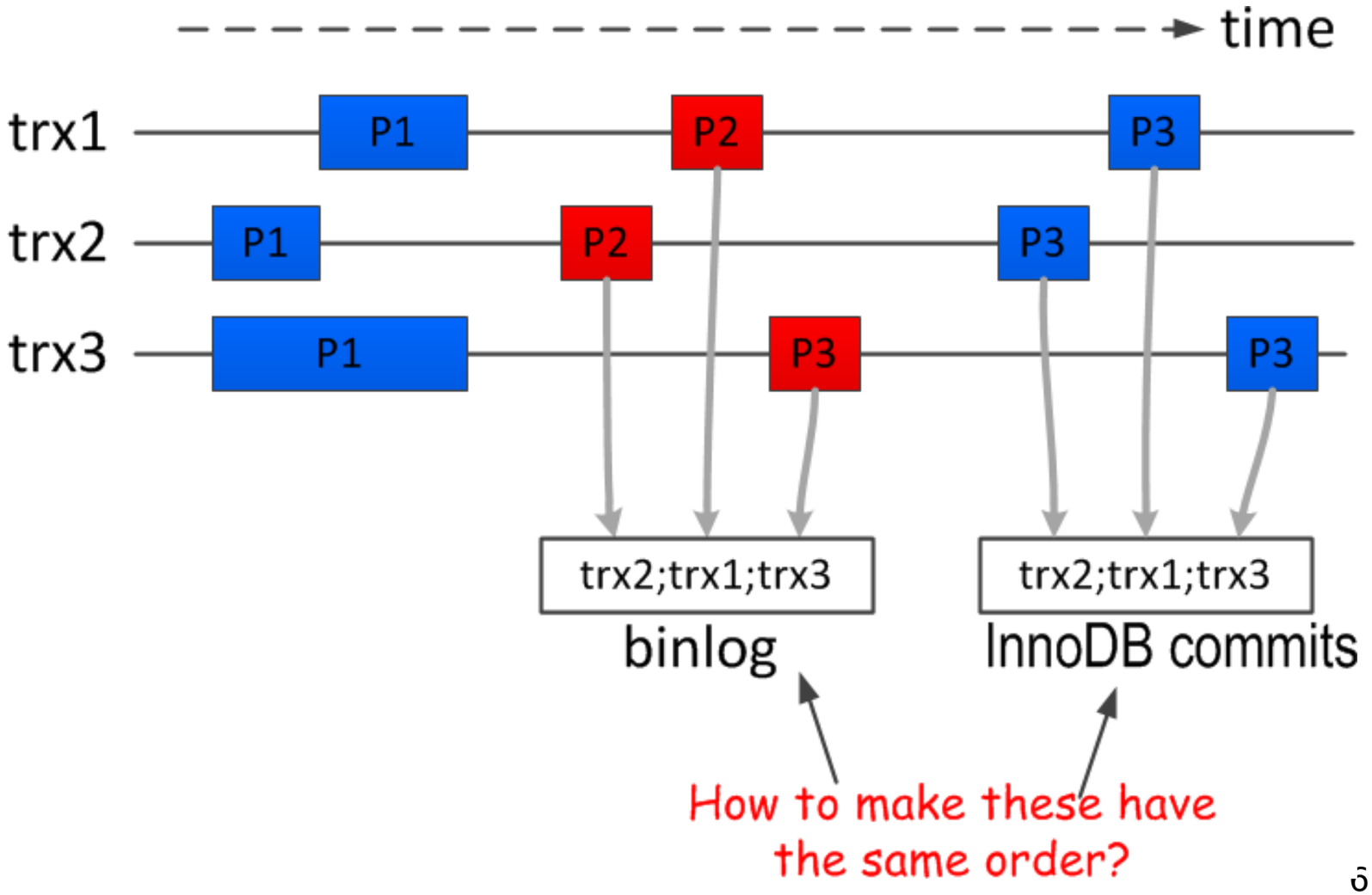




14:46:18

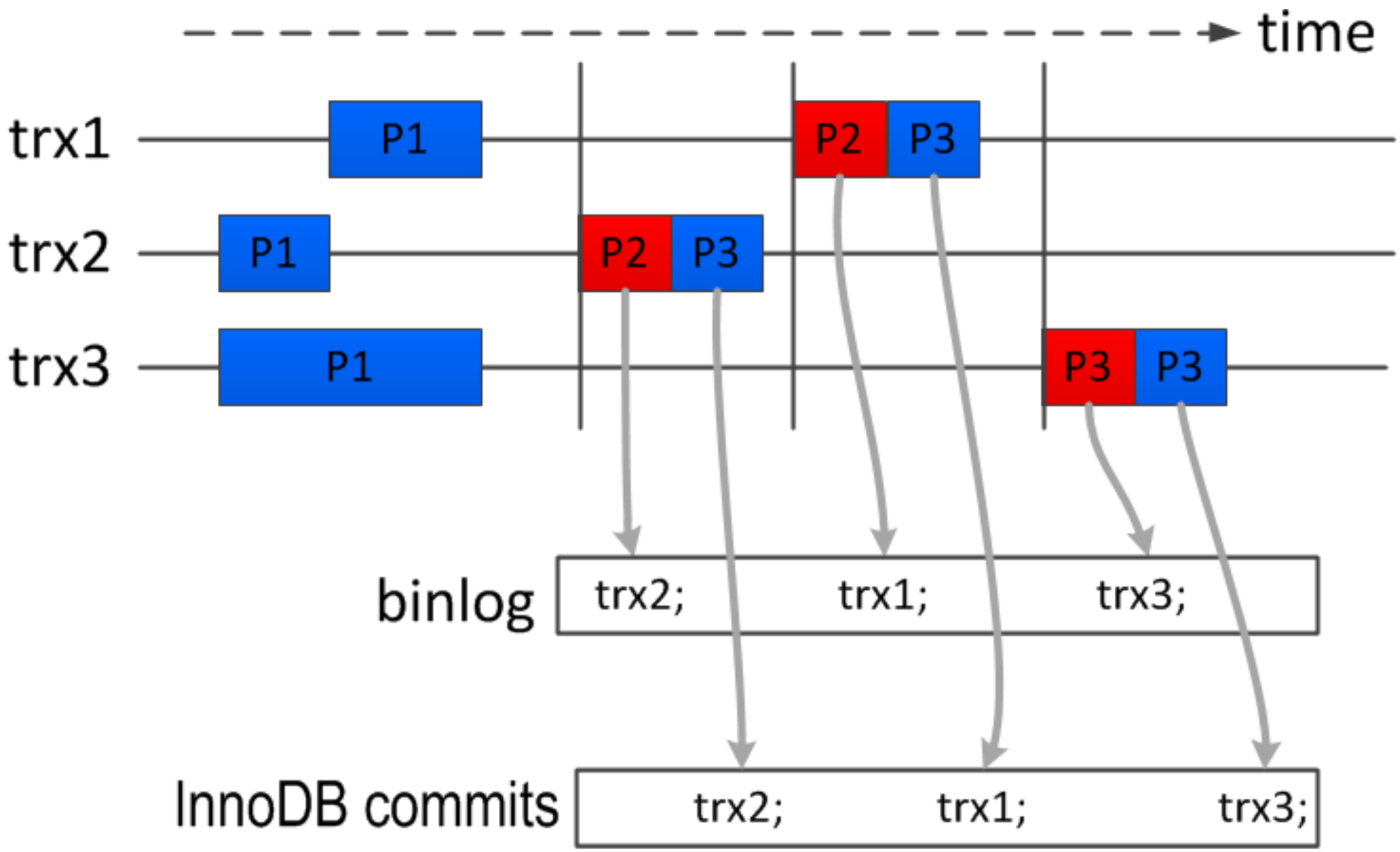
Group commit for binary log (2)

- Transactional operation with enabled binlog
  1. Write transaction to InnoDB transaction log
  2. Write transaction to binlog
     - This is the actual commit
  3. Write a commit record to InnoDB.



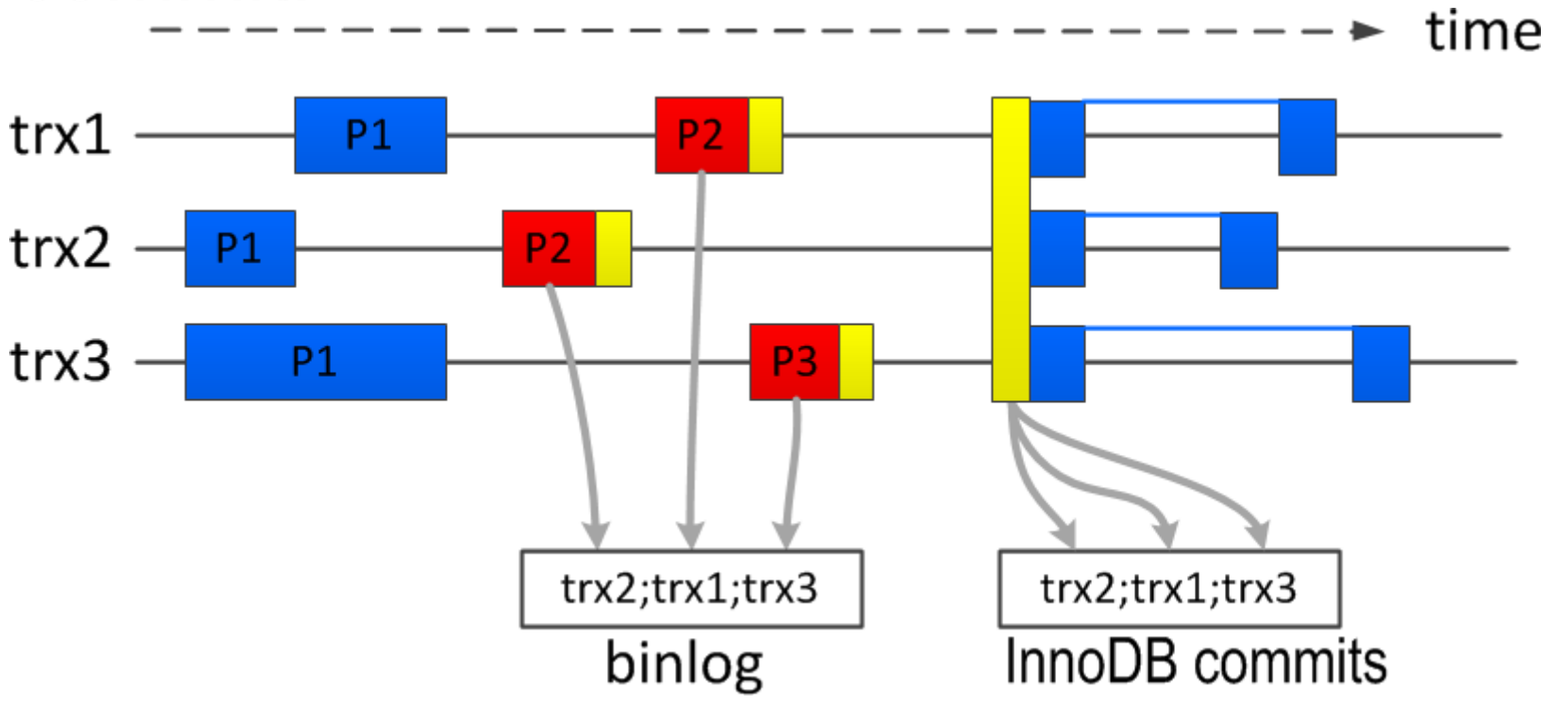
Group commit for binary log (2)

- Pre-group-commit way to ensure the same ordering:



Group commit for binary log (3)

- Group commit:



- Multiple implementations: Facebook, MariaDB, proposal by Oracle
- All different, based on similar ideas:
  - Get tickets when writing to the binlog
  - Fix InnoDB commits in the same order they were done with binlog
  - (Maybe) Split P3 into the “fix ordering” (fast) and write-out (slow) phases

Group commit for binary log

- So far, MariaDB 5.3 has the best
  - Ported into Percona Server 5.5

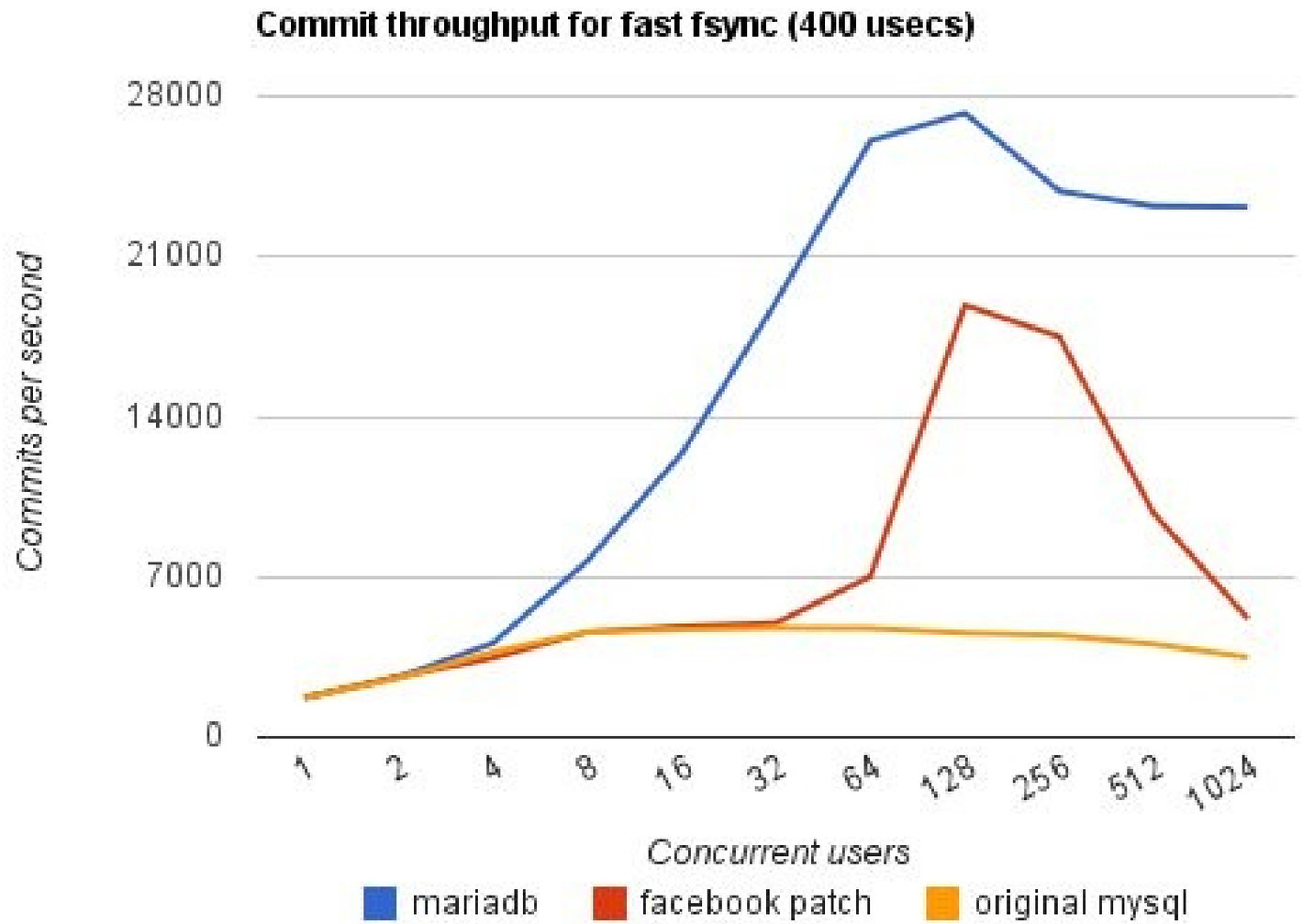


Chart from “Group commit in MariaDB is fast” post by Mark Callaghan

MariaDB 5.3: Annotated RBR events

Every RBR event is accompanied by its source query

- Master: --binlog_annotate_row_events
- Slave: --replicate-annotate-rows-events

```
MariaDB [test]> show binlog events in 'pslp.000299';
+-------------+-----+--------------+-----------+-------------+-----------------------------------------------+
| Log_name    | Pos | Event_type   | Server_id | End_log_pos | Info                                          |
+-------------+-----+--------------+-----------+-------------+-----------------------------------------------+
| pslp.000299 |   4 | Format_desc  |         1 |         245 | Server ver: 5.3.4-MariaDB-rc-log, Binlog ver: 4 |
| pslp.000299 | 245 | Query        |         1 |         313 | BEGIN                                         |
| pslp.000299 | 313 | Annotate_rows |         1 |         379 | /* app1 */ insert into t1 values('foo'),('bar') |
| pslp.000299 | 379 | Table_map    |         1 |         422 | table_id: 15 (test.t1)                        |
| pslp.000299 | 422 | Write_rows   |         1 |         461 | table_id: 15 flags: STMT_END_F                |
| pslp.000299 | 461 | Query        |         1 |         530 | COMMIT                                        |
| pslp.000299 | 530 | Query        |         1 |         598 | BEGIN                                         |
| pslp.000299 | 598 | Annotate_rows |         1 |         664 | /* app2 */ update t1 set a='test' where a='foo' |
| pslp.000299 | 664 | Table_map    |         1 |         707 | table_id: 15 (test.t1)                        |
| pslp.000299 | 707 | Update_rows  |         1 |         759 | table_id: 15 flags: STMT_END_F                |
+-------------+-----+--------------+-----------+-------------+-----------------------------------------------+
```

```
# at 379
# at 422
#120203 16:25:11 server id 1  end_log_pos 379   Annotate_rows:
#Q> /* app1 */ insert into t1 values('foo'),('bar')
#120203 16:25:11 server id 1  end_log_pos 422   Table_map: `test`.`t1` mapped to number 15
#120203 16:25:11 server id 1  end_log_pos 461   Write_rows: table id 15 flags: STMT_END_F

BINLOG '
J9IrTxMBAAAAKwAAAKYBAAAAAA8AAAAAAAEABHRlc3QAAnQxAAEPAgwAAQ==
```

MySQL 5.6: annotated events, too

- Implementation is [naturally] very similar to MariaDB's. The user interface is different:
- Master: --binlog-rows-query-log-events

```
MySQL [test]> show binlog events;
+-------------+-----+-------------+-----------+-------------+-----------------------------------------------------+
| Log_name    | Pos | Event_type  | Server_id | End_log_pos | Info                                                |
+-------------+-----+-------------+-----------+-------------+-----------------------------------------------------+
| pslp.000001 |   4 | Format_desc |         1 |         114 | Server ver: 5.6.5-m8-debug-log, Binlog ver: 4       |
| pslp.000001 | 114 | Query       |         1 |         215 | use `test`; create table t1 (a varchar(12))         |
| pslp.000001 | 215 | Query       |         1 |         283 | BEGIN                                               |
| pslp.000001 | 283 | Rows_query  |         1 |         350 | # /* app1 */ insert into t1 values('foo'),('bar')   |
| pslp.000001 | 350 | Table_map   |         1 |         393 | table_id: 66 (test.t1)                              |
| pslp.000001 | 393 | Write_rows  |         1 |         432 | table_id: 66 flags: STMT_END_F                      |
| pslp.000001 | 432 | Xid         |         1 |         459 | COMMIT /* xid=5 */                                  |
| pslp.000001 | 459 | Query       |         1 |         527 | BEGIN                                               |
| pslp.000001 | 527 | Rows_query  |         1 |         594 | # /* app2 */ update t1 set a='test' where a='foo'   |
| pslp.000001 | 594 | Table_map   |         1 |         637 | table_id: 66 (test.t1)                              |
| pslp.000001 | 637 | Update_rows |         1 |         678 | table_id: 66 flags: STMT_END_F                      |
| pslp.000001 | 678 | Xid         |         1 |         705 | COMMIT /* xid=6 */                                  |
+-------------+-----+-------------+-----------+-------------+-----------------------------------------------------+
```

A compatibility problem

- MariaDB 5.3 added **Annotate_rows** event
- MySQL 5.6 added **Rows_query** event
- They are different events
  - MariaDB 5.3 can't understand **Rows_query**
  - MySQL 5.6 can't understand **Annotate_rows**
- Seeing unknown event in binlog → error
  - (It is not safe to continue after you've skipped an unknown operation)
- MySQL 5.6 will have a flag to mark “ignorable” binlog events
  - We'll merge this to MariaDB
  - This will make binary logs compatible again

Optimized RBR for tables with no PK

- An RBR event is essentially something like this:

```
with table
    $tbl=`dbname.tablename` (col1 INT, col2 CHAR(N),...)
do
    delete in $tbl row with (col1,col2) = (10, 'foo')
```

- Execution before MariaDB 5.3
  - Use the *first* index to find records
    - If the table has PRIMARY KEY is present, it will be the first (OK)
    - If there is no PK, may end up using poor index
      - If it is an FT index, replication stops with error
- In MariaDB 5.3
  1. If there is a PRIMARY KEY, use it
  2. Otherwise, use first UNIQUE index w/o NULLs
  3. Otherwise, use the most-selective non-fulltext index

MySQL 5.6

- **Crash-safe slave**
  - replication info tables
- **Crash-safe master**
  - binary log recovery
- **Replication event checksums**
- **Time delayed replication**
- **Optimized row-based logging**
  - --binlog_row_image={full,minimal,noblob}
- **Informational log events**
- **Remote backup of binary logs**
- **Server UUIDs**
  - Replication topology detection
- **Parallel event execution on slave**

MariaDB 5.3

Parallel event execution: background

Slave SQL thread algorithm:

```
while (get next next event from relay log) {
   execute event;
}
```

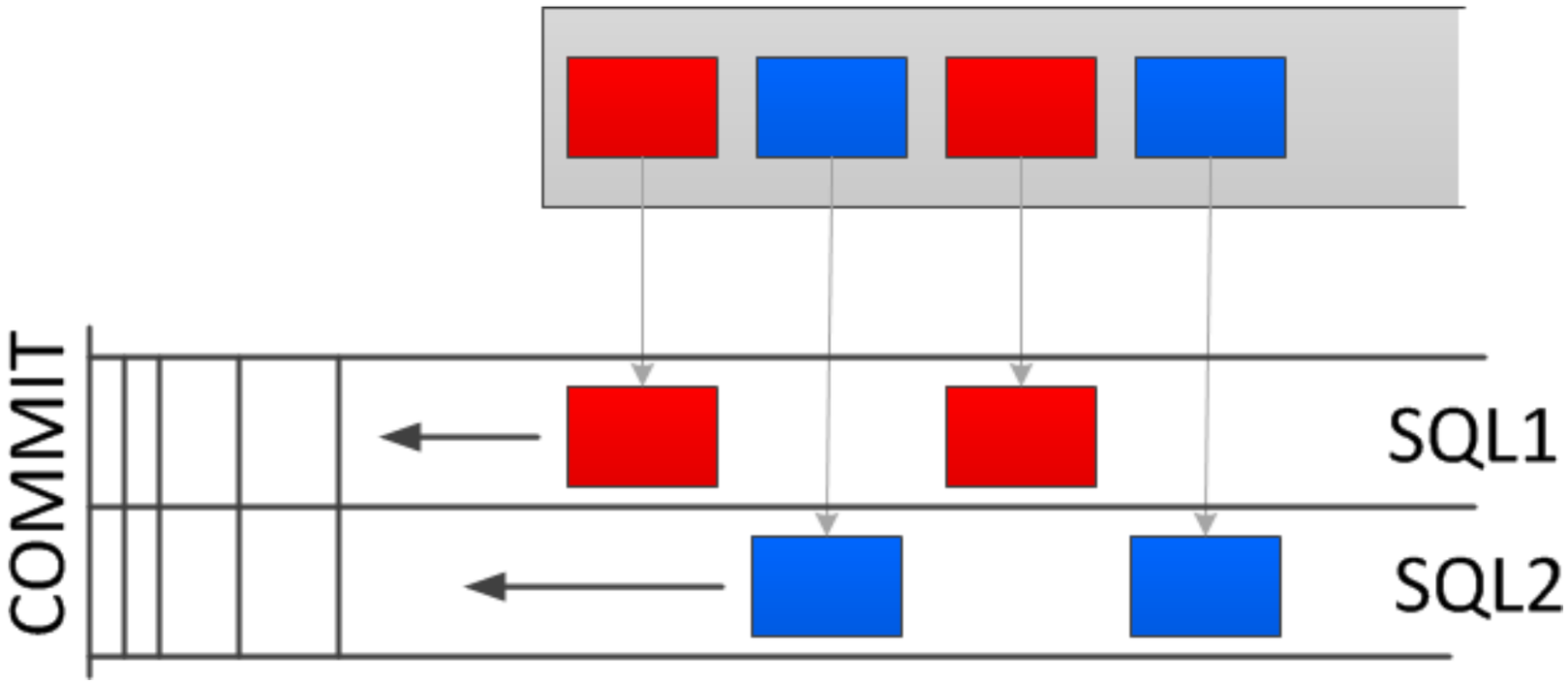
Problems

- Cannot use multiple CPUs
- Cannot enqueue multiple disk requests

=> slave may be unable to keep up with the master

Solution#1: Parallel slave

The idea: execute binlog events in parallel



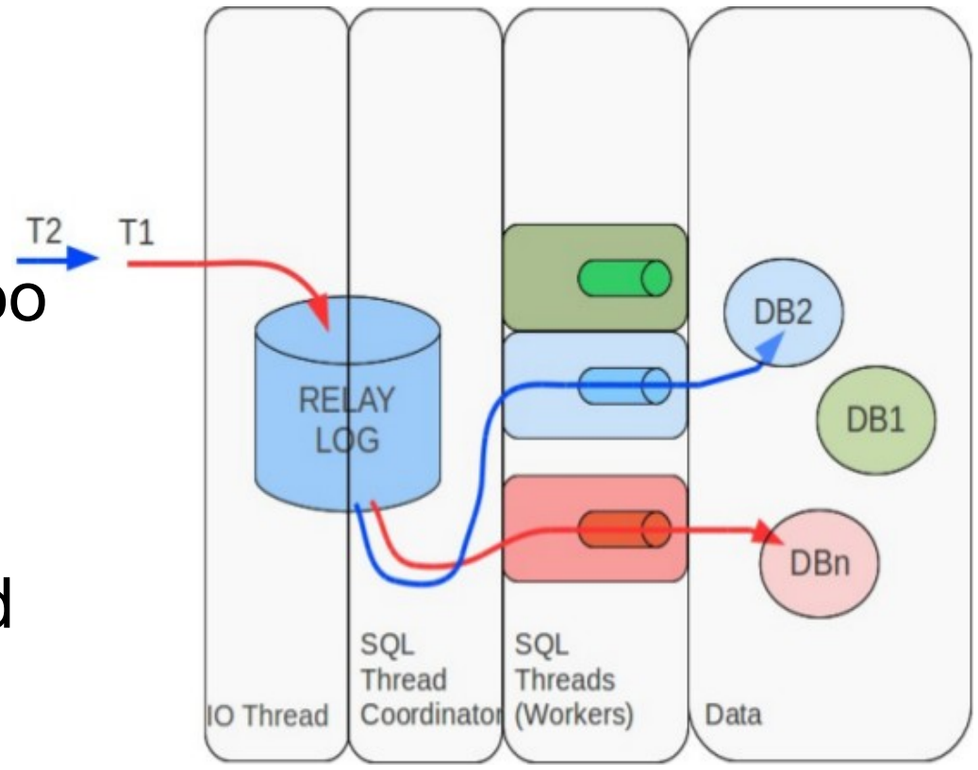
Problems to address:

- Out-of-order event execution must produce the same result
- Slave may be in a state that never existed on the master
  - Including being in this state after the crash

MySQL 5.6's parallel slave

Basic idea: partition events by hash (db_name)

- Changes to different db's can be ran out-of-order
  - This should be ok for your application, too
- Exec_Master_Log_Pos is a low-watermark now
  - Slave keeps track of what was commited above the watermark and will use it for recovery



(picture from Lars Thallman's OSCON 2011 talk)

The main problem

- Un-even distribution of load between workers
- A frequent case: most of updates are in one huge table

Alternate approach #1: pre-heating

IO-bound SQL thread can be made faster by

- Pre-read relay log
- Convert UPDATEs/DELETEs to SELECTs
- Execute the SELECTs
  - Will cause relevant pages be pre-loaded into buffer pool
  - Can be done in parallel

- First implementation: mk-slave-prefetch
  - Written in Perl
  - Baron Schwartz (author): “Don't use it unless you're Facebook”

- Second implementation
  - “Replication Booster” by Yoshinori Matsunobu

Replication booster

http://yoshinorimatsunobu.blogspot.com/2011/10/making-slave-pre-fetching-work-better.html

Replication booster

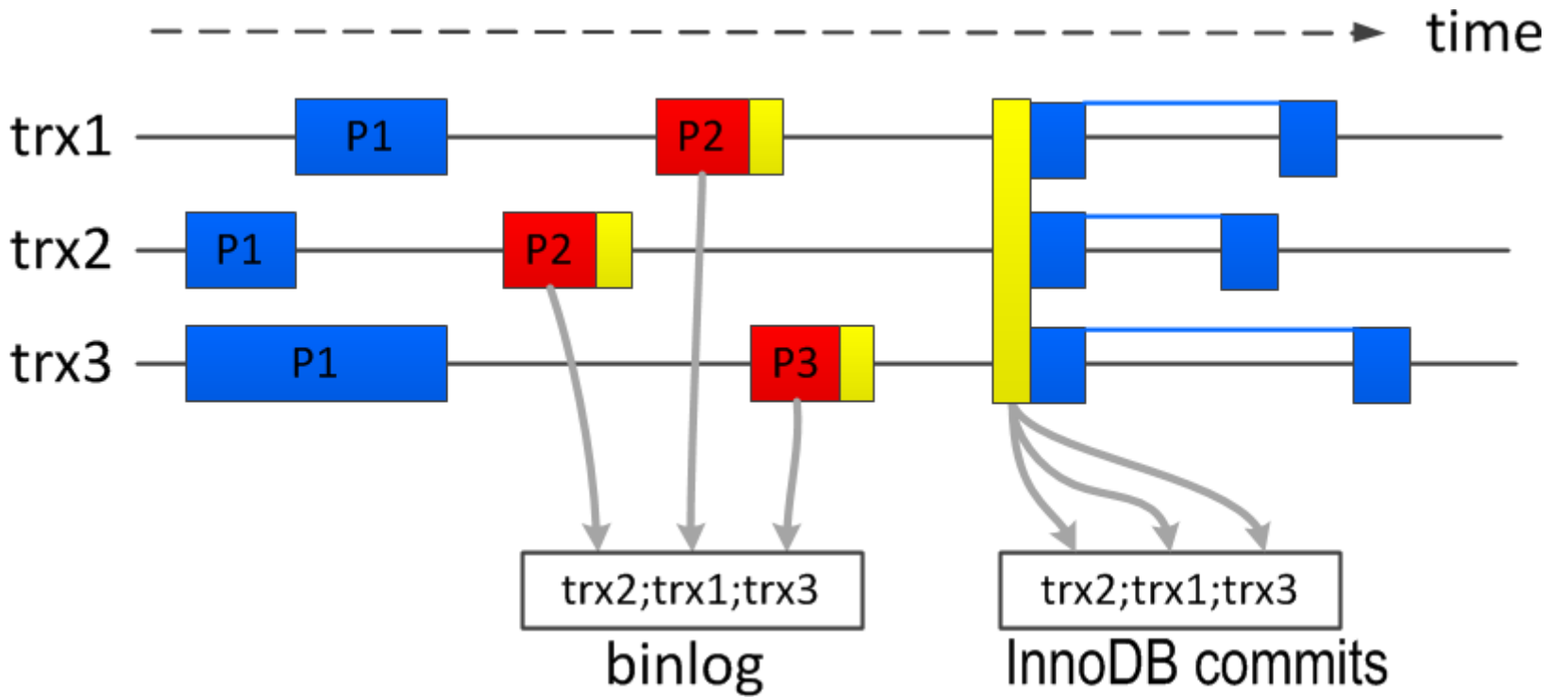
- Written in C++ using **mysql-replication-listener** API
- Uses regexps to convert UPDATEs/DELETEs to SELECTs
- Doesn't support RBR events
- Uses multiple threads to parse/re-write binlog events
- Keeps itself 0..3 sec ahead of the SQL thread

Performance improvements

- Benchmark: from 1779 updates/sec to 5,418 updates/sec
- 30..300% improvement on various slaves
- No negative impact when all data is already in cache

Alternative approach #2: group commit

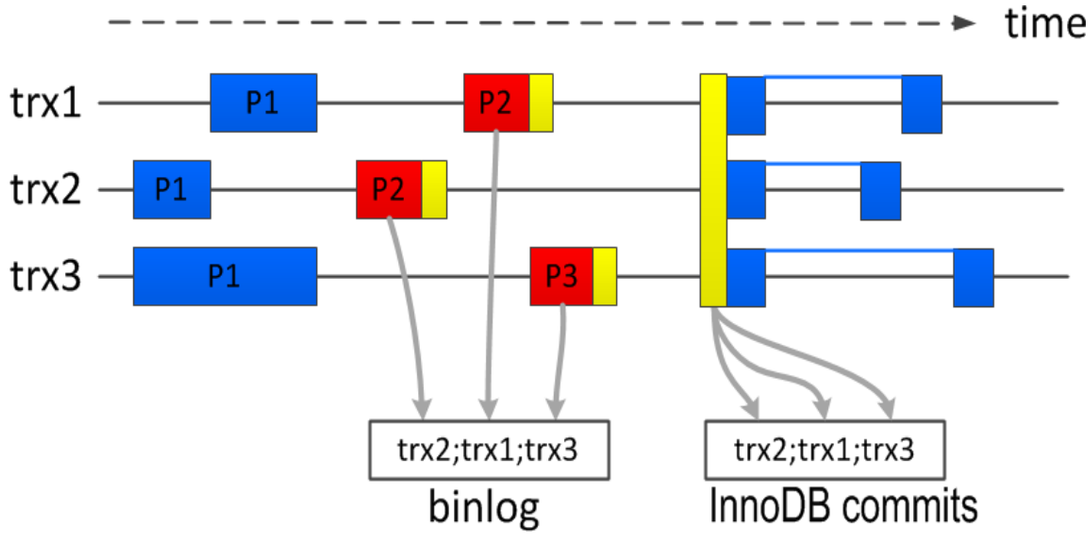
- A possible future development in MariaDB.
- Basic idea: c**ommits in a group do not conflict**
  - They were in “Prepare” stage together



- Hence, commits from the same group can be done out-of-order, in parallel

Alternative approach #2: group commit (2)

- How does the slave know which commits were in a group?
- Master should write markers into the binlog



- Pro
  - Can be better than hash(db_name) approach of MySQL 5.6
  - Especially for the “all updates hit one-big-table” case
- Contra
  - Grouping in binlog depends on how much the master was loaded

2011 in replication

- **Performance improvements**
  - Group commit for the binary log (MariaDB 5.3, Percona 5.5)
  - Multi-Threaded Slave (MySQL 5.6)
- **Row-Based Replication**
  - RBR event annotation (MariaDB 5.3, MySQL 5.6)
  - binlog_row_image={full,minimal,noblob} (MySQL 5.6)
  - Speedup for RBR of tables without PK (Percona~>MariaDB 5.3)
- **Safety**
  - Binlog checksums (MySQL 5.6 -> MariaDB 5.3)
  - Crash-safe master, Crash-save slave (MySQL 5.6)
- **Pseudo-slaves**
  - launchpad.net/mysql-replication-listener
  - mysqlbinlog --read-from-remote-server –raw (MySQL 5.6)
- **Misc**
  - Delayed replication (MySQL 5.6)
  - START TRX WITH CONSISTENT SNAPSHOT (MariaDB 5.3)

Links

Most of replication developers blog!

- http://www.mysqlperformanceblog.com
- http://www.facebook.com/MySQLatFacebook
- http://kristiannielsen.livejournal.com/
- http://mysqlmusings.blogspot.com/
- http://yoshinorimatsunobu.blogspot.com
- http://datacharmer.blogspot.com/
- http://d2-systems.blogspot.com/
- (hopefully didn't forget anyone)

Thanks!

Q & A