MySQL Parallel Replication (LOGICAL_CLOCK): all the 5.7 (and some of the 8.0) details

Jean-François Gagné (System Engineer)
jeanfrancois DOT gagne AT booking.com
April 27, 2017 – Percona Live Santa Clara 2017
Based in Amsterdam since 1996

Online Hotel and Accommodation (Travel) Agent (OTA):
- +1.220.000 properties in 227 countries
- +1.200.000 room nights reserved daily
- +40 languages (website and customer service)
- +13.000 people working in 187 offices worldwide

Part of the Priceline Group

And we use MySQL:
- Thousands (1000s) of servers, ~90% replicating
- >150 masters: ~30 >50 slaves & ~10 >100 slaves
And we are hiring!
- MySQL Engineer / DBA
- System Administrator
- System Engineer
- Site Reliability Engineer
- Developer / Designer
- Technical Team Lead
- Product Owner
- Data Scientist
- And many more…

https://workingatbooking.com/
Session Summary

1. Introducing Parallel Replication (// Replication)
2. Reminders of previous session
3. MySQL 5.7: Logical Clock and Intervals
4. MySQL 5.7: Tuning Intervals
5. MySQL 8.0 and Group Replication: Write Set
// Replication

- Relatively new because it is hard
- It is hard because of data consistency
  - Running trx in // must give the same result on all slaves (= the master)
- Why is it important?
  - Computers have many Cores, using a single one for writes is a waste
  - Some computer resources can give more throughput when used in parallel
    (RAID1 has 2 disks $\rightarrow$ we can do 2 Read IOs in parallel)
    (SSDs can serve many Read and/or Write IOs in parallel)
Reminder

- MySQL 5.6 has support for schema based parallel replication
- MariaDB 10.0 has support for domain id based parallel replication and also has support for group commit based parallel replication
- MariaDB 10.1 adds support for optimistic parallel replication
- MySQL 5.7 adds support for logical clock parallel replication
  - In early version, the logical clock is group commit based
  - In current version, the logical clock is interval based
- MySQL 8.0 adds support for Write Set parallelism identification
MySQL 5.7: LOGICAL CLOCK

- MySQL 5.7 has two slave_parallel_type:
  - both need "SET GLOBAL slave_parallel_workers = N;" (with N > 1)
  - DATABASE: the schema based parallel replication from MySQL 5.6
  - LOGICAL_CLOCK: “Transactions that are part of the same binary log group commit on a master are applied in parallel on a slave.” (from the doc. but not exact: Bug#85977)
  - the LOGICAL_CLOCK type is implemented by putting interval information in the binary logs

- Slowing down the master to speedup the slave:
  - binlog_group_commit_sync_delay
  - binlog_group_commit_sync_no_delay_count

- We can expect the same problems as with MariaDB 10:
  - Problems with long/big transactions
  - Problems with intermediate masters (IM)
MySQL 5.7: LOGICAL CLOCK

- By default, MySQL 5.7 in logical clock does out-of-order commit:
  - There will be gaps ("START SLAVE UNTIL SQL_AFTER_MTS_GAPS;")
  - Not replication crash safe without GTIDs
  - And also everything else:
    binary logs content, SHOW SLAVE STATUS, skipping transactions, backups, …

- Using `slave_preserve_commit_order = 1` does what you expect:
  - This configuration does not generate gap
  - But it needs log-slave-updates, there is a feature request to remove this limitation: [Bug#75396](http://example.com/bugs/75396)
  - And it is still not replication crash safe (surprising because no gap): [Bug#80103](http://example.com/bugs/80103) & [Bug#81840](http://example.com/bugs/81840)
MySQL 5.7 – Intervals

- To understand MySQL 5.7, let’s look at something simpler first

- In MariaDB 10, each trx in the binlogs is tagged with a Group Commit Id ($cid$)

  ...  
  #150316 11:33:46 ... GTID 0-1-184 cid=2324  
  #150316 11:33:46 ... GTID 0-1-185 cid=2335  
  ...  
  #150316 11:33:46 ... GTID 0-1-189 cid=2335  
  #150316 11:33:46 ... GTID 0-1-190  
  #150316 11:33:46 ... GTID 0-1-191 cid=2346  
  ...  
  #150316 11:33:46 ... GTID 0-1-197 cid=2346  
  #150316 11:33:46 ... GTID 0-1-198 cid=2361  
  ...
MySQL 5.7 – Intervals’

- In MySQL 5.7, each transaction is tagged with two (2) numbers:
  - `sequence_number`: increasing id for each trx (not to confuse with GTID)
  - `last_committed`: sequence_number of the **latest trx** on which this trx depends
    (This can be understood as the “write view” of the current transaction)

- The `last_committed/sequence_number` pair is the parallelism *interval*

- Here an example of intervals for MySQL 5.7:

  . . .
  \#170206 20:08:33 . . . last_committed=6201  sequence_number=6203
  \#170206 20:08:33 . . . last_committed=6203  sequence_number=6204
  \#170206 20:08:33 . . . last_committed=6203  sequence_number=6205
  \#170206 20:08:33 . . . last_committed=6203  sequence_number=6206
  \#170206 20:08:33 . . . last_committed=6205  sequence_number=6207
  . . .
MySQL 5.7 – Intervals Generation

- `sequence_number` is an increasing id for each trx (not GTID) (Reset to 1 at the beginning of each new binary log)
- `last_committed` is (in MySQL 5.7) the sequence number of the most recently committed transaction when the current transaction gets its last lock (Reset to 0 at the beginning of each new binary log)

```
#170206 20:08:33 ... last_committed=6201 sequence_number=6203
#170206 20:08:33 ... last_committed=6203 sequence_number=6204
#170206 20:08:33 ... last_committed=6203 sequence_number=6205
#170206 20:08:33 ... last_committed=6203 sequence_number=6206
#170206 20:08:33 ... last_committed=6205 sequence_number=6207
...```
MySQL 5.7 – Intervals Quality

- For **MariaDB 10**, the parallelism identification quality is the “Group Commit Size”
Importance of tuning in MariaDB 10
MySQL 5.7 – Intervals Quality

- For MariaDB 10, the identification quality metric is the “Group Commit Size”
- For MySQL 5.7, it is not as straightforward
- For measuring parallelism identification quality with MySQL 5.7, I came up with a metric: the Average Modified Interval Length (AMIL)

- If we prefer to think in terms of group commit size, the AMIL can be mapped to a pseudo-group commit size by multiplying the AMIL by 2 and subtracting one
  - For a group commit of size \( n \), the sum of the intervals length is \( n^2(n+1) / 2 \)

```bash
#170206 20:08:33 ... last_committed=6203  sequence_number=6204
#170206 20:08:33 ... last_committed=6203  sequence_number=6205
#170206 20:08:33 ... last_committed=6203  sequence_number=6206
```
MySQL 5.7 – Intervals Quality

- For MariaDB 10, the identification quality metric is the “Group Commit Size”
- For MySQL 5.7, it is not as straightforward
- For measuring parallelism identification quality with MySQL 5.7, I came up with a metric: the *Average Modified Interval Length (AMIL)*
  - If we prefer to think in terms of group commit size, the AMIL can be mapped to a *pseudo*-group commit size by multiplying the AMIL by 2 and subtracting one
    - For a group commit of size $n$, the sum of the intervals length is $n \times (n+1)/2$
    - $AMIL = (n+1)/2$ (after dividing by $n$), algebra gives us $n = AMIL \times 2 - 1$
  - This mapping gives a hint on the value needed for *slave_parallel_workers*

MySQL 5.7 – Intervals Quality

Why do we need to “modify” the interval length?

Because of a limitation in the current MTS applier which will only start trx 93136 once 93131 is completed → last_committed=93124 is modified to 93131

```
#170206 21:19:31 ... last_committed=93124 sequence_number=93131
#170206 21:19:31 ... last_committed=93131 sequence_number=93132
#170206 21:19:31 ... last_committed=93131 sequence_number=93133
#170206 21:19:31 ... last_committed=93131 sequence_number=93134
#170206 21:19:31 ... last_committed=93131 sequence_number=93135
#170206 21:19:31 ... last_committed=93124 sequence_number=93136
#170206 21:19:31 ... last_committed=93131 sequence_number=93137
#170206 21:19:31 ... last_committed=93131 sequence_number=93138
#170206 21:19:31 ... last_committed=93132 sequence_number=93139
#170206 21:19:31 ... last_committed=93138 sequence_number=93140
```
MySQL 5.7 – Intervals Quality’’

- Script to compute the Average Modified Interval Length:

```bash
file=my_binlog_index_file;
echo _first_binlog_to_analyse_ > $file;
mysqlbinlog --stop-never -R --host 127.0.0.1 $(cat $file) |
grep "^#" | grep -e last_commited -e "Rotate to" |
awk -v file=$file -F "[ \t]*|=" '$_11 == "last_commited" { 
  if (length($2) == 7) {$2 = "0" $2;}
  if ($12 < max) {$12 = max;} else {max = $12;}
  print $1, $2, $14 - $12;
$10 == "Rotate"{print $12 > file; close(file); max=0;}' |
awk -F " |:" '{my_h = $2 " " $3 " :" $4;)
NR == 1 {d=$1; h=my_h; n=0; sum=0; sum2=0;)
d != $1 || h < my_h {print d, h, n, sum, sum2; d=$1; h=my_h;}
{n++; sum += $5; sum2 += $5 * $5;}'
```

MySQL 5.7 – Intervals Quality””

- AMIL without and with tuning (delay) on four (4) Booking.com masters:
MySQL 5.7 – Intervals Quality

- Computing the AMIL needs parsing the binary logs
- This is complicated and needs to handle many special cases

- Exposing counters for computing the AMIL would be better:
  - Bug# 85965: Expose, on the master, counters for monitoring // information quality.
  - Bug# 85966: Expose, on slaves, counters for monitoring // information quality.

MySQL 8.0 – Write Set

- MySQL 8.0.1 has a new way to identify parallelism
- Instead of setting `last_committed` to “the seq. number of the most recently committed transaction when the current trx gets its last lock”…
- MySQL 8.0.1 uses “the sequence number of the last transaction that updated the same rows as the current transaction”
- To do that, MySQL 8.0 remembers which rows (tuples) are modified by each transaction: this is the `Write Set`
- Write Set are not put in the binary logs, they allow to “widen” the intervals
MySQL 8.0 – Write Set’

- MySQL 8.0.1 introduces new global variables to control Write Set:
  - \texttt{transaction\_write\_set\_extraction} = \texttt{[ OFF | XXHASH64 ]}
  - \texttt{binlog\_transaction\_dependency\_history\_size} (default to 25000)
  - \texttt{binlog\_transaction\_dependency\_tracking} = \texttt{[ COMMIT\_ORDER | WRITESET\_SESSION | WRITESET ]}

- \texttt{WRITESET\_SESSION}: no two updates from the same session can be reordered
- \texttt{WRITESET}: any transactions which write different tuples can be parallelized

- \texttt{WRITESET\_SESSION} will not work well for cnx recycling (Cnx Pools or Proxies):
  - Recycling a connection with \texttt{WRITESET\_SESSION} impedes parallelism identification
MySQL 8.0 – Write Set”

- To use Write Set on a Master:
  - `transaction_write_set_extraction = XXHASH64`
  - `binlog_transaction_dependency_tracking = [ WRITESET_SESSION | WRITESET ]`

- To use Write Set on an Intermediate Master (even single-threaded):
  - `transaction_write_set_extraction = XXHASH64`
  - `binlog_transaction_dependency_tracking = WRITESET`

- To stop using Write Set:
  - `binlog_transaction_dependency_tracking = COMMIT_ORDER`
  - `transaction_write_set_extraction = OFF`
MySQL 8.0 – Write Set”

- Result for *single-threaded* Booking.com Intermediate Master (before and after):

```plaintext
#170409  3:37:13  
[...] last_committed=6695 sequence_number=6696 

#170409  3:37:14  
[...] last_committed=6696 sequence_number=6697 

#170409  3:37:14  
[...] last_committed=6697 sequence_number=6698 

#170409  3:37:14  
[...] last_committed=6698 sequence_number=6699 

#170409  3:37:14  
[...] last_committed=6699 sequence_number=6700 

#170409  3:37:14  
[...] last_committed=6700 sequence_number=6701 

#170409  3:37:14  
[...] last_committed=6700 sequence_number=6702 

#170409  3:37:14  
[...] last_committed=6700 sequence_number=6703 

#170409  3:37:14  
[...] last_committed=6700 sequence_number=6704 

#170409  3:37:14  
[...] last_committed=6704 sequence_number=6705 

#170409  3:37:14  
[...] last_committed=6700 sequence_number=6706 
```

MySQL 8.0 – Write Set’’’’’'
MySQL 8.0 – Write Set”” ”

- AMIL on a single-threaded 8.0.1 Intermediate Master (IM) without/with Write Set:
MySQL 8.0 – Write Set

Write Set advantages:

- No need to slowdown the master
- Will work even at low concurrency on the master
- Allows to test without upgrading (works on an intermediate master)  
  (however, this sacrifices session consistency, which might give optimistic results)
- Mitigate the problem of losing parallelism via intermediate masters  
  (only with `binlog_transaction_dependency_tracking = WRITESET`)  
  (→ the best solution is still Binlog Servers)
MySQL 8.0 – Write Set”” ”” ’

- Write Set limitations:
  - Needs Row-Based-Replication on the master (or intermediate master)
  - Not working for trx updating tables without PK and trx updating tables having FK (it will fall back to COMMIT_ORDER for those transactions)
  - Barrier at each DDL (Bug#86060 for adding counters)
  - Barrier at each binary log rotation: no transactions in different binlogs can be run in //
  - With WRITESET_SESSION, does not play well with connection recycling (Could use COM_RESET_CONNECTION if Bug#86063 is fixed)

- Write Set drawbacks:
  - Slowdown the master ? Consume more RAM ?
  - New technology: not fully mastered yet and there are bugs (still 1st DMR release)
MySQL 8.0 – Write Set & Bugs

- I know of at least one case where Write Set miss a transaction dependency:
  - Bad Write Set with UNIQUE KEY on a DELETE followed by an INSERT: Bug#86078
  - This happened 7 times for 5 million trx in 1 of 7 test environments (6 other are OK)
  - Restarting the slave (START SLAVE;) resumed replication in my case

- This bug deadlock the SQL_THREAD with slave_preserve_commit_order = 1
  - Deadlock with slave_preserve_commit_order=ON with Bug#86078: Bug#86079
  - The only solution I found is to “kill -9” mysqld

- Both bugs above are not a surprise to me:
  - Parallel replication is hard and MySQL 8.0 is young
  - Many fixed bugs in MariaDB parallel repl. including MDEV-7326, 7458 and 10863
InnoDB Bugs

- Parallel Replication allows to identify old bugs
- An InnoDB race condition caused a query by Primary Key to do a full table scan:
  - Hard to notice for a “1-in-a-million” SELECT (or UPDATE/DELETE on master)
  - But obvious in replication:
    one of several slaves blocked for minutes on an UPDATE via PK
    (other slaves not blocked because did not hit the race condition)
  - Bug#82968 (fixed in 5.7.18) and Bug#82969 (still open)
    https://www.facebook.com/valerii.kravchuk/posts/1073608056064467
- What other interesting bugs will we find because of // replication?
MySQL 8.0 – Write Set vs Delay

- AMIL on Booking.com masters with delay vs Write Set on Intermediate Master:
MySQL 8.0 – Write Set vs Delay

- In some circumstances, combining delay and Write Set gives better results
  - It looks like trx reordering by delay reduces the number of conflicts in Write Set
MySQL 8.0 – Write Set Speedups

- Tests on seven (7) real Booking.com environments (different workloads):
  - A is MySQL 5.6 and 5.7 masters (3 and 4 respectively)
  - B is MySQL 8.0.1 Intermediate Master with Write Set to identify parallelism (intervals)
  - C1 and C2 are two slaves: one with SSDs and one with magnetic disks

```
+----+ +----+ +----+
| A  |    | B   |    |
+----+ +----+ +----+
|     |    |     |    |
|     |    |     |    |
|     |    |     |    |
|     |    |     |    |

|     |    |     |    |
|     |    |     |    |
|     |    |     |    |
```

|     |    |     |    |
|     |    |     |    |
|     |    |     |    |
|     |    |     |    |
|     |    |     |    |
|     |    |     |    |
|     |    |     |    |
|     |    |     |    |
MySQL 8.0 – Write Set Speedups’

- I only have preliminary results:
  - Alternating 2 minutes of single-threaded with 2 min. of MTS and stopping 30 sec.
  - Only for slaves with binary logs enabled, log-slave-updates and high durability (sync_binlog=1 and trx_commit=1)
  - Only one value of MTS: a “good” slave_parallel_workers is guessed
  - Un-tuned application (probably un-optimal workload)
  - Un-tuned mysqld (maybe unknown bottleneck)
  - Measuring only once (but with many iterations)
MySQL 8.0 – Write Set Speedups

- For each of the seven (7) environments, I am reporting the following:
  - Number of occurrences of bug (which voids the result of one iteration)
  - Number of iterations (alternate 2m/2m/30s)
  - Speedups for SSDs and magnetic disks
  - Graphs for AMIL on IM and SSDs commit rate for 3 “selected” iterations

SSDs  1.6 (50 / 0)
Disks  1.1 (50 / 0)
Nb Workers  4
MySQL 8.0 – Write Set Speedups””

SSDs  5.8 (20 / 0)
Disks  3.8 (20 / 0)
Nb Workers  16

SSDs  2.9 (15 / 0)
Disks  2.5 (15 / 0)
Nb Workers  8
MySQL 8.0 – Write Set Speedups

SSDs 4.1 (15 / 0)
Disks 2.5 (50 / 0)
Nb Workers 32

SSDs 3.9 (10 / 3)
Disks 2.4 (15 / 3)
Nb Workers 16
MySQL 8.0 – Write Set Speedups

SSDs 6.9 (15 / 0)
Disks 3.5 (50 / 0)
Nb Workers 32

SSDs 1.8 (15 / 0)
Disks 1.4 (50 / 0)
Nb Workers 4
MySQL 8.0 – Write Set Speedups

- **Summary for SSDs:**
  - Two (2) “interesting” speedups: 1.6, 1.8
  - Three (3) very good speedups: 2.9, 3.9, 4.1
  - Two (2) **great** speedups: 5.8, 6.9

- **Summary for Disks:**
  - Two (2) “small” speedups: 1.1, 1.4
  - Three (3) good speedups: 2.4, 2.5, 2.5
  - Two (2) very good speedups: 3.5, 3.8

- All that without tuning MySQL or the application
- But we need to do more rigorous benchmarks
Write Set in Group Replication (5.7)

- Write Set is used in MySQL 5.7 for Group Replication (GR):
  - Write Set is part of the certification process (conflict detection)
  - Once accepting commit, Write Set is used to do parallel remote query execution

- Parallel remote query execution with Write Set explains why a MySQL 5.7 GR node can apply trx “faster” than an asynchronous slave

- With MySQL 8.0.1, an asynchronous slave should be as fast as GR
Parallel replication is not simple

MariaDB 10.0 in-order (and probably MySQL 5.7 logical clock) has limitations:
- Long transactions block the parallel replication pipeline
- Intermediate master loses parallelism and reduce replication speed on slaves

MySQL 5.6 and 5.7 are not fully MTS crash-safe (without GTIDs)

MariaDB out-of-order needs careful and precise developer involvement

MySQL schema-based solution looks safer and simpler to use than MariaDB out-of-order which is more flexible but more complex

MariaDB 10.1 aggressive mode much better than conservative

Try very high number of threads

In all cases, avoid big transactions in the binary logs
MySQL 5.7 and 8.0 // Repl. Summary

- Parallel replication is still not simple
  (maybe even more complicated in 5.7 and 8.0: intervals, tuning, AMIL, …)

- Write Set in MySQL 8.0.1 very promising:
  - Some great speedups and most of them good
  - But more rigorous test needs to be done
  - Some feature requests and bugs (I am looking forward to the next version)
  - Evolving understanding of the technology: expect new things

- Future work:
  - A better replication applier for slaves (no barrier on 1st dependency)
  - Slave Group Commit (is it useful ?)
  - Optimistic parallel replication (is it better than Write Set ?)
And please test by yourself and share results
Replication: Links

- Replication crash safety with MTS in MySQL 5.6 and 5.7: reality or illusion?

- A Metric for Tuning Parallel Replication in MySQL 5.7

- Solving MySQL Replication Lag with LOGICAL_CLOCK and Calibrated Delay

- How to Fix a Lagging MySQL Replication
  [https://thoughts.t37.netfixing-a-very-lagging-mysql-replication-db6eb5a6e15d](https://thoughts.t37.netfixing-a-very-lagging-mysql-replication-db6eb5a6e15d)

- Binlog Servers:
  - [http://blog.booking.com/mysql_slave_scaling_and_more.html](http://blog.booking.com/mysql_slave_scaling_and_more.html)
  - [http://blog.booking.com/abstracting_binlog_servers_and_mysql_master_promotion_wo_reconfiguring_slaves.html](http://blog.booking.com/abstracting_binlog_servers_and_mysql_master_promotion_wo_reconfiguring_slaves.html)
// Replication: Links’

- Bugs/feature requests:
  - The doc. of slave-parallel-type=LOGICAL_CLOCK wrongly reference Group Commit: Bug#85977
  - Allow slave_preserve_commit_order without log-slave-updates: Bug#75396
  - MTS with slave_preserve_commit_order not repl. crash safe: Bug#80103
  - Automatic Repl. Recovery Does Not Handle Lost Relay Log Events: Bug#81840
  - Expose, on the master/slave, counters for monitoring // info. quality: Bug#85965 & Bug#85966
  - Expose counters for monitoring Write Set barriers: Bug#86060
  - The function reset_connection does not reset Write Set in WRITESET_SESSION: Bug#86063
  - Bad Write Set tracking with UNIQUE KEY on a DELETE followed by an INSERT: Bug#86078
  - Deadlock with slave_preserve_commit_order=ON with Bug#86078: Bug#86079

- Fixed bugs:
  - Message after MTS crash misleading: Bug#80102 (and Bug#77496)
  - Replication position lost after crash on MTS configured slave: Bug#77496
  - Full table scan bug in InnoDB: MDEV-10649, Bug#82968 and Bug#82969
Thanks

Jean-François Gagné
jeanfrancois DOT gagne AT booking.com