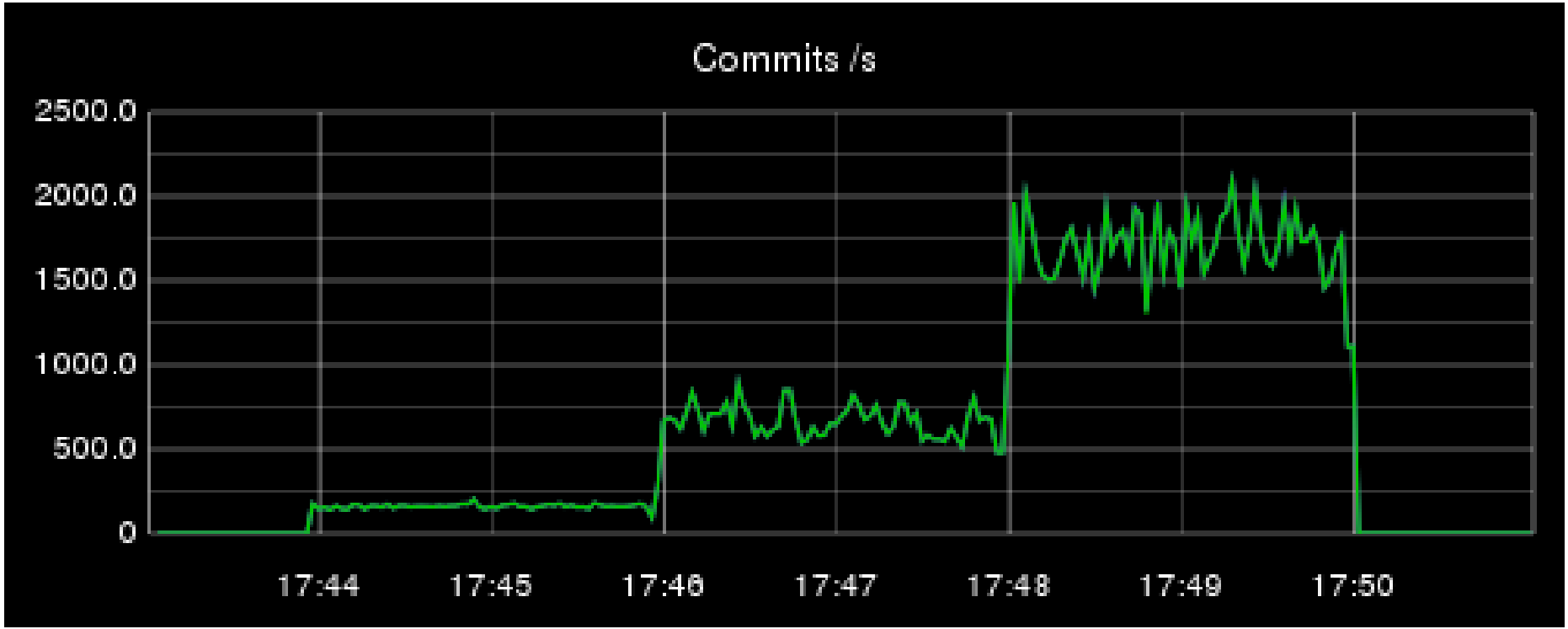
Booking.com

MySQL/MariaDB Parallel Replication: inventory, use-cases and limitations

Jean-François Gagné (System Engineer)
jeanfrancois DOT gagne AT booking.com
April 26, 2017 – Percona Live Santa Clara 2017


Commits /s
2500.0
2000.0
1500.0
1000.0
500.0
0
17:44
17:45
17:46
17:47
17:48
17:49
17:50

Booking.com

- 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

Session Summary

1. Introducing Parallel Replication (*// Replication*)
2. MySQL 5.6: schema based
   MariaDB 10.0: Out-of-Order and In-Order
   MariaDB 10.1: MariaDB 10.0 + Optimistic
   MySQL 5.7: MySQL 5.6 + Logical Clock
   MySQL 8.0 & Group Rpl: MySQL 5.7 + Write Set
3. Benchmark Results from Booking.com

// 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 → we can do 2 Read IOs in parallel)
    (SSDs can serve many Read and/or Write IOs in parallel)

// Replication: MySQL 5.6

- **Concept**: if transactions are “schema-local”,
  two transactions in different schema can be run in parallel on slaves
- **Implementation**:
  - the master tags transactions with their schema in the binary logs
  - the SQL thread dispatches work to worker threads according to the schema from the binlog
- **Deployment**:
  - On the master: nothing to do (except having multiple independent schemas)
  - On the slave: “`SET GLOBAL slave_parallel_workers = N;`” (with N > 1)
- MySQL 5.7 has the same feature (default for `slave-parallel-type = DATABASE`)
- MySQL 8.0 defaults might be different:
  - Would need to “`SET GLOBAL slave-parallel-type = DATABASE;`”
    http://mysqlhighavailability.com/mysql-replication-defaults-after-5-7/

// Replication: MySQL 5.6’

- **Implication**: transactions on slaves can be committed in a different order than the order they appear in the binary logs of the master
- On the master, some transactions in schema A and B:
  - Order in the binary logs of the master: `A1, A2, B1, B2, A3, B3`
- On the slave, transactions in different schema are run in parallel:
  - “`A1, A2, A3`” run in parallel with “`B1, B2, B3`”
  - One possible commit order: `A1, B1, A2, B2, A3, B3`
  - Another if `B1` is long to execute : `A1, A2, A3, B1, B2, B3`
  - Many other possible orders…
- Out-of-order commit on slave has many impacts…

// Replication: MySQL 5.6”

- Impacts on the binary log content on slaves:
  - 2 slaves can have different binlogs (also different from the master binlogs)
- Impacts on “`SHOW SLAVE STATUS`”:
  - All transactions before the reported SQL thread file and position are committed
  - This “all committed before” position is called a checkpoint
  - Some trx might be committed after the checkpoint and some might still be running → gaps
- Impacts on replication crash recovery (because gaps)
- Impacts on GTIDs: temporary holes in @@global.gtid_executed (because gaps)
- More impacts: skipping transactions, backups, heartbeat, …

// Replication: MySQL 5.6”’

- Removing gaps in transaction execution:
  - “`STOP SLAVE; START SLAVE UNTIL SQL_AFTER_MTS_GAPS;`”
- MySQL is not parallel replication crash safe without GTIDs (this is a bug):
  - http://jfg-mysql.blogspot.com/2016/01/replication-crash-safety-with-mts.html
- For skipping transactions (with `sql_slave_skip_counter`): first remove gaps
- For backups: make sure your tool is parallel replication aware
- Worker states stored in mysql.slave_worker_info:
  - https://dev.mysql.com/worklog/task/?id=5599 (not an easy read)
- Tuning parameters:
  - slave-pending-jobs-size-max: RAM for unprocessed events (default 16M)
  - slave_checkpoint_group: see next slide (default 512)
  - slave_checkpoint_period: see next slide (default 300 ms)

Booking.com

// Replication: MySQL 5.6”’ ’

- MTS checkpoint:
  - After making sure gaps are filled, checkpointing advances the position of “`SHOW SLAVE STATUS`”
- Checkpointing is tried every *slave_checkpoint_period* (300 ms by default)
- A checkpoint attempt might fail if a worker is still working on the next needed transaction → long transaction might block checkpointing:
  - Binlog content: A1,A2,B1,B2,B3,B4,B5…B500,B501,…B600
  - If A2 is very long (ALTER TABLE), it will block checkpointing
  - This will block the slave execution at ~B511
- If this happens, workers will not be able to go beyond the group size
  - Solution: increase slave_checkpoint_group (512 by default)
- Similar problems happen if transactions are big (in the binlogs)
  - Solution: increase slave-pending-jobs-size-max (16M by default)
  - But try keeping your trx small (avoid LOAD DATA INFILE and others…)

Booking.com

// Replication: MariaDB 10.0 (out-of-order)

- **Concept**: manually tags independent transactions in "*write domains*"
- **Implementation**:
  - MariaDB GTIDs: `<domain ID>-<server ID>-<Sequence Number> (0-1-10)`
  - the SQL thread becomes a coordinator that dispatches work
- **Deployment**:
  - On the master and for each trx: "`SET SESSION gtid_domain_id = D;`"
  - On the slave: "`SET GLOBAL slave_parallel_threads = N;`" (with N > 1)
- But advertise the Write Domain **right** !
  - MySQL protects you from multi-schema trx, MariaDB cannot do the same for write domains
- Also out-of-order commits of transactions on slaves:
  - There will be gaps, those gaps are managed by MariaDB GTIDs,
  - Impact on binary logs, SHOW SLAVE STATUS, skipping transactions, backups, heartbeat, …

// Replication: MariaDB 10.0 (out-of-order)’

- Difference with MySQL 5.6:
  - “`SHOW SLAVE STATUS`”: position of the latest committed trx (there might be gaps before…)
  - If the SQL thread stops (or is stopped), its position will “rewind” to a “safe” position https://jira.mariadb.org/browse/MDEV-6589 & MDEV-9138 & MDEV-10863 (all fixed bugs)
- Removing gaps: `STOP SLAVE; SET GLOBAL slave_parallel_threads = 0; START SLAVE;`
  - To avoid re-downloading relay logs, use below but see two MDEVs above:
    `STOP SLAVE SQL_THREAD; SET GLOBAL slave_parallel_threads = 0; START SLAVE;`
    http://jfg-mysql.blogspot.com/2015/10/bad-commands-with-mariadb-gtids-2.html
- Skipping transactions:
  - Go back to single threaded replication, START SLAVE → break again, then skip
  - Like above, restart the IO thread if you want to avoid problems

// Replication: MariaDB 10.0 (in-order)

- **Concept**:
  trx committing together on the master can be executed in parallel on slaves
- **Implementation**:
  - Build on top of the binary log *Group Commit* optimization:
    the master tags transactions in the binary logs with their Commit ID (*cid*)
  - As the name implies, trx are committed in the same order as in the binlogs of the master
- **Deployment**:
  - Needs a MariaDB 10.0 master
  - On slaves: "`SET GLOBAL slave_parallel_threads = N;`" (with N > 0)

// Replication: MariaDB 10.0 (in-order)’

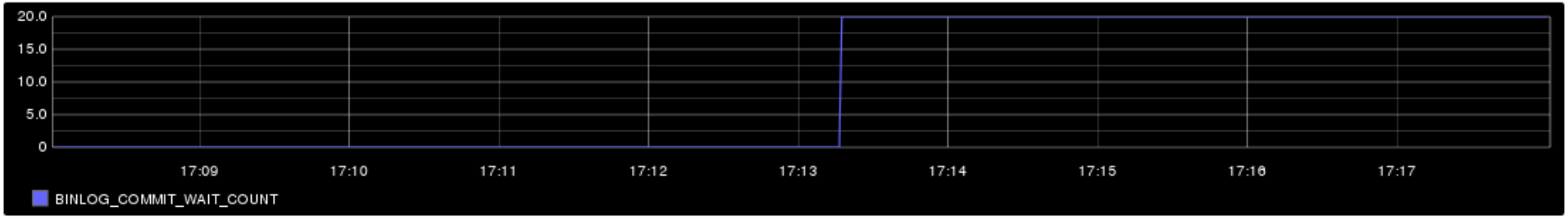
- Binary logs extract:

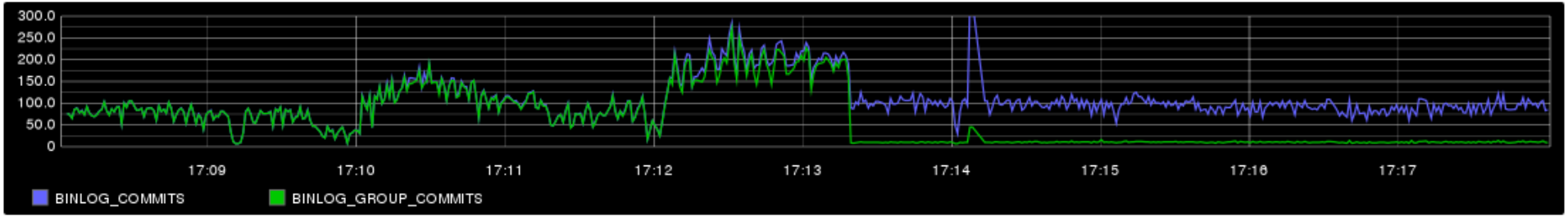
```
...
#150316 11:33:46 ... GTID 0-1-184 cid=2324
#150316 11:33:46 ... GTID 0-1-185 cid=2335
...
#150316 11:33:46 ... GTID 0-1-189 cid=2335
#150316 11:33:46 ... GTID 0-1-190
#150316 11:33:46 ... GTID 0-1-191 cid=2346
...
#150316 11:33:46 ... GTID 0-1-197 cid=2346
#150316 11:33:46 ... GTID 0-1-198 cid=2361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

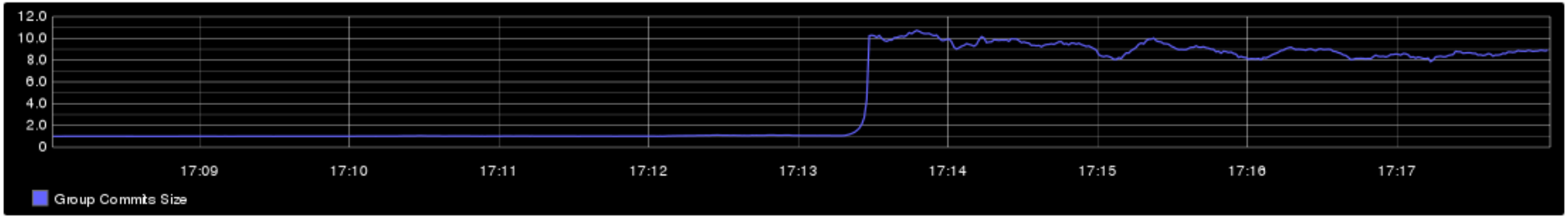
// Replication: MariaDB 10.0 (in-order)”

- Good (large groups) or bad (small groups) parallelism from the master:
  - When `sync_binlog = 1`, instead of syncing the binlog after each transaction, MariaDB buffers trx during previous sync before writing all of them as a group and then syncing
  - Setting `sync_binlog = 0` or `> 1` might lead to smaller groups (bad for parallel replication)
  - When there is not enough parallelism, or if sync are very fast, grouping might also be suboptimal
- Global Statuses can be used to monitor grouping on the master:
  - BINLOG_COMMITS: number of commits in the binary logs
  - BINLOG_GROUP_COMMITS: number of group commits in the binary logs (lower is better)
  - The first divided by the second gives the group size (larger is better)
- Grouping optimization (slowing down the master to speedup slaves):
  - `BINLOG_COMMIT_WAIT_USEC` (*BCWU*): timeout for waiting more transactions joining the group
  - `BINLOG_COMMIT_WAIT_COUNT` (*BCWC*): number of transactions that short-circuit waiting

// Replication: MariaDB 10.0 (in-order)”’







// Replication: MariaDB 10.0 (in-order)”’ ’

- Long transactions can block the parallel execution pipeline
- On the master: `---------- Time ---------->`

```
T1:  B------------------------C
T2:                          B--C
T3:                          B--C
```

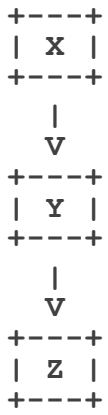
- On the slaves:

```
T1:  B------------------------C
T2:  B-- . . . . . . . . . . C
T3:  B-- . . . . . . . . . . C
```

- Try reducing as much as possible the number of big transactions:
  - Easier said than done: 10 ms is big compared to 1 ms
- Avoid monster transactions (LOAD DATA, unbounded UPDATE or DELETE, …)

// Replication: MariaDB 10.0 (in-order)”’ ”

- Replicating through intermediate masters (*IM*) loses grouping
- Four transactions on X, Y and Z:



On X:

```
    ----Time---->
T1        B---C
T2        B---C
T3    B-------C
T4    B-------C
```

On Y:

```
    ----Time---->
T1      B---C
T2      B---C
T3    B-------C
T4    B-------C
```

On Z:

```
    ----Time--------->
T1    B---C
T2    B---C
T3          B-------C
T4          B-------C
```

- To get maximum replication speed, replace intermediate master by Binlog Servers
- More details at http://blog.booking.com/better_parallel_replication_for_mysql.html

// Replication: Slave Group Commit

- On a single-threaded slave, transactions are run sequentially:

```
-------- Time -------->
  T1:  B----C
  T2:       B----C
```

- If T1 and T2 are in different cid, they cannot be run in parallel
- But if they do not conflict, delaying committing of T1 might allow to completely run T2 in another thread, achieving group commit:

```
  T1:  B---- . . C     (in thread #1)
  T2:       B----C     (in thread #2)
```

- Above has identified that T1 and T2 can be run in parallel (and saved a fsync)

// Replication: Slave Group Commit'

- MariaDB 10.0 implements Slave Group Commit when
  1. the master is running MariaDB 10.0,
  2. `slave_parallel_threads (`*`SPT`*`) > 1 & BCWC > 1 & BCWU > 0`

- Waiting is short-circuited when a transaction Tn blocks on Tn-i
  so this should not happen:

```
T1:  B---- . . . . C
T2:       B--- . . . --C
```

- ➢ No penalty for using big value of BCWU on slaves
  - This mitigates the problem with intermediate masters
  - Except for DDL where short-circuit is not implemented

// Replication: MariaDB 10.1 (in-order)

- MariaDB 10.1 has five different slave parallel modes:
  1. `none`: classic single-threaded slave (same as `slave_parallel_threads = 0`)
  2. `minimal`: in different threads, serialized execution of transaction
     (this is for slave group commit: needs `BCWC > 1` and `BCWU > 0`)
     (and out-of-order parallel replication disabled in this mode)
  3. `conservative`: parallel execution based on group commit (= MariaD 10.0)
  4. `optimistic`: a new type of parallel execution
  5. `aggressive`: a more aggressive optimistic mode

// Replication: Back to MariaDB 10.0

- With MariaDB 10.0, naïve parallel replication could deadlock
- On the master, T1 and T2 commit together:

```
T1:  B-------C
T2:        B--C
```

- On the slaves, T2 (ready to commit) blocks T1 (because index update, …), but T1 must commit before T2 → deadlock

```
T1:  B---- . . . . . . . . . . . . .
T2:  B-- . . . . . . . . . . . . . .
```

- To solve this deadlock, MariaDB kills T2, which unblocks T1
- Corresponding global status: `slave_retried_transactions`

// Replication: Back to MariaDB 10.0’

- Number of retried transactions catching up many hours of replication delay (~2.5K transactions per second):

- Retry happen 3 times for 600,000 transactions → not often at all

// Replication: MariaDB 10.1 (optimistic)

- **Concept**: run all transactions in parallel, if they conflict (replication blocked because in-order commit), deadlock detection unblocks the slave
- **Deployment**:
  - Needs a MariaDB 10.1 master
  - Assume same table transactional property on master and slave
    (could produce corrupted results if the master is InnoDB and slave MyISAM)
  - `SET GLOBAL slave_parallel_thread = N;` (with N > 1)
  - `SET GLOBAL slave_parallel_mode = {optimistic | aggressive};`
    Optimistic will try to reduce the number of deadlocks (and rollbacks) using information put in the binary logs from the master, aggressive will run as many transactions in parallel as possible (bounded by the number of threads)
- DDLs cannot be rollbacks → they cannot be replicated optimistically:
  - DDL blocks the parallel replication pipeline (and same for other non-transactional operations)

// Replication: MySQL 5.7

- 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 10.0:
  - Problems with long/big transactions
  - Problems with intermediate masters

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
    http://jfg-mysql.blogspot.com/2016/01/replication-crash-safety-with-mts.html
  - 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
  - And it is still not replication crash safe (surprising because no gap): Bug#80103 & Bug#81840

// Replication: MySQL 8.0.1 and GR

- Group Replication (*GR*) and MySQL 8.0.1 adds Write Set
- Write Set are a new way to identify parallelism on masters or even on slaves
  - allows a node in Group Replication to replicate faster than a “standard” slave
  - not limited by “Group Commit Size” (no need for transaction to “commit together”)
  - works on a single-threaded workload
- Write Set can be used to identify parallelism on a single-threaded intermediate master, which allows to do test without upgrading the master

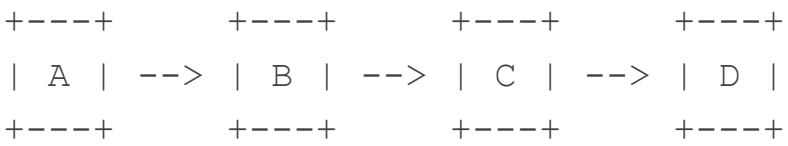
I cannot explain how this works in this talk (not enough time)
come to my talk tomorrow for all the details
MySQL Parallel Replication: all the 5.7 (and some of the 8.0) details

// Replication: results from B.com

- MariaDB 10.0 tests:
  - On 4 environments, from a MySQL 5.6 masters, thanks to Slave Group Commit
- MariaDB 10.1 tests: conservative vs optimistic (aggressive)
  - Same environments and trx: how much better (or worse) will aggressive be ?
- MySQL 5.6 real production deployment (schema based)
- MariaDB 10.0 and 10.1 real production deployment
- No results from MySQL 5.7 in this talk (more in another talk)

// Replication: MariaDB 10.0

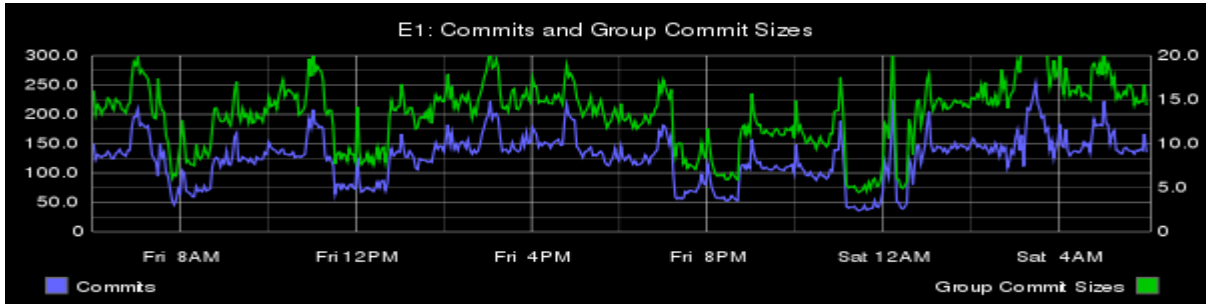
- Four environments (E1, E2, E3 and E4):
  - A is a MySQL 5.6 master
  - B is a MariaDB 10.0 intermediate master
  - C is a MariaDB 10.0 intermediate master doing slave group commit
  - D is using the group commit information from C to run transaction in parallel

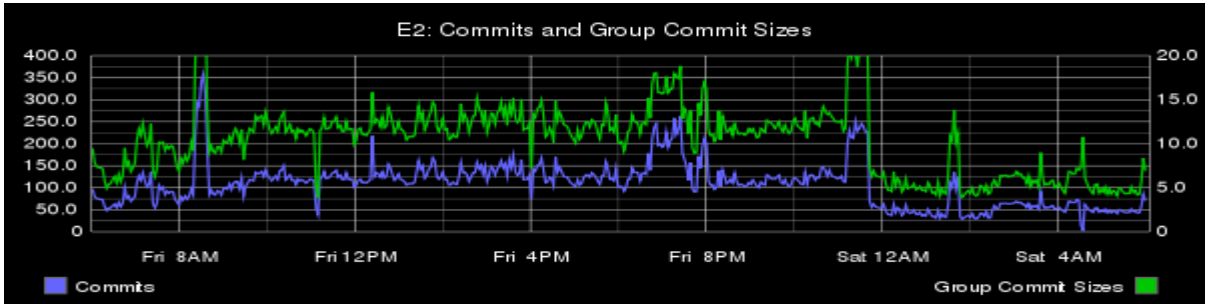


- Note that slave group commit generates smaller group than a group committing master, more information in: http://blog.booking.com/evaluating_mysql_parallel_replication_3-under_the_hood.html#group_commit_slave_vs_master

// Replication: MariaDB 10.0 tests

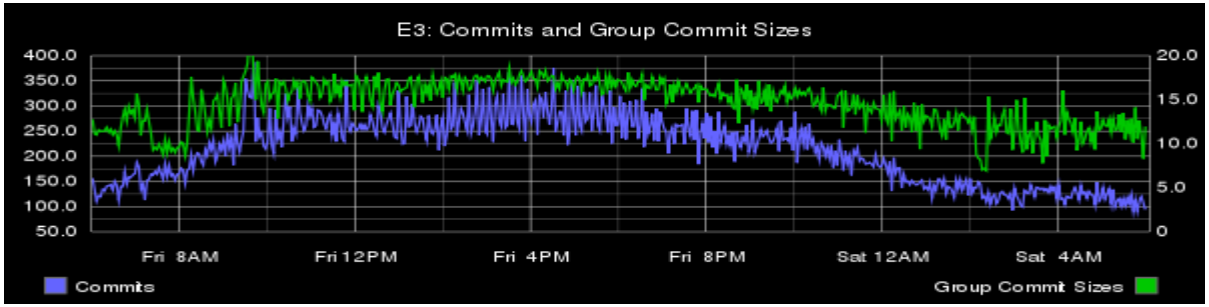
- Parallel replication with MariaDB 10.0 (or with 10.1 conservative):
  - Catching up 24 hours of replication delay with 0, 5, 10, 20 and 40 threads

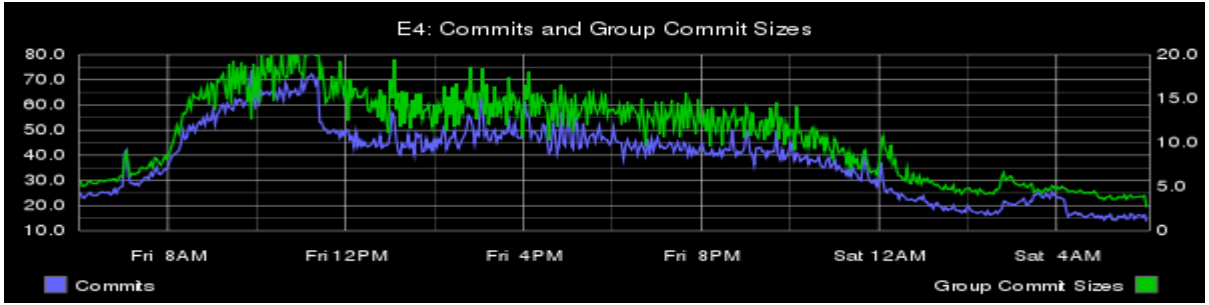




Booking.com

// Replication: MariaDB 10.0 tests’





More details at:

http://blog.booking.com/evaluating_mysql_parallel_replication_3-benchmarks_in_production.html

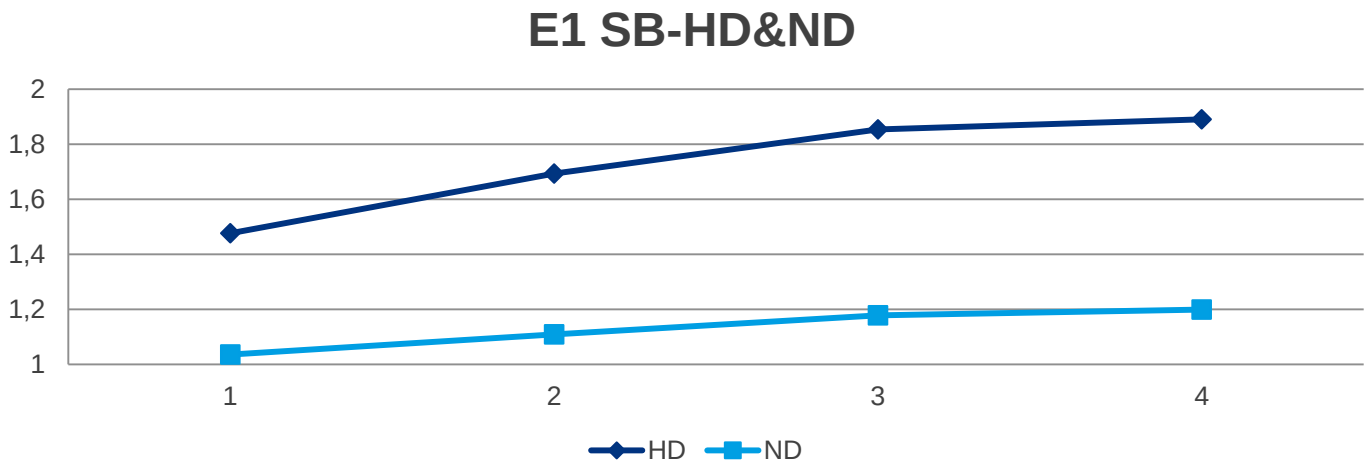
Booking.com

// Replication: MariaDB 10.0 tests”

Slave with binlogs (*SB*) but without log-slave-updates

High Durability (*HD*): “`sync_binlog = 1`” + “`trx_commit = 1`”

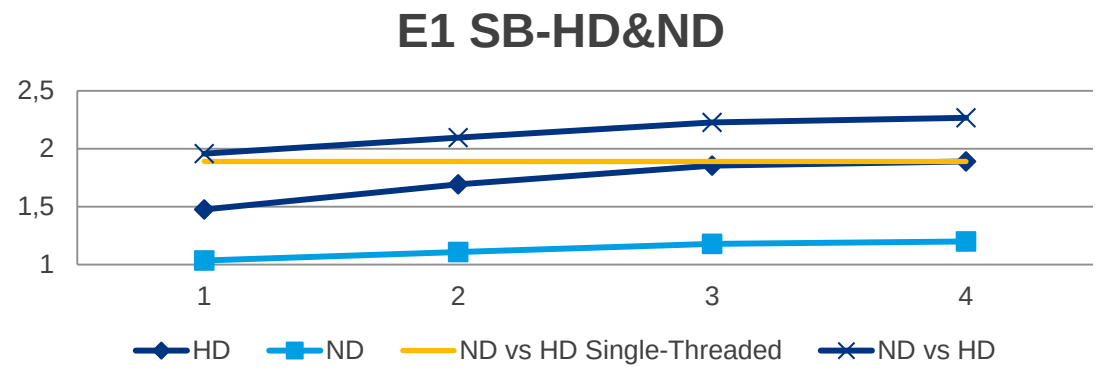
No Durability (*ND*): “`sync_binlog = 0`” + “`trx_commit = 2`”

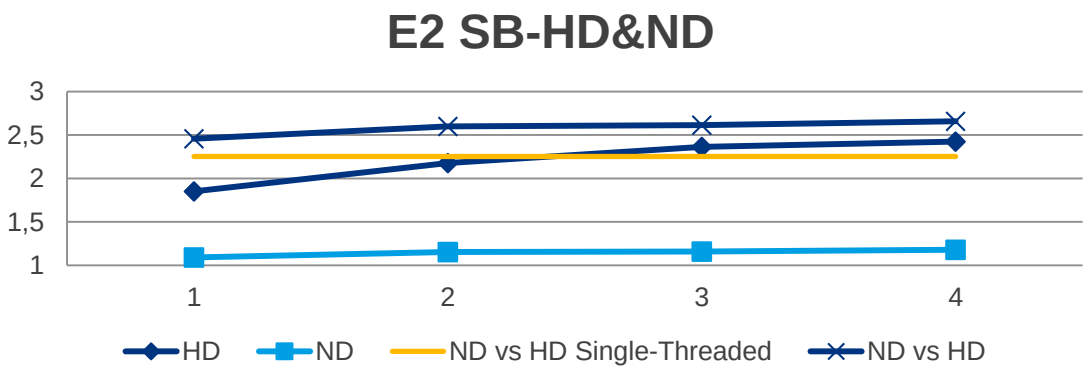


HD Single-Threaded: 3h09.34

ND Single-Threaded: 1h24.09

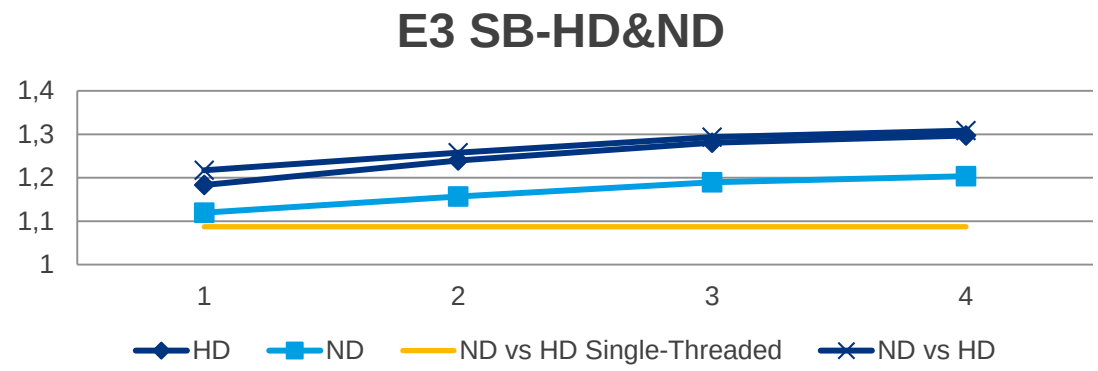
// Replication: MariaDB 10.0 tests”’

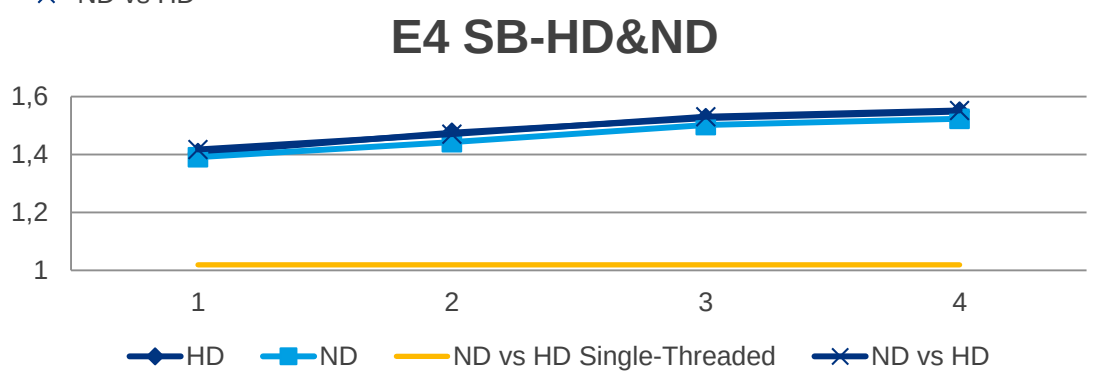




Booking.com

// Replication: MariaDB 10.0 tests”’ ’

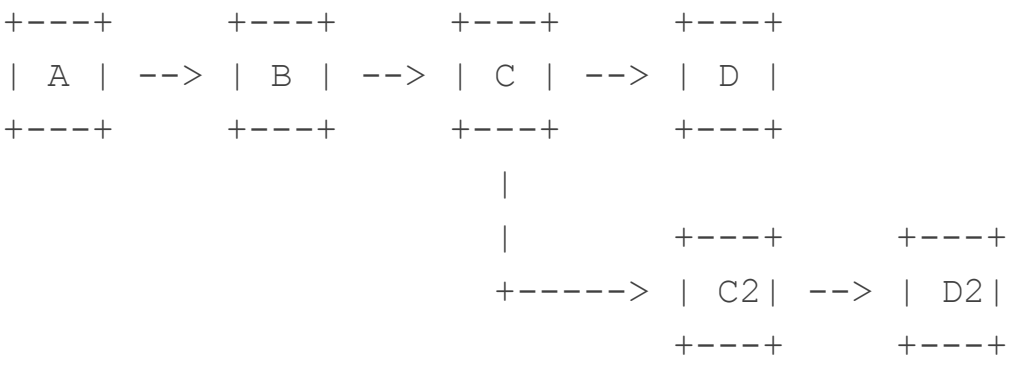




Booking.com

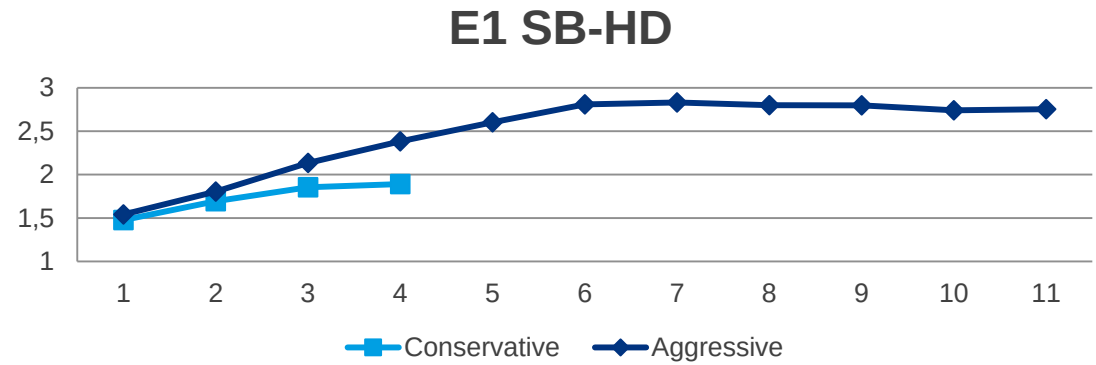
// Replication: MariaDB 10.1 tests

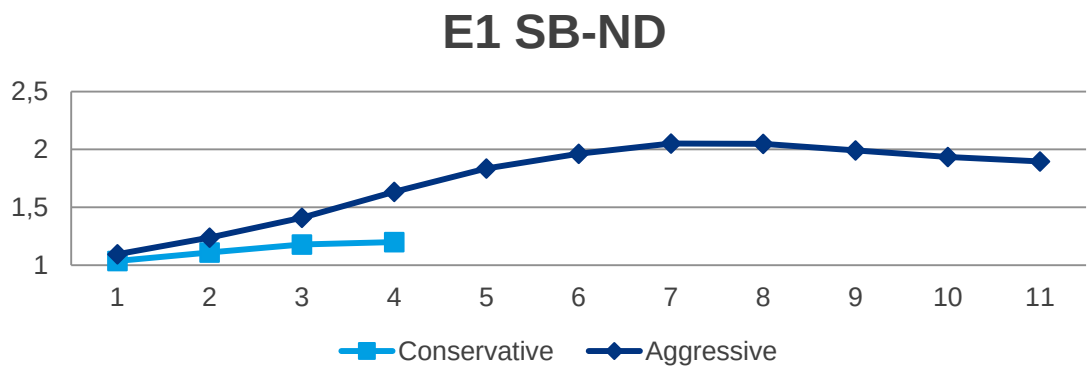
- Four same environments, D now runs MariaDB 10.1, and to take advantage of optimistic parallel replication, we need a 10.1 master → add C2



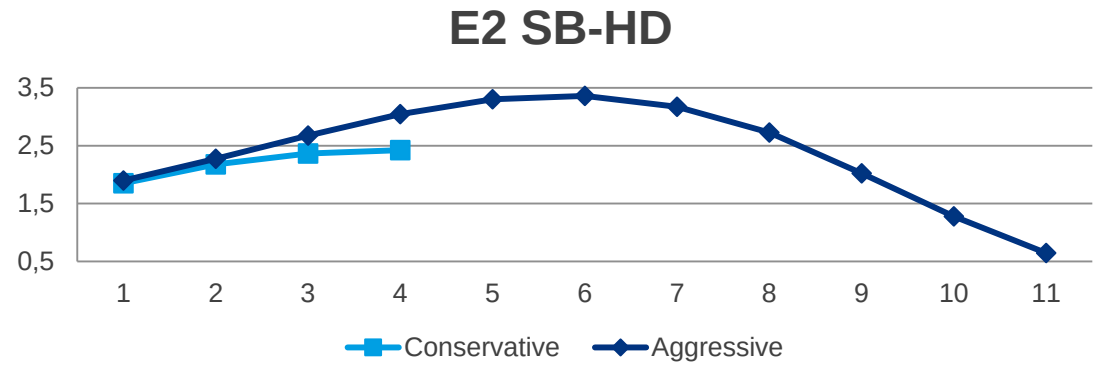
- D and D2 are the same hardware
- D runs with SPT = conservative (using the “slave Group Commit” binary logs)
- D2 runs with SPT = aggressive (needs a 10.1 master to work)

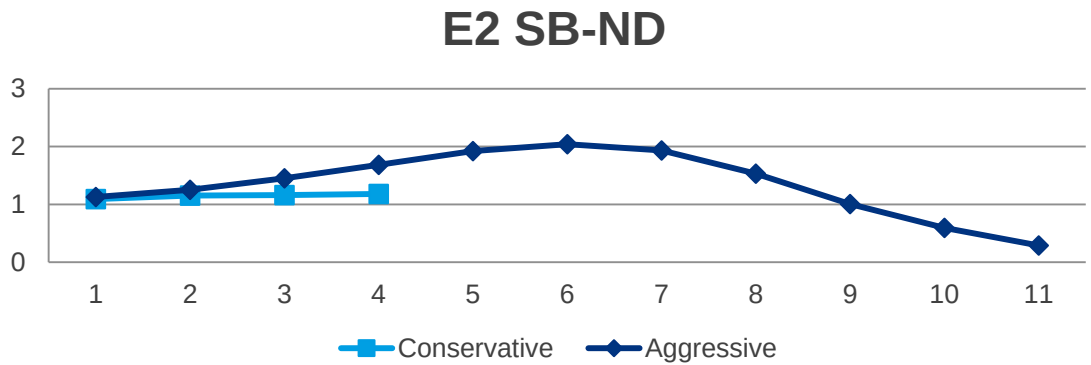
// Replication: MariaDB 10.1 tests’



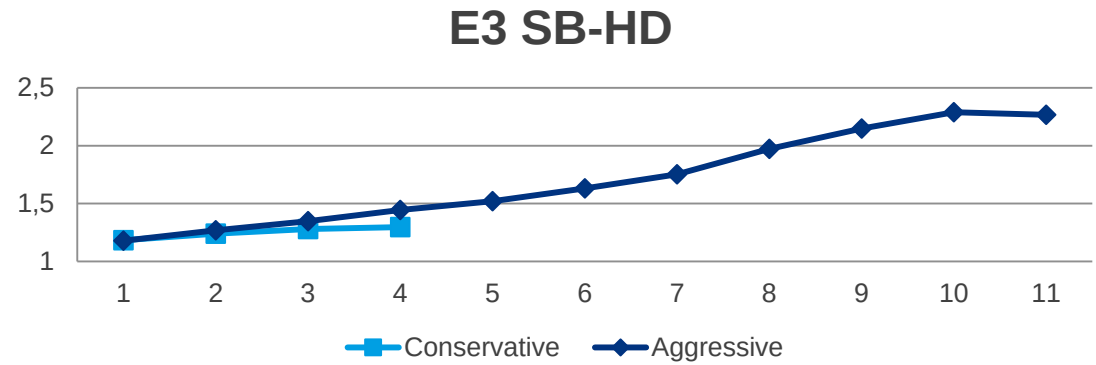


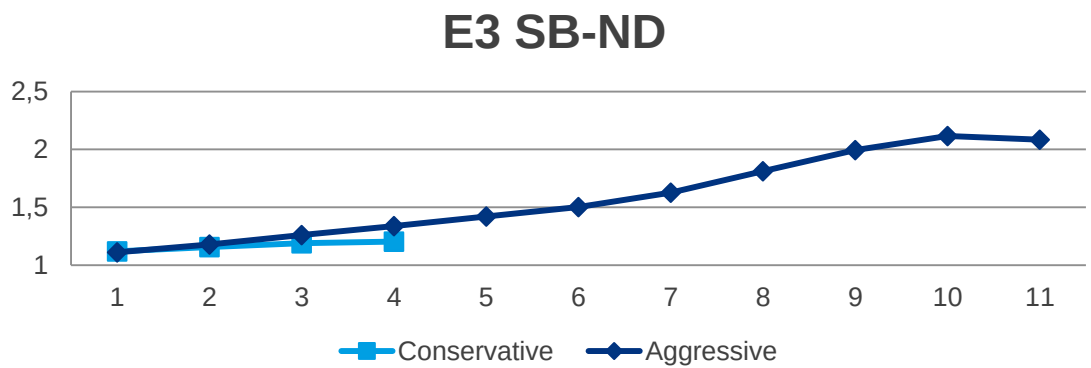
// Replication: MariaDB 10.1 tests”



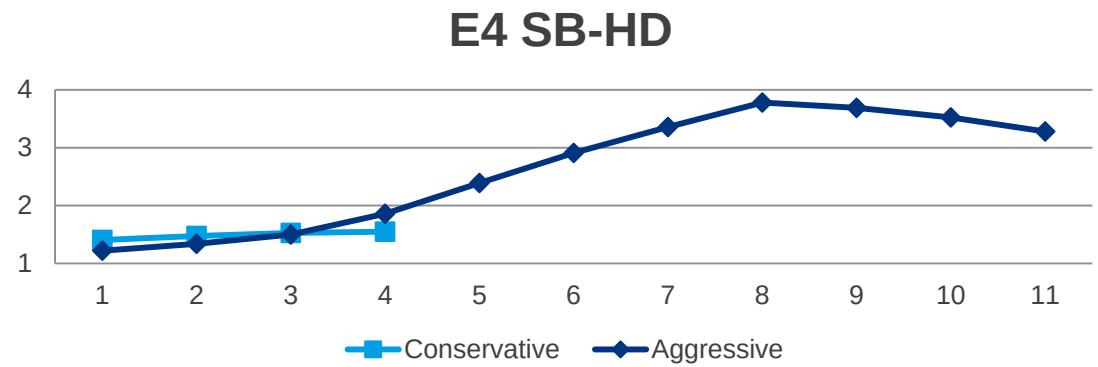


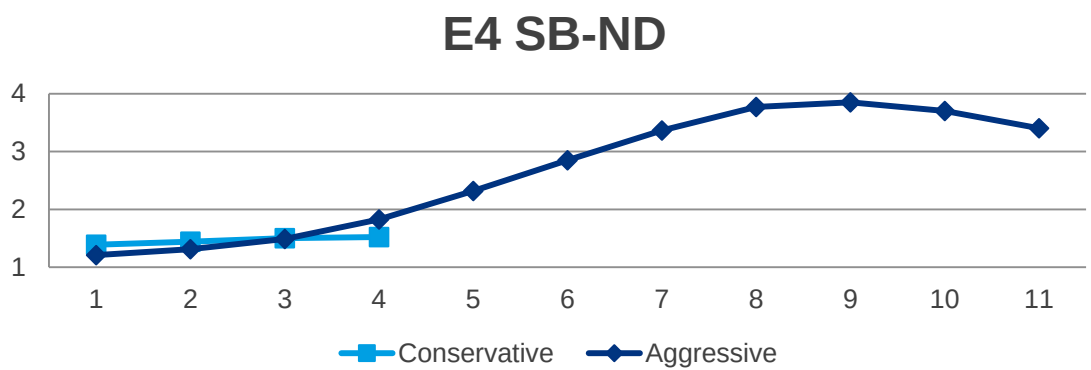
// Replication: MariaDB 10.1 tests”’





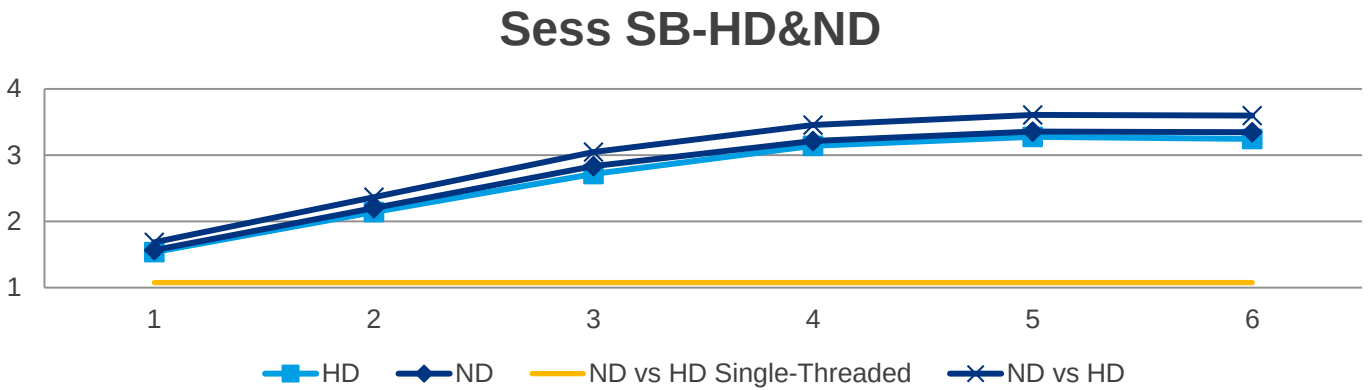
// Replication: MariaDB 10.1 tests”’ ’





// Replication: MySQL 5.6 real

- Booking.com session store is sharded with many schema per database:
  - Running MySQL 5.6, >1 TB per node, 20 schema per node, magnetic disks
  - PLAMS 2015: Combining Redis and MySQL to store HTTP cookie data https://www.percona.com/live/europe-amsterdam-2015/sessions/combining-redis-and-mysql-store-http-cookie-data



// Replication: MariaDB 10.0 real

- Booking.com is also running MariaDB 10.0:
  - Test is run on a multi-tenths of TB database
  - Hosted on a disk storage array: low write latency (cache) and many reads in parallel
- We optimized group commit and enabled parallel replication:

```
set global BINLOG_COMMIT_WAIT_COUNT = 75;
set global BINLOG_COMMIT_WAIT_USEC = 300000;
set global SLAVE_PARALLEL_THREADS = 80;
```

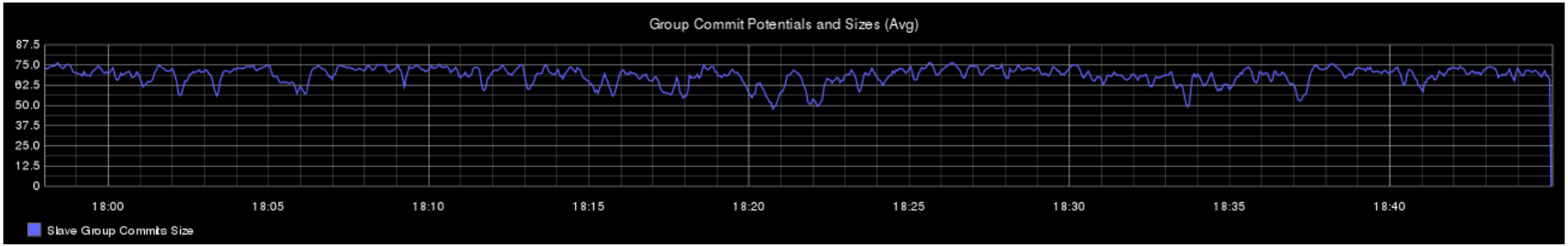
(300 milliseconds)

- And we tuned the application:
  - Break big transaction in many small transactions
  - Increase the number of concurrent sessions to the database

// Replication: MariaDB 10.0 real’

- Commit rate when disabling/enabling parallel replication:

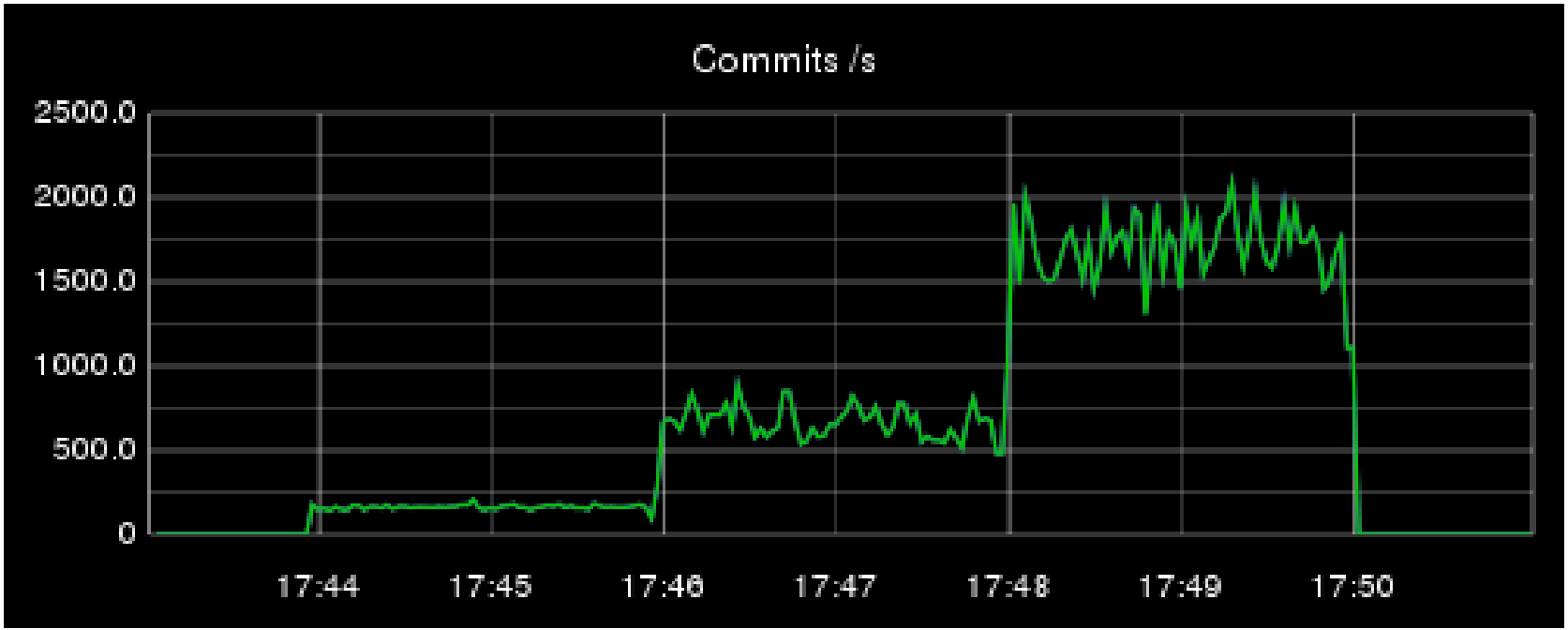
- Corresponding group commit size:



(Time is not the same on X-Axis as the slave was delayed on purpose)

// Replication: MariaDB 10.1 real

- Single-threaded, conservative and aggressive commit rate:



Kudos to MariaDB Engineers

- Parallel Replication allows to identify old bugs:
  - Queries at the same time on the same table is unlikely IRL but common with // replication
- 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 in replication, this is obvious: one of many slaves blocked for minutes on an UPDATE by PK
- Kudos to MariaDB Engineers for finding and fixing this very hard bug:
  - Valerii Kravchuk spotted a strange behavior in `SHOW ENGINE INNODB STATUS` output and narrowed it down by rightfully asking for a `SHOW EXPLAIN FOR` on the “slow” UPDATE
  - Michael Widenius (Monty) provided me with many patches to identify the problem (debug in prod.)
  - Sergey Petrunya fixed the bug in InnoDB and reported it upstream (MDEV-10649 fixed in MariaDB 10.0.28 and 10.1.18) (Bug#82968 fixed in 5.7.18 and Bug#82969 still open) (more context in https://www.facebook.com/valerii.kravchuk/posts/1073608056064467)

// Replication: Summary

- 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

Booking.com

And please
test by yourself
and share results

// Replication: Links

- Evaluating MySQL Parallel Replication Part 4: More Benchmarks in Production: http://blog.booking.com/evaluating_mysql_parallel_replication_4-more_benchmarks_in_production.html (see also Part 3, 2 and 1 that are linked in the post)
- Replication crash safety with MTS in MySQL 5.6 and 5.7: reality or illusion? https://jfg-mysql.blogspot.com/2016/01/replication-crash-safety-with-mts.html
- Binlog Servers:
  - http://blog.booking.com/mysql_slave_scaling_and_more.html
  - Better Parallel Replication for MySQL: http://blog.booking.com/better_parallel_replication_for_mysql.html
  - http://blog.booking.com/abstracting_binlog_servers_and_mysql_master_promotion_wo_reconfiguring_slaves.html
- Others:
  - https://mariadb.com/blog/how-get-mysql-56-parallel-replication-and-percona-xtrabackup-play-nice-together
  - https://www.percona.com/blog/2015/01/29/multi-threaded-replication-with-mysql-5-6-use-gtids/

// 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
  - Automatic Replication Recovery Does Not Handle Lost Relay Log Events: Bug#81840
  - FTWRL deadlock: MDEV-10644 (workaround: kill the FTWRL session, maybe needing gdb magic)
  - (More MySQL 5.7 and 8.0 bugs in the other talk)

- Fixed bugs
  (make sure to use **MariaDB 10.0.29+ or 10.1.20+** to avoid fixed bugs):
  - Replication position lost after crash on MTS configured slave: Bug#77496
  - MariaDB deadlocks and replication breakages: MDEV-7326, MDEV-7458 and MDEV-10863
  - Full table scan bug in InnoDB: MDEV-10649, Bug#82968 and Bug#82969

Thanks

Jean-François Gagné
jeanfrancois DOT gagne AT booking.com