Booking.com

The Full MySQL and MariaDB Parallel Replication Tutorial

Eduardo Ortega (MySQL Database Engineer)
eduardo DOT ortega AT booking.com

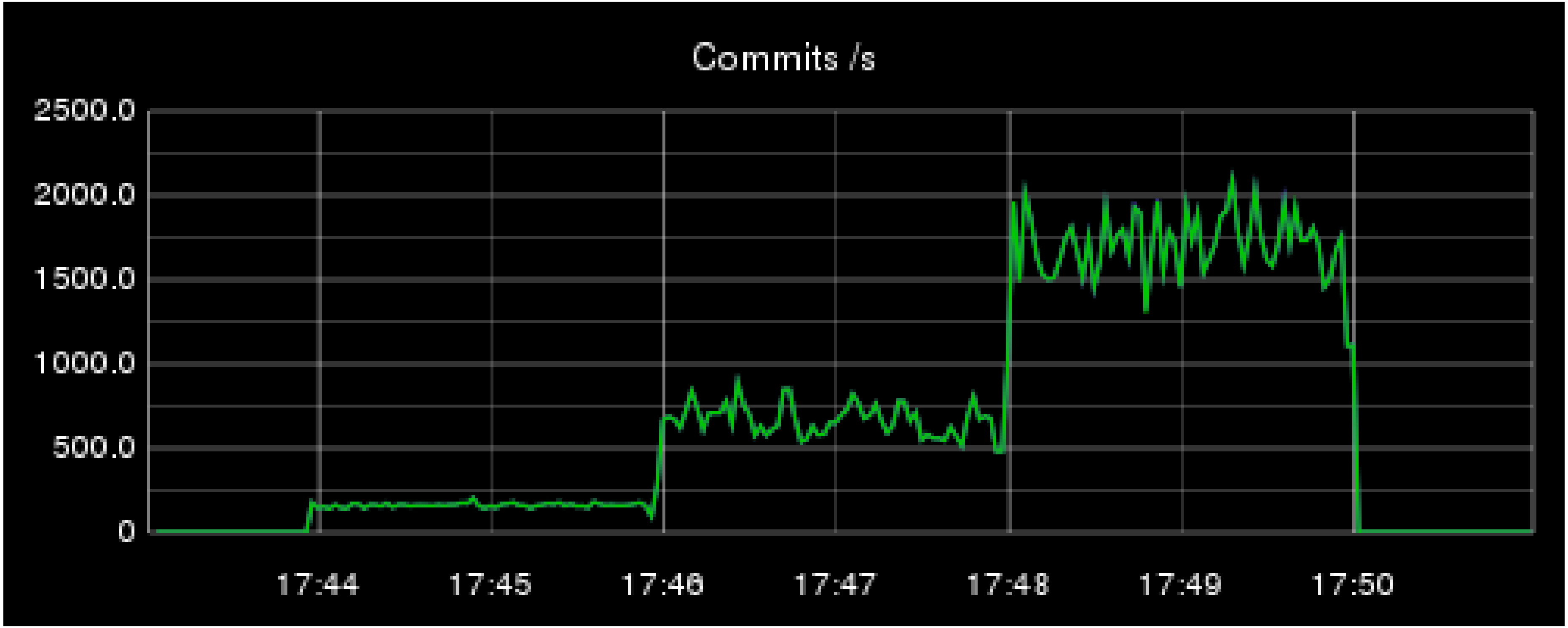
Jean-François Gagné (System Engineer)
jeanfrancois DOT gagne AT booking.com

Context

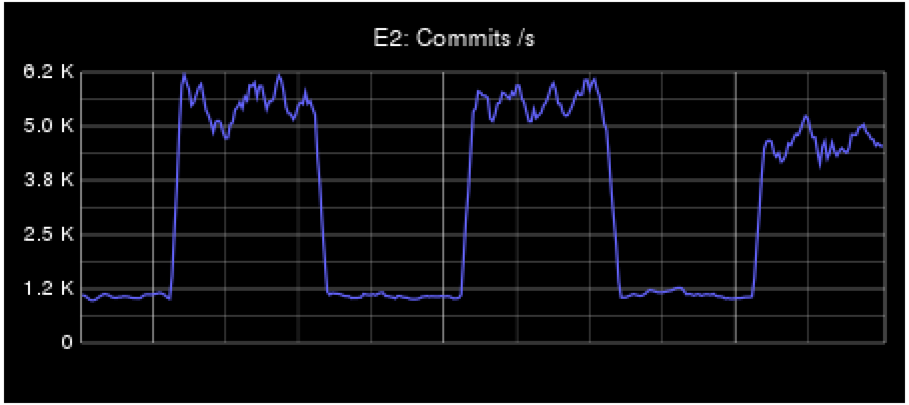
- We will cover MySQL 5.6, 5.7 and 8.0… and MariaDB 10.0 and 10.1

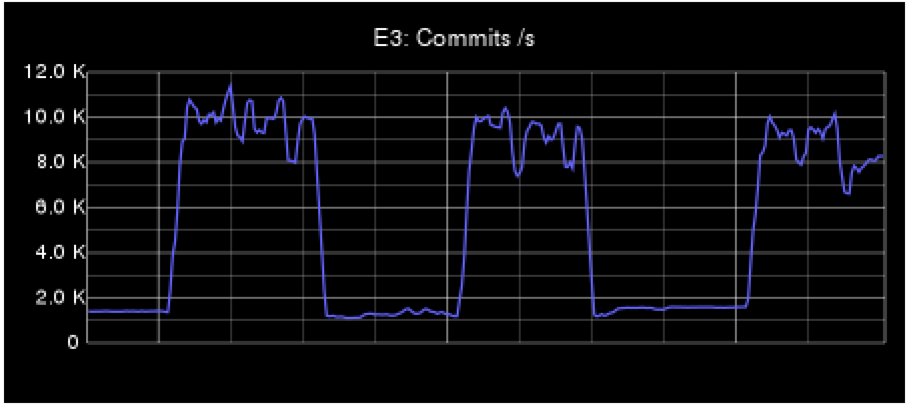
- MySQL 5.6 has support for *schema* based parallel replication
- MariaDB 10.0 has support for *domain id* based parallel replication
  and also has support for *group commit* based parallel replication (conservative)
- MariaDB 10.1 <u>adds</u> support for *optimistic* parallel replication
- MySQL 5.7 <u>adds</u> 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 <u>adds</u> support for *Write Set* parallelism identification
  (Write Set can also be found in MySQL 5.7 in Group replication)

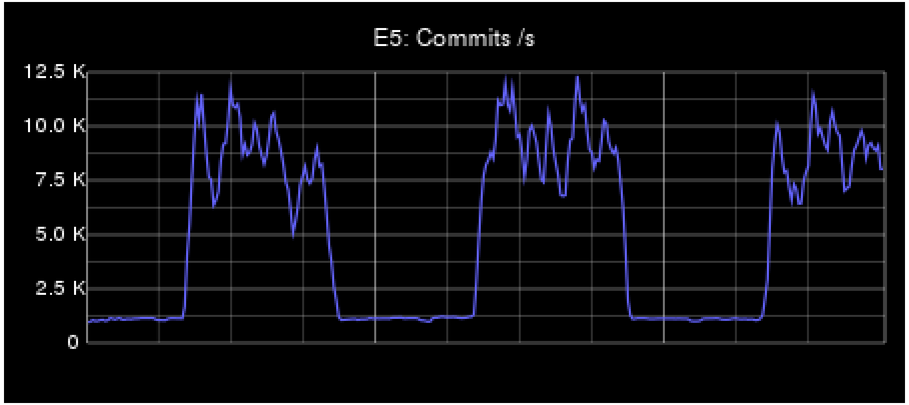
MariaDB 10.1 improvements

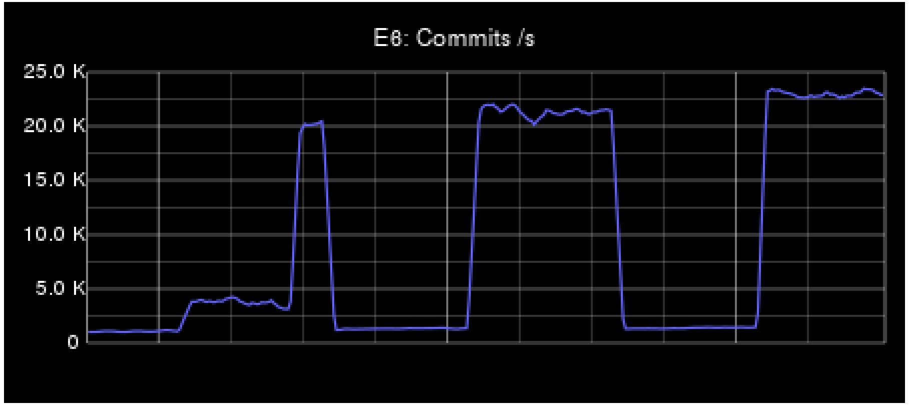


MySQL 8.0 improvements


E2: Commits /s


E3: Commits /s


E5: Commits /s


E6: Commits /s

Booking.com

- Based in Amsterdam since 1996
- Online Hotel and Accommodation (Travel) Agent (OTA):
  - +1.751.000 properties in 229 countries
  - +1.555.000 room nights reserved daily
  - +40 languages (website and customer service)
  - +15.000 people working in 198 offices worldwide
- Part of ~~the Priceline Group~~ Booking Holdings

- And we use MySQL and MariaDB:
  - Thousands (1000s) of servers

Booking.com’

- 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/

Summary

- Introducing Replication and Parallel Replication (*// Replication*)

- MySQL 5.6: schema based
- MariaDB 10.0: out-of-order and in-order
- MariaDB 10.1: +optimistic
- MySQL 5.7: +logical clock
- MySQL 8.0: +write set (also in 5.7 Group Replication)

- Along the way: benchmarks from Booking.com environments

Replication

- One master / one or more slaves

- The master records all its writes in a journal: *the binary logs*
- Each slave:
  - Downloads the journal and saves it locally (IO thread): *relay logs*
  - Executes the relay logs on the local database (SQL thread)
  - Could produce binary logs to be itself a master (log-slave-updates)
- Replication is:
  - Asynchronous → lag
  - Single threaded (in MySQL 5.5 and MariaDB 5.5) → slower than the master

Parallel 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 parallel replication 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
  - Some computer resources have better throughput at the cost of worse latency:
    - For CPU: number of cores increase but clock rate decrease
    - For remote storage: needs a network round-trip

In both cases, it would penalize single-threaded replication → // replication is the future

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: default for `slave_parallel_type` is `'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/

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…

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 chkpt 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, breaks p-GTID, …

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
- For heartbeat: one heartbeat per schema
- Do not use pseudo-GTID with schema-based parallel replication

MySQL 5.6”’ ’

- Worker states stored in mysql.slave_worker_info:
  - https://dev.mysql.com/worklog/task/?id=5599 (brace yourself for the 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)

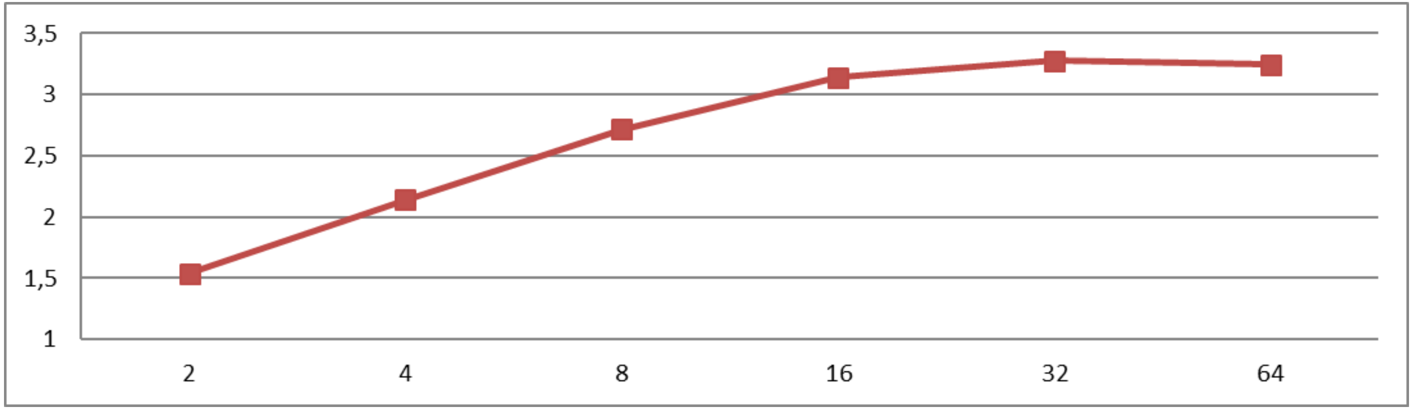
- MTS checkpoint:
  - After making sure gaps are filled, checkpointing advances the position of “`SHOW SLAVE STATUS`”
- Checkpointing is tried every *`slave_checkpoint_period`* (300 ms by default)

MySQL 5.6”’ ”

- Checkpointing might fail if the next needed transaction is still running → 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…)

MySQL 5.6: benchmark

- Booking.com session store is sharded with many schema per database:
  - 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
  - MySQL 5.6 (in 2016), >1 TB per node, 20 schema (designed for // replication), magnetic disks



Booking.com

MySQL 5.6: improvements

- What I would like to see improved (this does not exist yet):
  - Replace transaction execution tracking (mysql.slave_worker_info) by multi-source
  - This makes things more visible and easier to understand:
    - Each “sources” would process a schemawith observability of each source position
  - This also makes schema-based parallel replication crash-safe without GTID
  - But maybe improving this type of parallel replication is not needed with MySQL 8.0…

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, …

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 (related bugs with some context, all fixed: MDEV-6589 & MDEV-9138 & MDEV-10863)
- Removing gaps: `STOP SLAVE; SET GLOBAL slave_parallel_threads = 0; START SLAVE;`
  - To avoid re-downloading relay logs, use below:
    `STOP SLAVE SQL_THREAD; SET GLOBAL slave_parallel_threads = 0; START SLAVE;`
    (https://jfg-mysql.blogspot.com/2015/10/bad-commands-with-mariadb-gtids-2.html)
    (https://jira.mariadb.org/browse/MDEV-8945)
- 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

MariaDB 10.0: out-of-order”

- Same improvement wished: implement this with multi-source.

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 they aprear in the binary logs of the master

- **Deployment**:
  - Needs a MariaDB 10.0 master
  - On slaves: "`SET GLOBAL slave_parallel_threads = N;`" (with N > 0)

MariaDB 10.0: in-order’

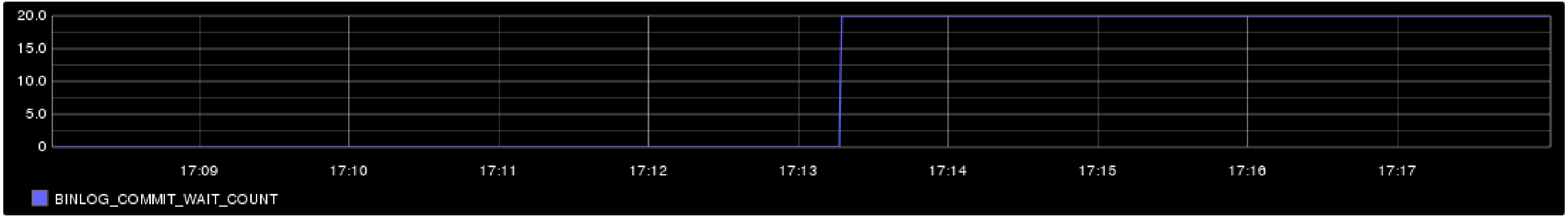
- Binary logs extract:

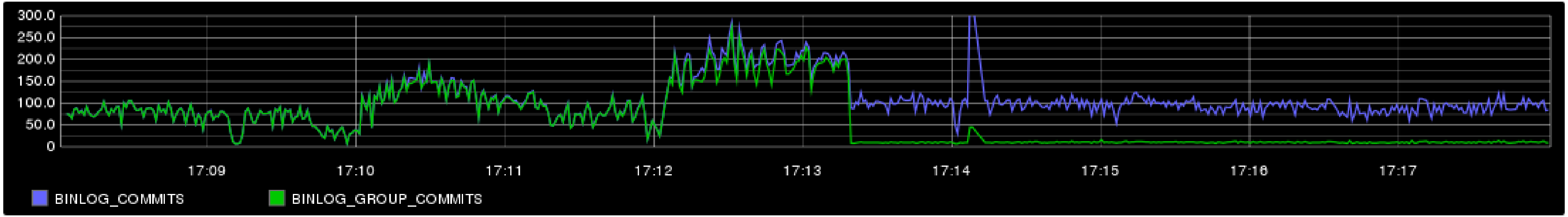
```
...
#150316 11:33:46 ... GTID 0-1-184 cid=2324
#150316 11:33:46 ... GTID 0-1-185 cid=2335
...
#150316 11:33:46 ... GTID 0-1-189 cid=2335
#150316 11:33:46 ... GTID 0-1-190
#150316 11:33:46 ... GTID 0-1-191 cid=2346
...
#150316 11:33:46 ... GTID 0-1-197 cid=2346
#150316 11:33:46 ... GTID 0-1-198 cid=2361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

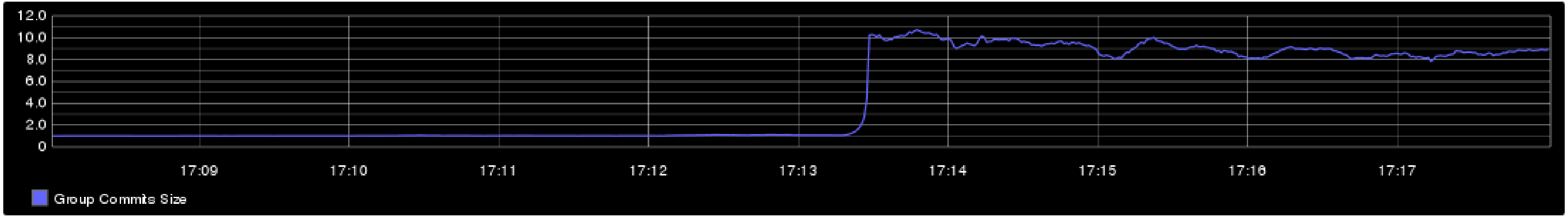
MariaDB 10.0: in-order”

- 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

MariaDB 10.0: in-order’’’







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, …)

MariaDB 10.0: in-order”’ ”

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

```
+---+
| X |
+---+
  |
  V
+---+
| Y |
+---+
  |
  V
+---+
| Z |
+---+
```

On X:

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

On Y:

```
----Time---->
  B---C
  B---C
  B--------C
  B--------C
```

On Z:

```
----Time---------->
  B---C
  B---C
       B--------C
       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

MariaDB 10.0: scheduling

- How is work scheduled to threads:
  - One queue per thread containing transactions to execute by this thread
  - The coordinator is dispatching work round-robin to threads until a queue is full
    - If a queue is full, dispatching pauses (big transactions block scheduling)
  - Once a thread is scheduled work in a domain, it is stuck on this domain
    - If all threads are busy, a new domain will starve until a thread has processed all its queue

- Solutions: tuning parameters:
  - `slave-parallel-max-queued` (default 128KB): size of the buffer to queue trx
  - `slave_domain_parallel_threads` (default 0): nb. of threads a domain can use

- Again: avoid big transactions (size in the binlogs)

MariaDB 10.0: 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)

MariaDB 10.0: 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
  → below should not happen (MDEV-7249):

```
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

- More details at: https://blog.booking.com/evaluating_mysql_parallel_replication_2-slave_group_commit.html

MariaDB 10.0: benchmarks

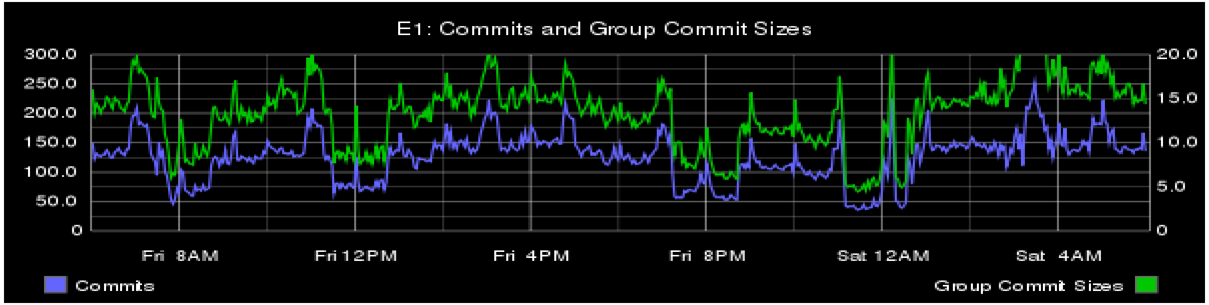
- Four environments (E1, E2, E3 and E4):
  - A is a MySQL 5.6 master
  - B is a MariaDB 10.0 intermediate master
  - C is a MariaDB 10.0 intermediate master doing slave group commit
  - D is using the group commit information from C to run transaction in parallel

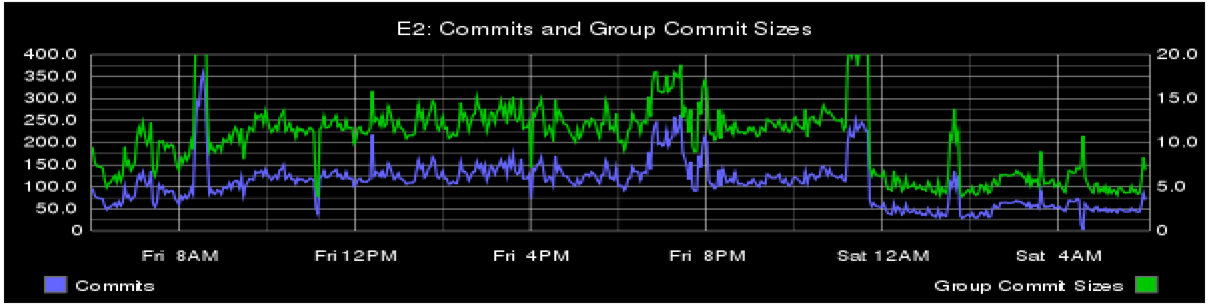
```
+----+     +----+     +----+     +----+
| A  | --> | B  | --> | C  | --> | D  |
+----+     +----+     +----+     +----+
```

- 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

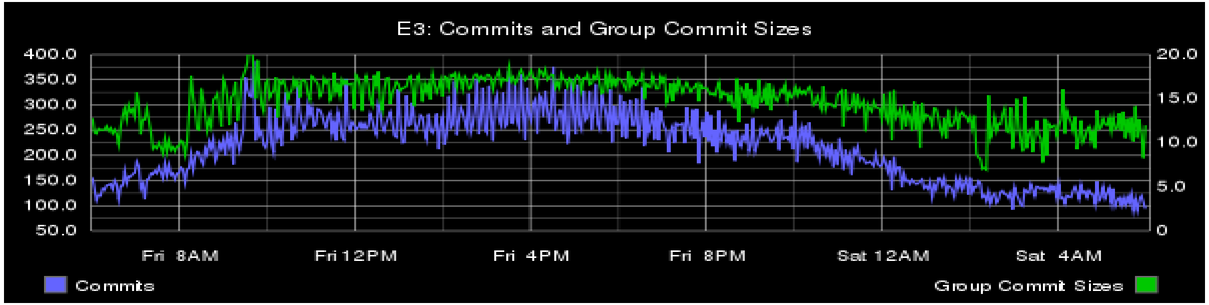
MariaDB 10.0: benchmarks’

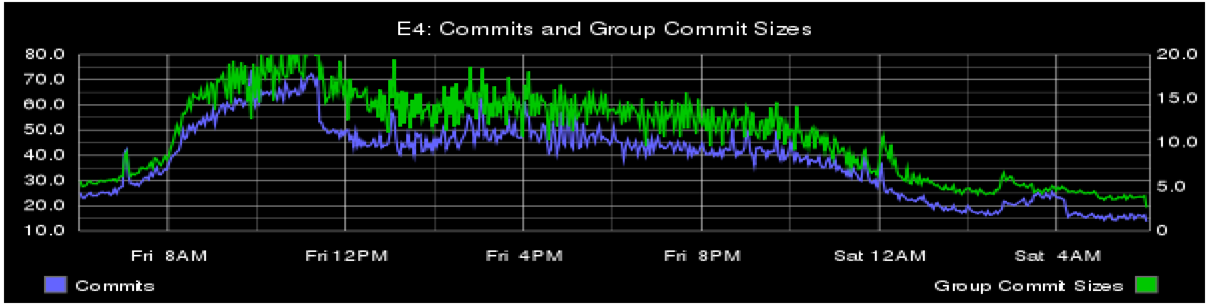
- Catching up 24 hours of replication delay with 0, 5, 10, 20 and 40 threads





MariaDB 10.0: benchmarks’’





More details at:

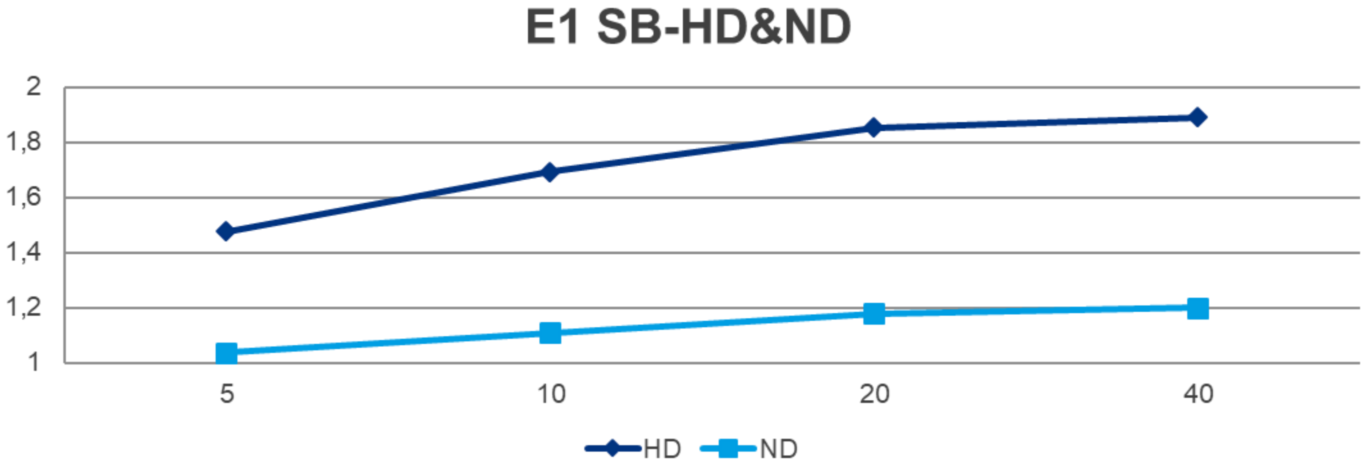
http://blog.booking.com/evaluating_mysql_parallel_replication_3-benchmarks_in_production.html

MariaDB 10.0: benchmarks’’’

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

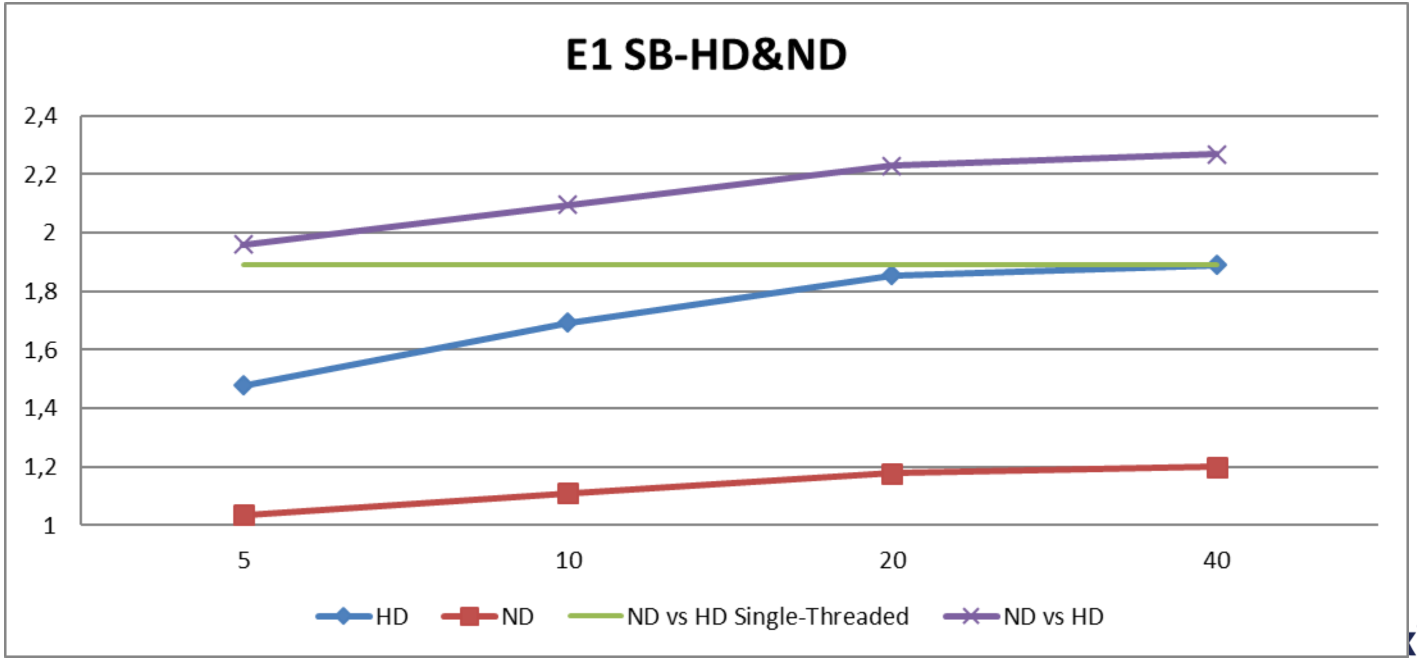
ND Single-Threaded: 1h24.09

MariaDB 10.0: benchmarks’’’

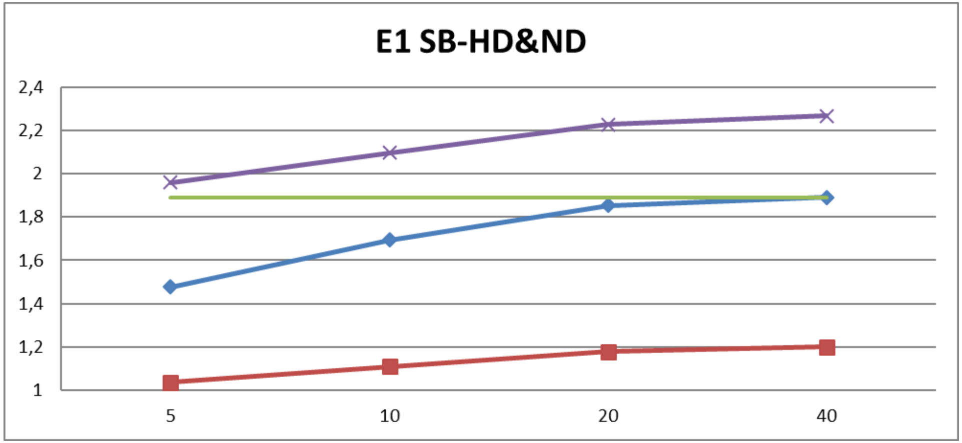
Slave with binlogs (*SB*) but without log-slave-updates

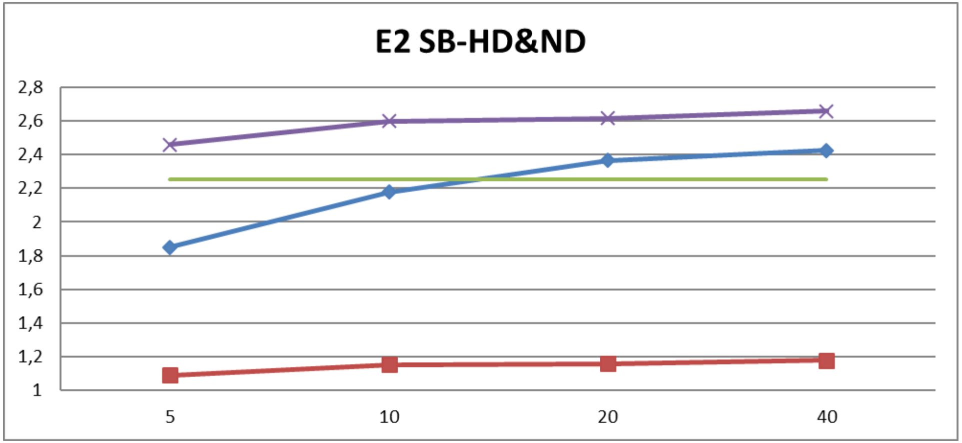
High Durability (*HD*): “`sync_binlog = 1`” + “`trx_commit = 1`”

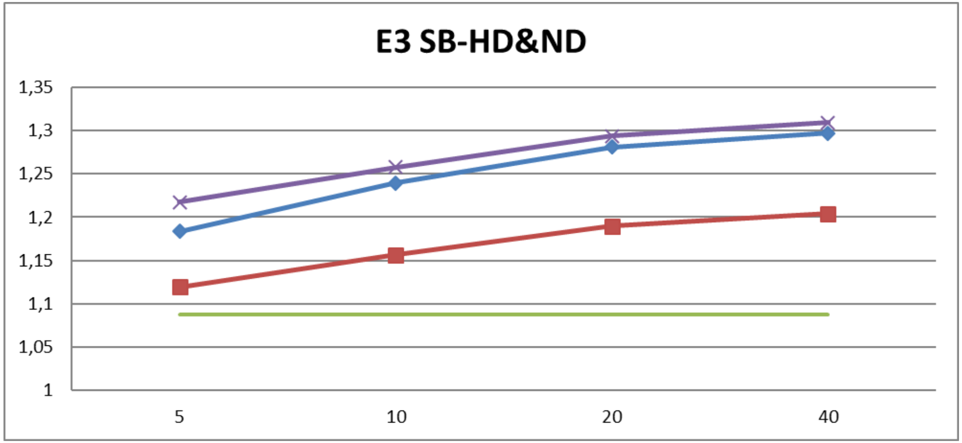
No Durability (*ND*): “`sync_binlog = 0`” + “`trx_commit = 2`”

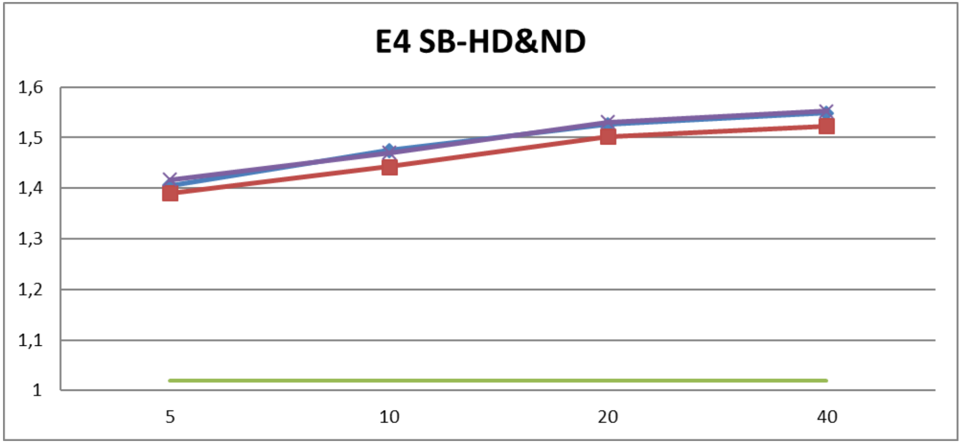


MariaDB 10.0: benchmarks”’ ’

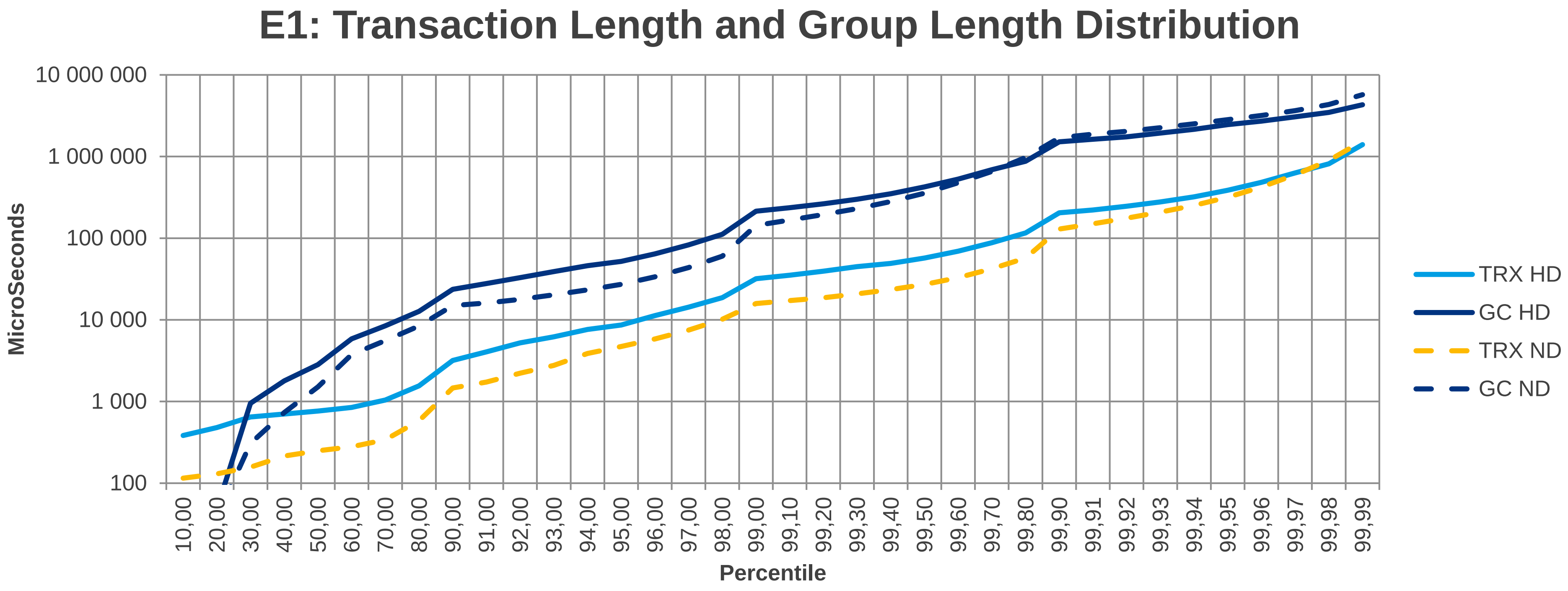






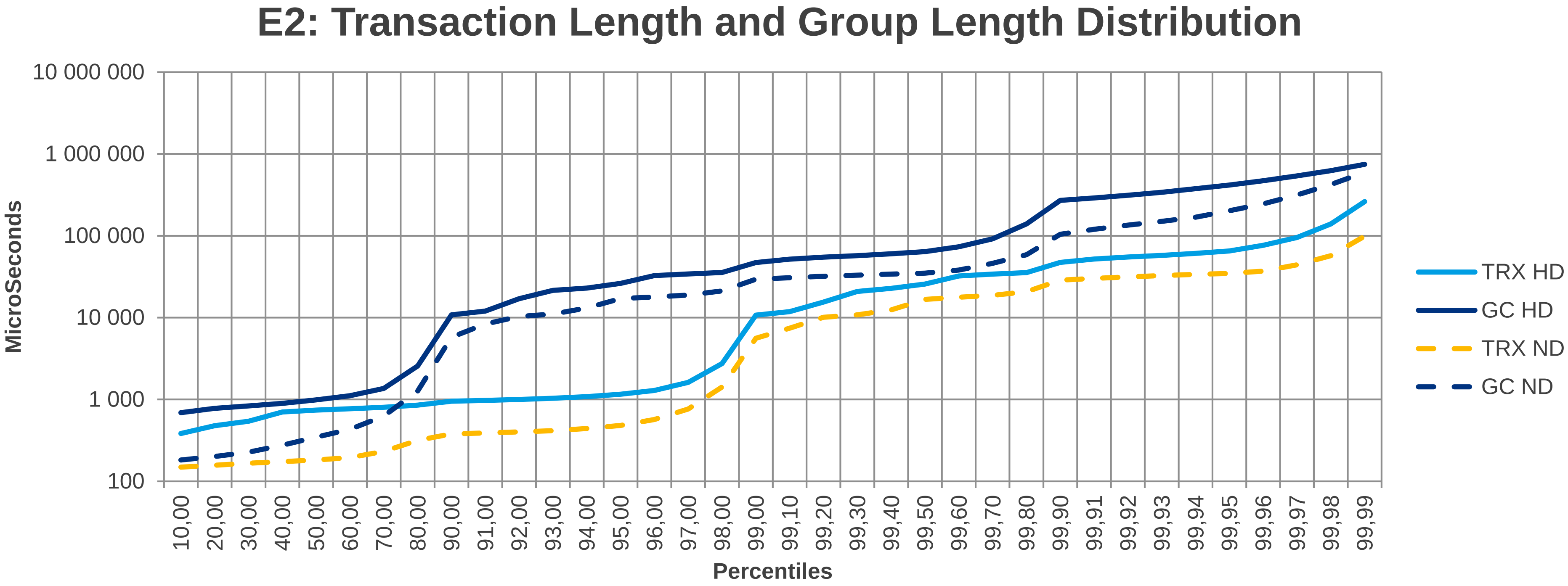


MariaDB 10.0: benchmarks’’’ ’’



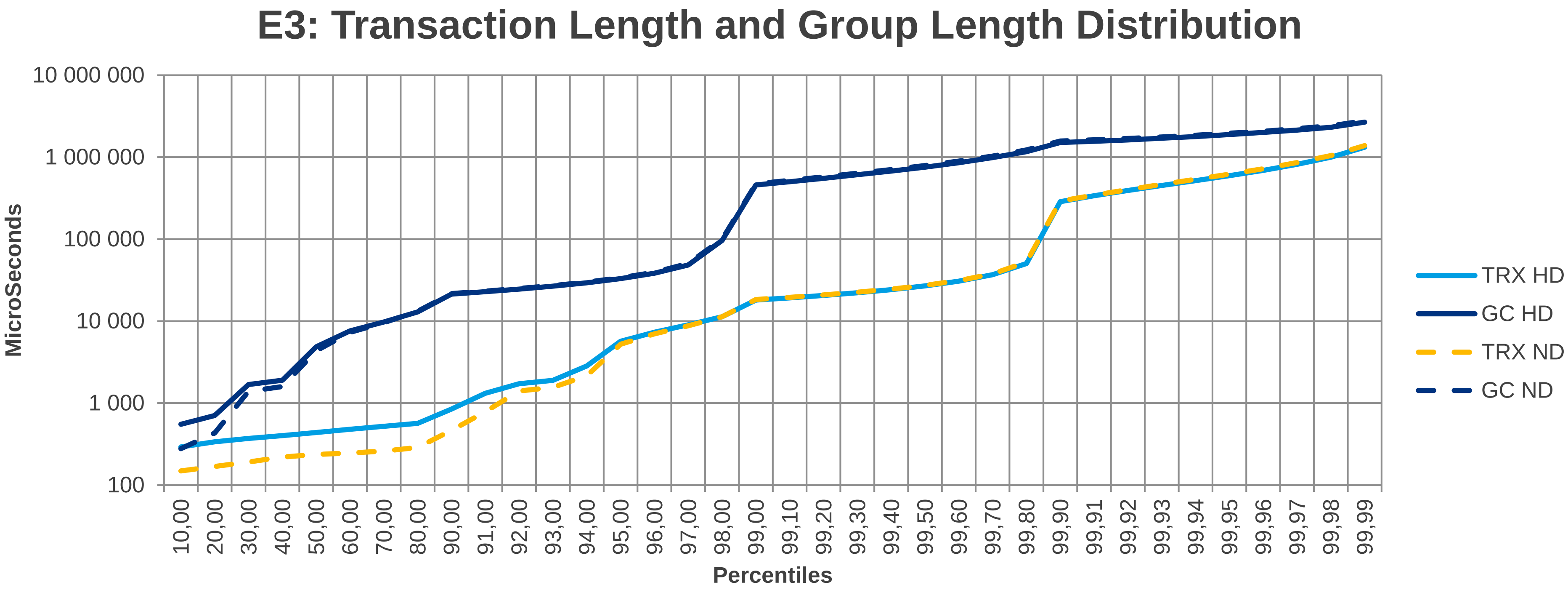
Booking.com

MariaDB 10.0: benchmarks’’’ ’’’



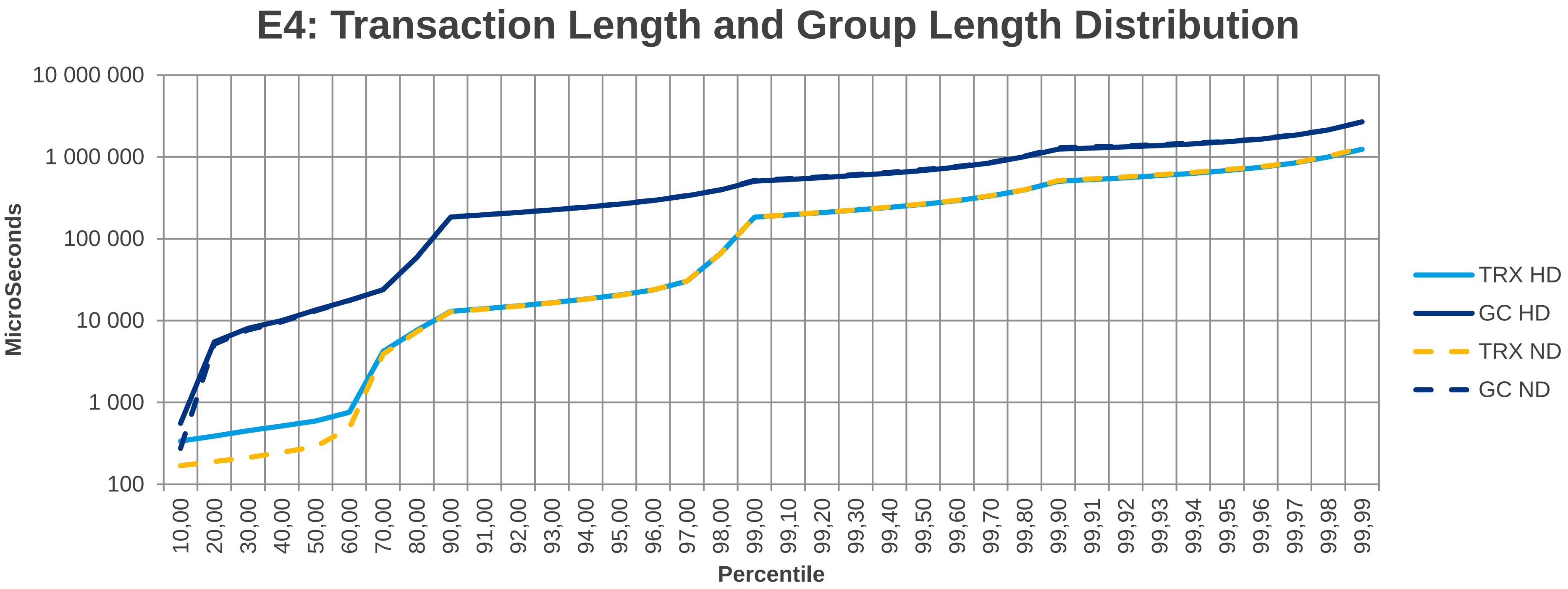
Booking.com

MariaDB 10.0: benchmarks’’’ ’’’ ’



Booking.com

MariaDB 10.0: benchmarks’’’ ’’’ ’’



MariaDB 10.0 in production at B.com

- 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

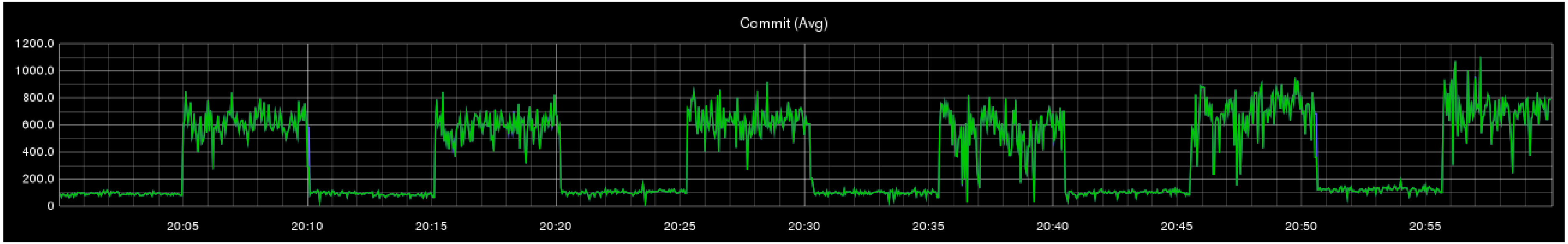
- We optimized group commit and enabled parallel replication:

```
set global BINLOG_COMMIT_WAIT_COUNT = 75;
set global BINLOG_COMMIT_WAIT_USEC  = 300000;  (300 milliseconds)
set global SLAVE_PARALLEL_THREADS   = 80;
```

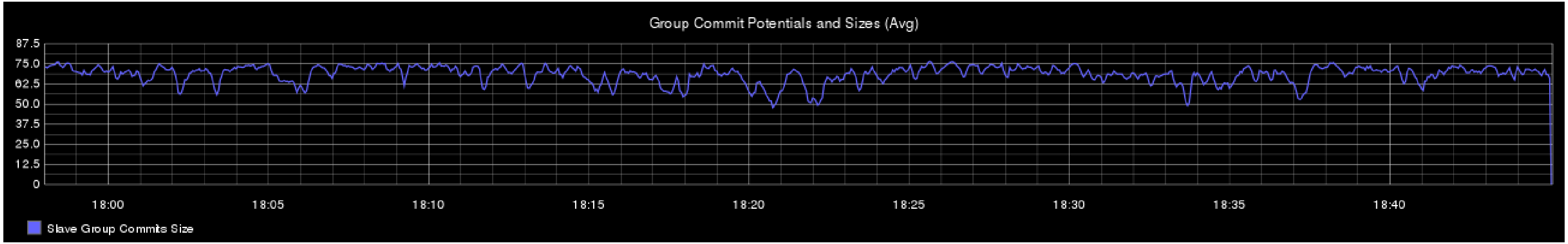
- And we tuned the application:
  - Break big transaction in many small transactions
  - Increase the number of concurrent sessions to the database

MariaDB 10.0 in production at B.com’

- 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)

Kudos to MariaDB Engineers

- Parallel Replication allows to identify old bugs
- An InnoDB race condition caused a query by Primary Key to do a full table scan:
  - Happens if two queries request InnoDB table statistics exactly at the same time (in the optimizer)
  - This is unlikely IRL but frequent with // replication (two trx on same table in same group commit)
  - Hard to notice/reproduce for normal queries, very observable with replication
    (one of many slaves blocked for minutes on an UPDATE by Primary Key)
- 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 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)
  - More context in https://www.facebook.com/valerii.kravchuk/posts/1073608056064467

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

MariaDB 10.1: in-order - Back to 10.0

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

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

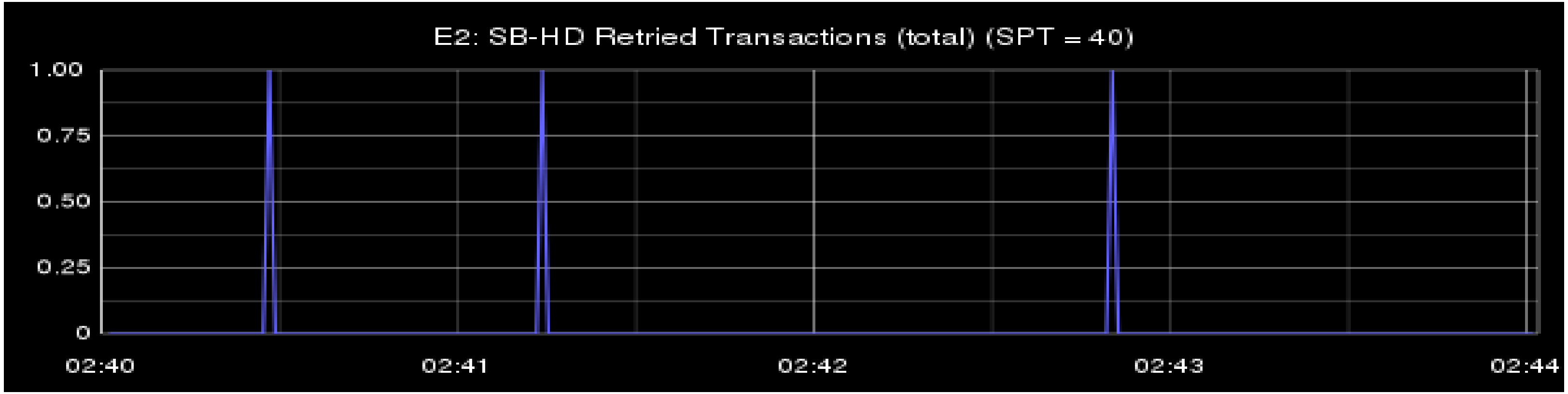
- On the slaves, T2 (ready to commit) blocks T1 (because index update, …), but T1 must commit before T2 → deadlock

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

- To solve this deadlock, MariaDB replication applier kills T2, unblocking T1
- Corresponding global status: `slave_retried_transactions`

MariaDB 10.1: in-order - Back to 10.0’

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



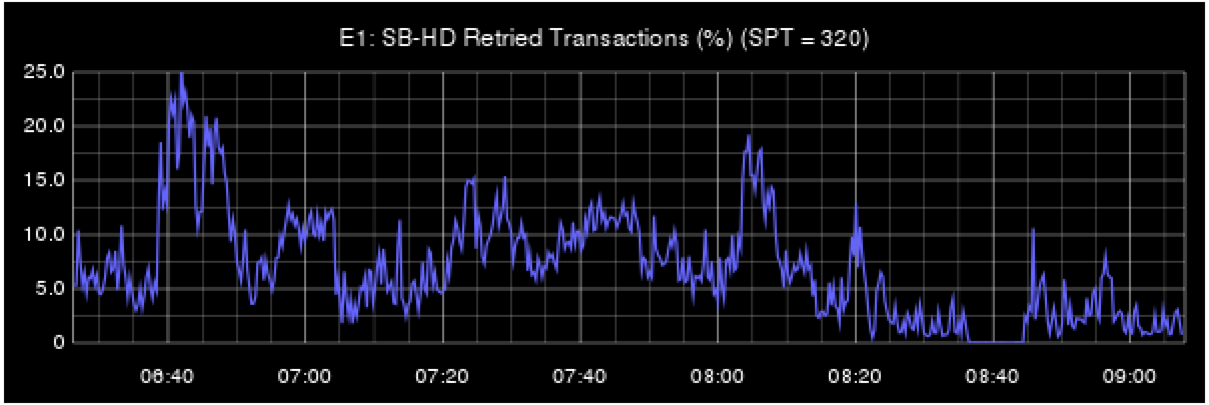
➢ Retry happen 3 times for 600,000 transactions → not often at all

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)

MariaDB 10.1: optimistic’

- By default, MariaDB will retry transactions up to 10 times
- This can be tuned with `slave_transaction_retries`



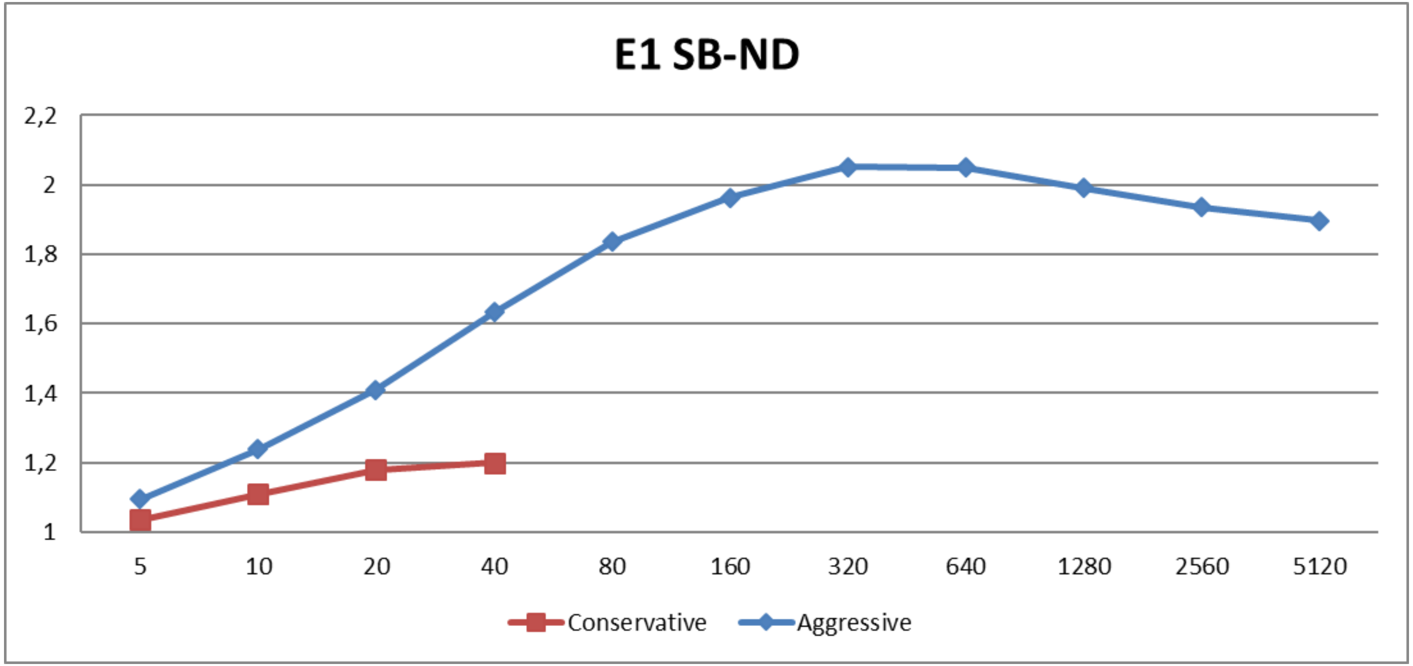
MariaDB 10.1: benchmarks

- 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

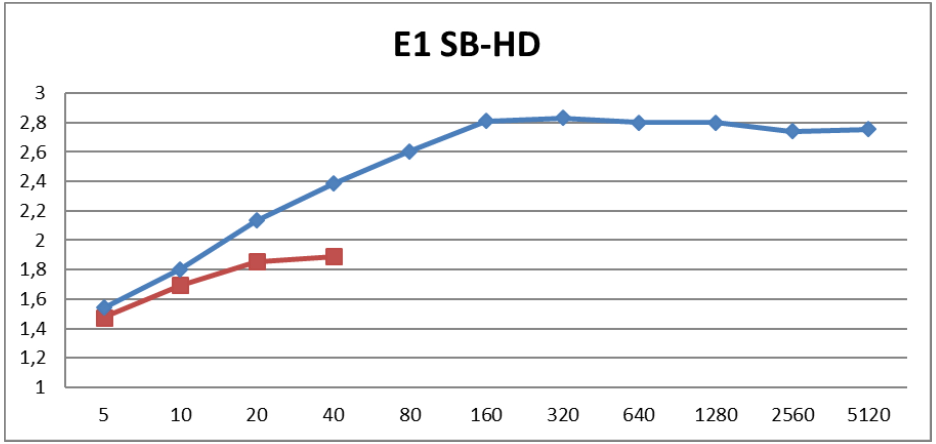
```
+----+     +----+     +----+     +----+
| A  | --> | B  | --> | C  | --> | D  |
+----+     +----+     +----+     +----+
                        |
                        |        +----+     +----+
                        +------> | C2 | --> | D2 |
                                 +----+     +----+
```

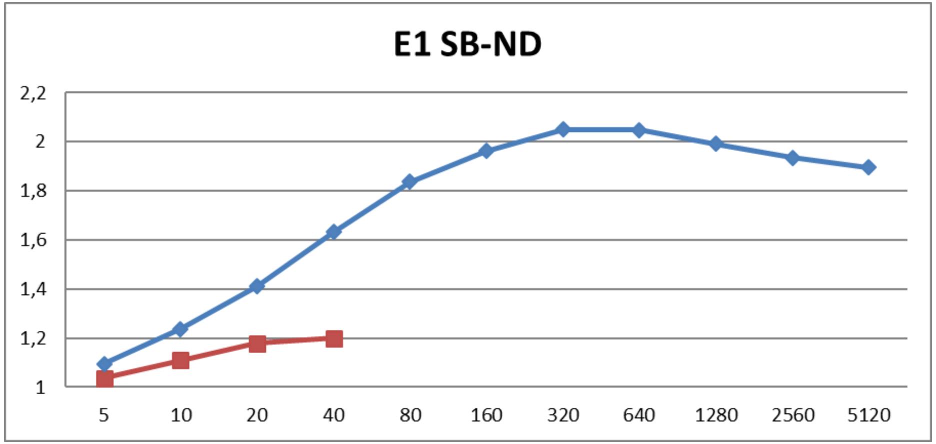
- D and D2 are the same hardware for comparing conservative and aggressive:
  - D runs with SPT = conservative (using the “slave Group Commit” binary logs)
  - D2 runs with SPT = aggressive (needs a 10.1 master to work)

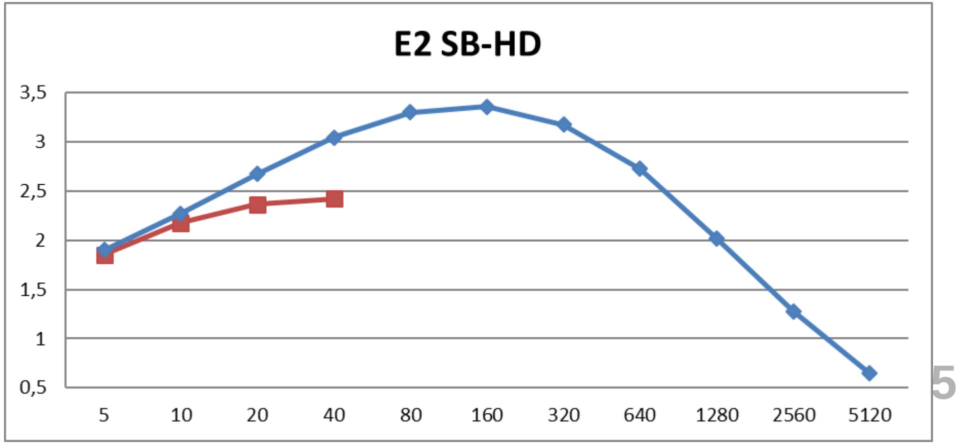
MariaDB 10.1: benchmarks’

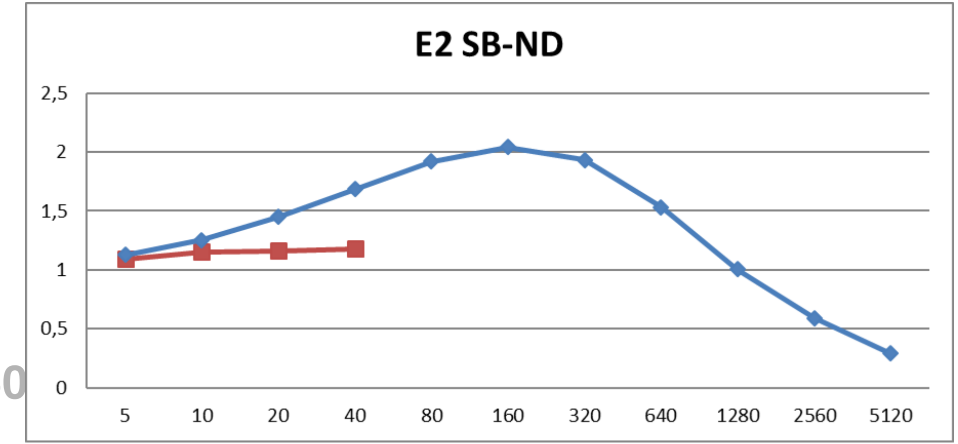


MariaDB 10.1: benchmarks”

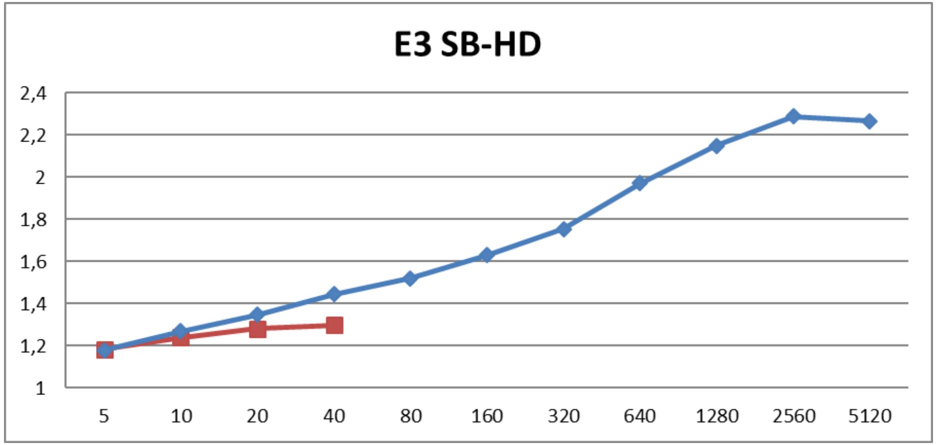


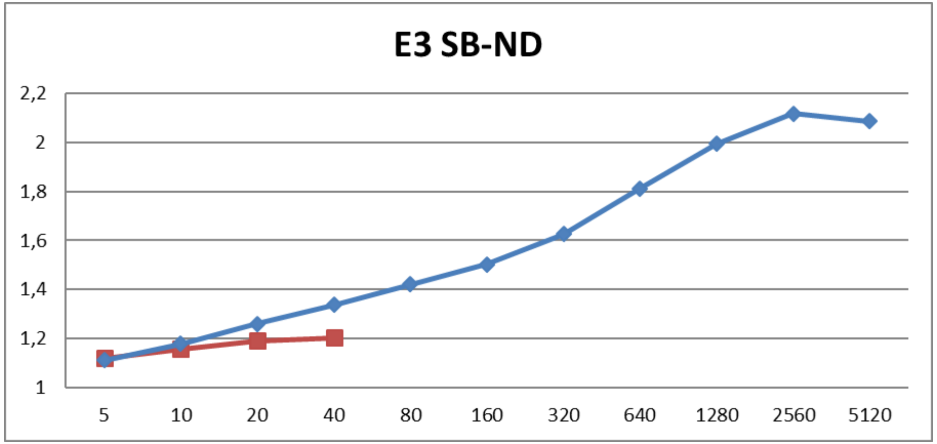


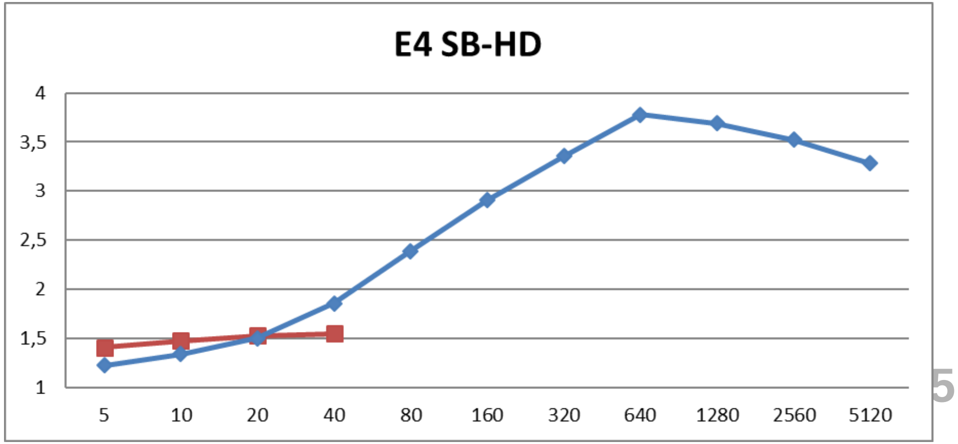


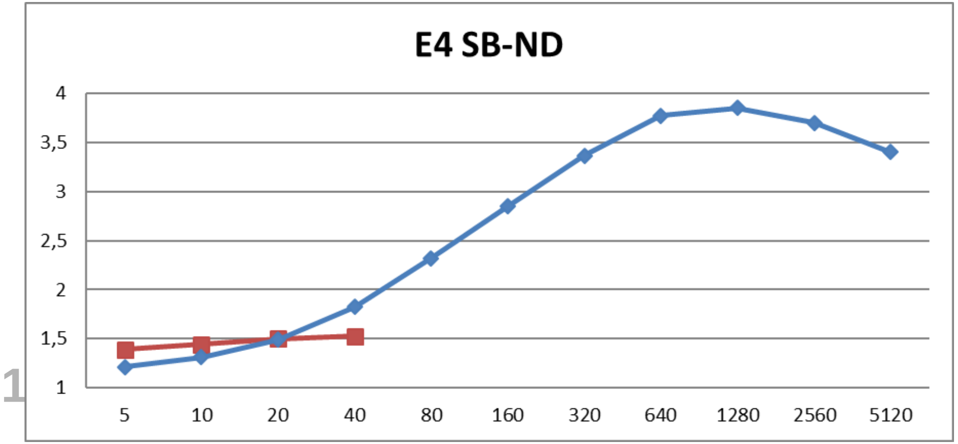


MariaDB 10.1: benchmarks’’’

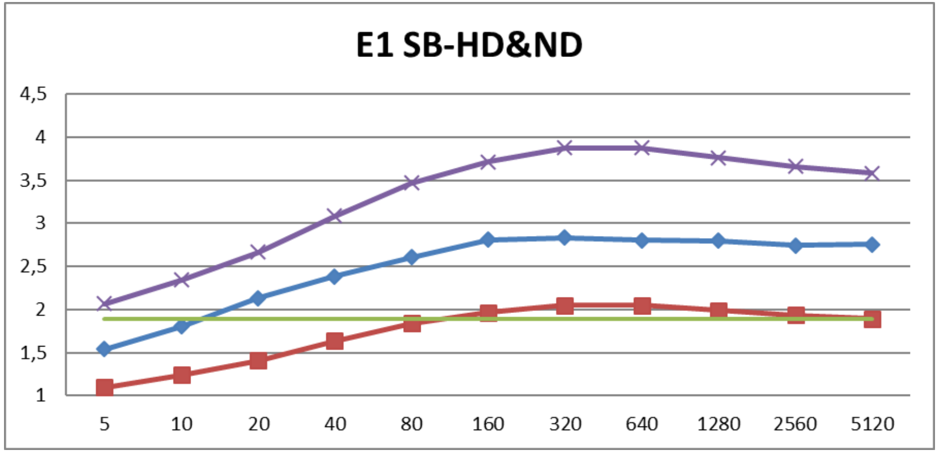


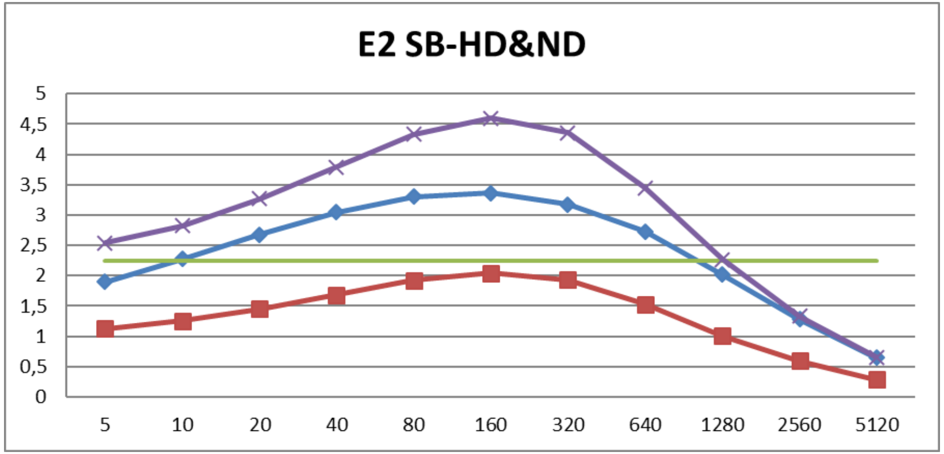


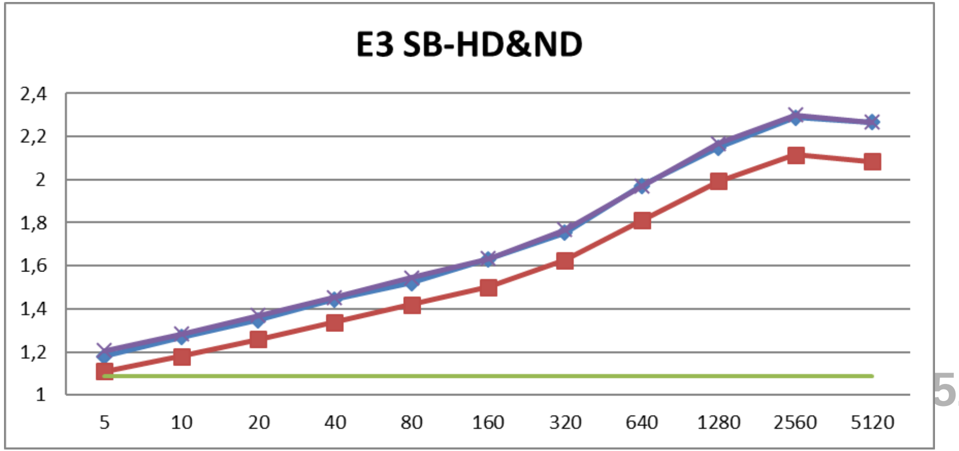


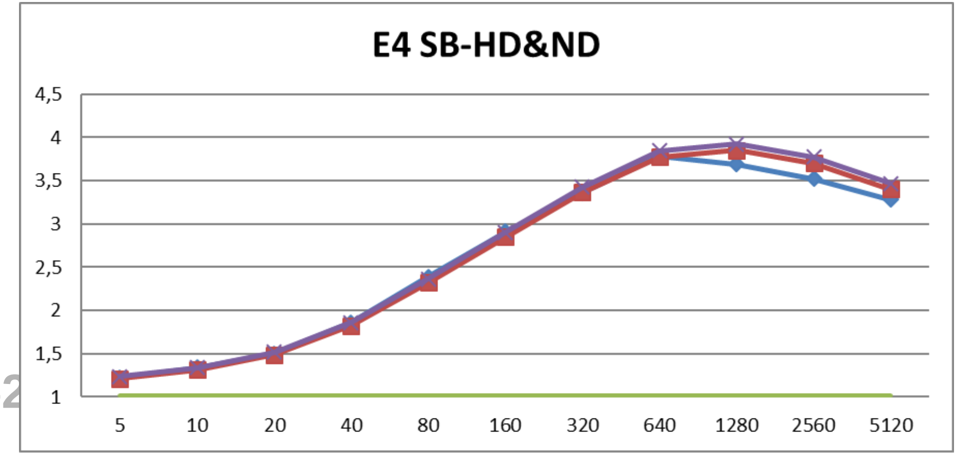


MariaDB 10.1: benchmarks”’ ’





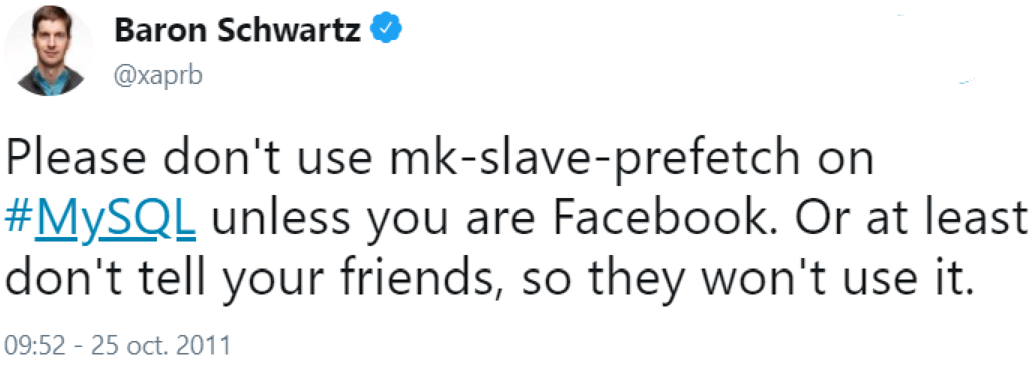




MariaDB 10.1: optimistic

- Getting speedups for high number of threads is surprising
- It might be because the high number of threads acts as a “prefetcher”
- It looks like we “re-visited” replication booster…
  - mk-slave-prefetch (in 2011 by Baron Schwartz)
  - Making slave pre-fetching work better with SSD (in 2011 by Yoshinori Matsunobu)
  - replication-booster-for-mysql (in 2012 still by Yoshinori)

- This is not true anymore:
- You can use optimistic parallel replication



Baron Schwartz

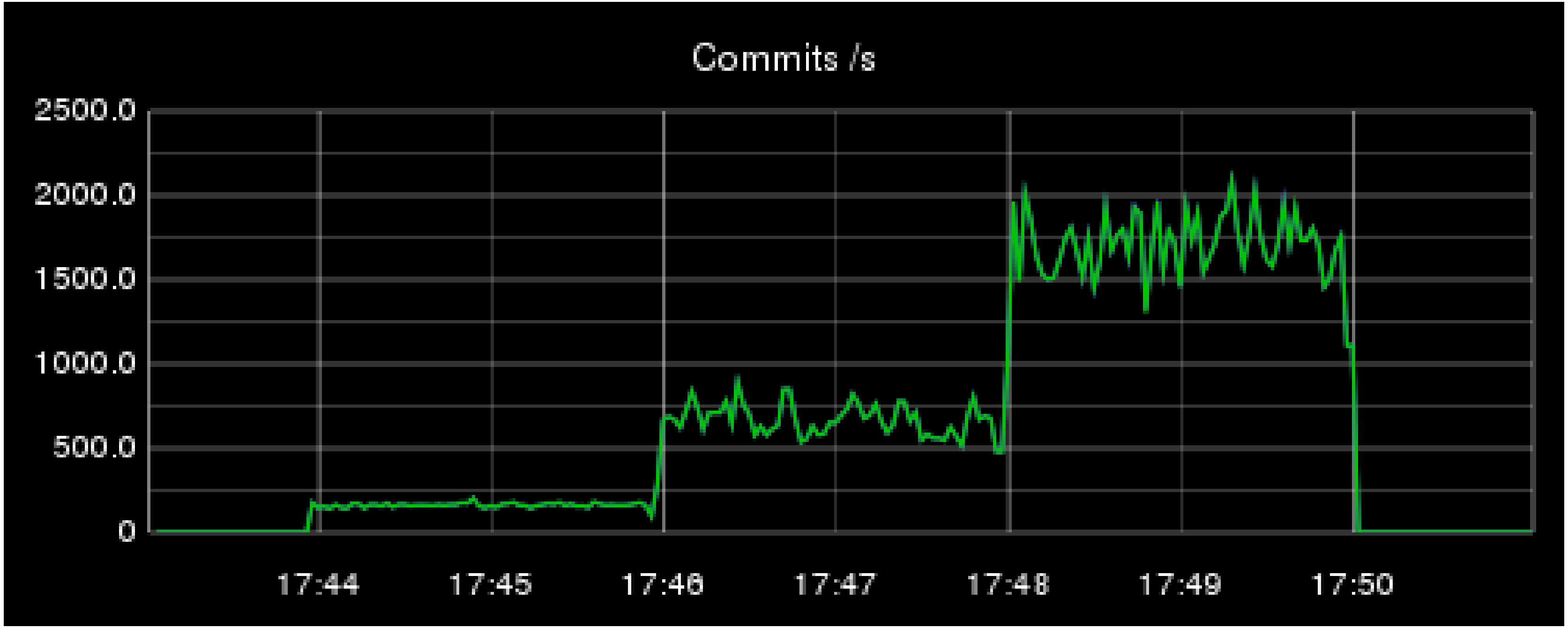
@xaprb

Please don't use mk-slave-prefetch on #MySQL unless you are Facebook. Or at least don't tell your friends, so they won't use it.

09:52 - 25 oct. 2011

MariaDB 10.1 in production at B.com

- Single-threaded, conservative and aggressive commit rate:



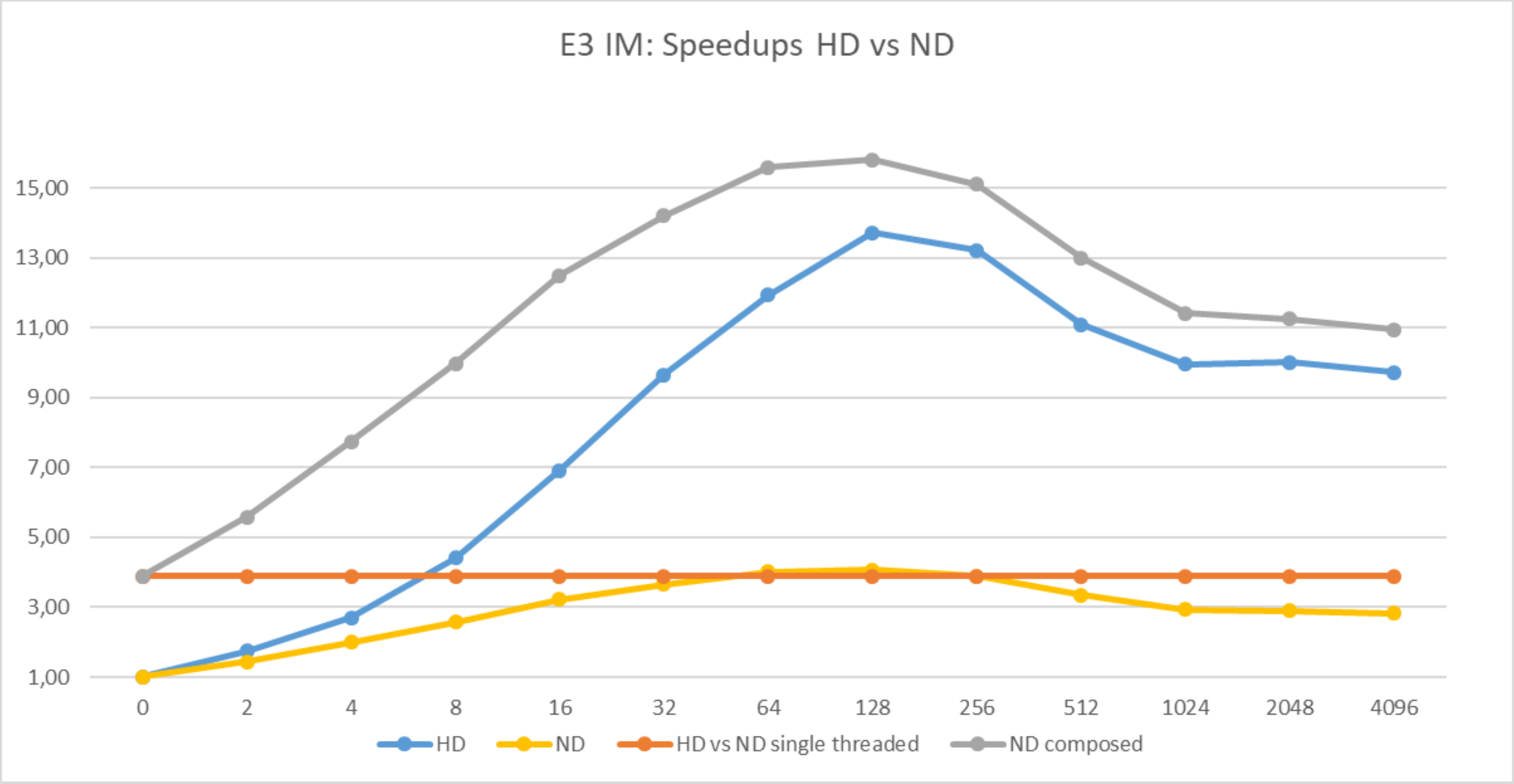
MariaDB 10.2: recent benchmarks

- The results presented so far are dating from 2015
- And they were done on servers with magnetic disk

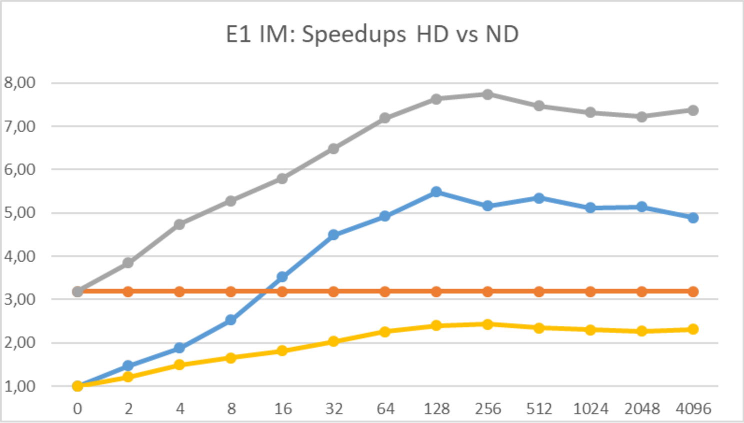
- What about more recent hardware ?
- And more recent software ?

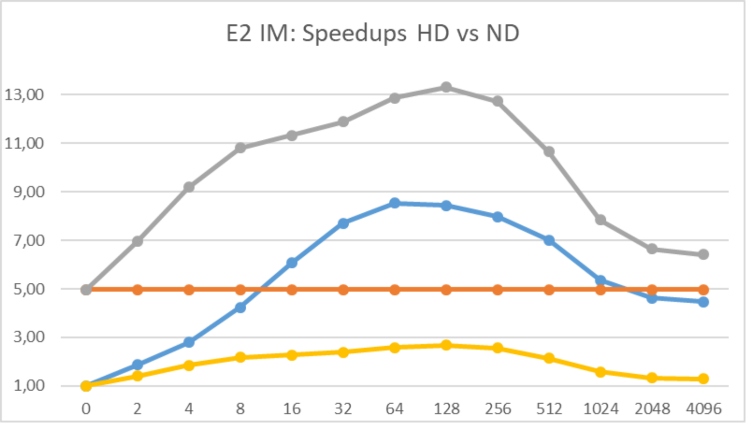
- Newer benchmarks with MariaDB 10.2.12
- On recent hardware with SSDs
- On four different environments (but E3 of 10.1 == E3 of 10.2)
- Not in SB configuration (slave with binlogs) but in IM (intermediate master)

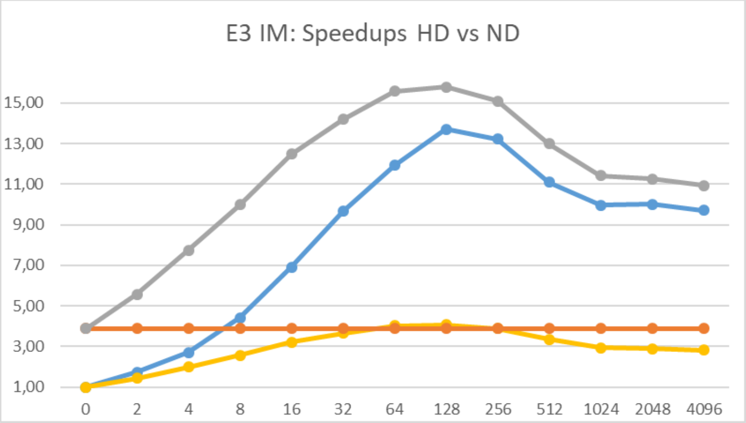
MariaDB 10.2: recent benchmarks’

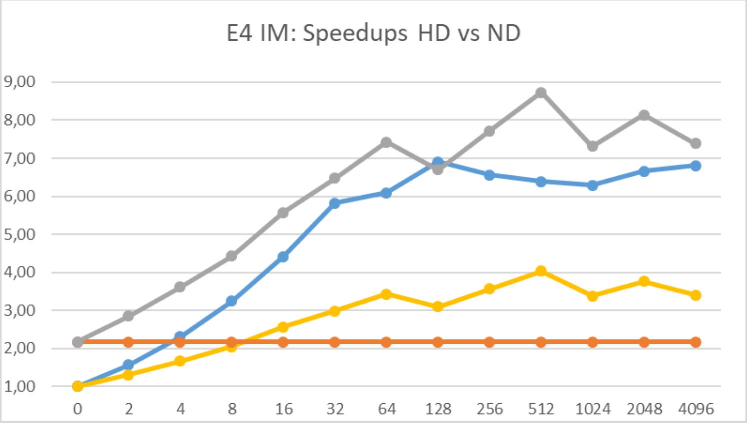


MariaDB 10.2: recent benchmarks”


E1 IM: Speedups HD vs ND


E2 IM: Speedups HD vs ND


E3 IM: Speedups HD vs ND


E4 IM: Speedups HD vs ND

MariaDB 10.2: recent benchmarks’’’

- Some things to know about Optimistic Parallel Replication in MariaDB 10.2
  - MDEV-15135: SHOW GLOBAL STATUS can deadlock Optimistic Parallel Replication (so does a SELECT in INFORMATION_SCHEMA)
  - MDEV-15152: Optimistic Parallel Replication deadlocks with START SLAVE UNTIL
  - MDEV-15608: MariaDB sometimes crashes on rollback of a transaction with BLOBs

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 // replication from 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)
  - `LOGICAL_CLOCK` type is implemented by putting interval information in the binary logs
- `LOGICAL_CLOCK` is limited by the following:
  - Problems with long/big transactions and problems with intermediate masters (*IM*)
- And it is optimized by slowing down the master to speedup the slave:
  - `binlog_group_commit_sync_delay`
  - `binlog_group_commit_sync_no_delay_count`

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 be careful about these:
    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` (FR to remove this limitation: Bug#75396)
  - Unclear if replication crash safe (surprising because no gap): Bug#80103 & Bug#81840
  - And it can hang if `slave_transaction_retries` is too low: Bug#89247

MySQL // Replication Guts: Intervals

- In MySQL (5.7 and higher), 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

MySQL 5.7 uses parallelism on master to generate intervals:

- `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: Intervals Quality

- For measuring parallelism identification quality with MySQL, we have 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*(n+1) / 2

```
#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: Intervals Quality

- For measuring parallelism identification quality with MySQL, we have 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*(n+1)/2`
  - ➢ `AMIL = (n+1)/2` (after dividing by n), algebra gives us `n = AMIL * 2 - 1`
- This mapping could give a hint for `slave_parallel_workers`

(http://jfg-mysql.blogspot.com/2017/02/metric-for-tuning-parallel-replication-mysql-5-7.html)

MySQL: 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: Intervals Quality”

- Script to compute the Average Modified Interval Length:

```
file=my_binlog_index_file;
echo _first_binlog_to_analyse_ > $file;
mysqlbinlog --stop-never -R --host 127.0.0.1 $(cat $file) |
  grep "^#" | grep -e last_committed -e "Rotate to" |
  awk -v file=$file -F "[ \t]*|=" '$11 == "last_committed" {
                                         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;}'
```

(https://jfg-mysql.blogspot.com/2017/02/metric-for-tuning-parallel-replication-mysql-5-7.html)

MySQL: Intervals Quality’’’

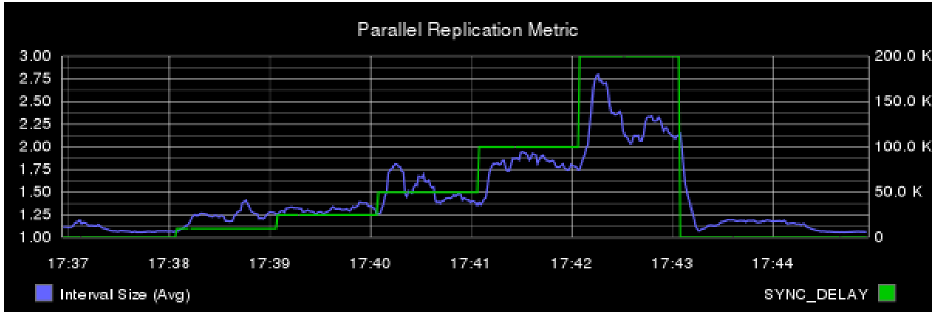
- Computing the AMIL needs parsing the binary logs
- This is complicated and needs to handle many special cases

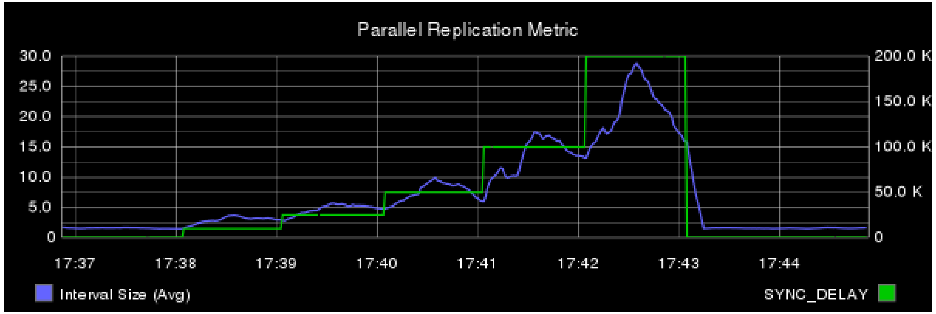
- Exposing counters for computing the AMIL would be better:
  - Bug#85965: Expose, on the master, counters for monitoring // information quality.
  - Bug#85966: Expose, on slaves, counters for monitoring // information quality.

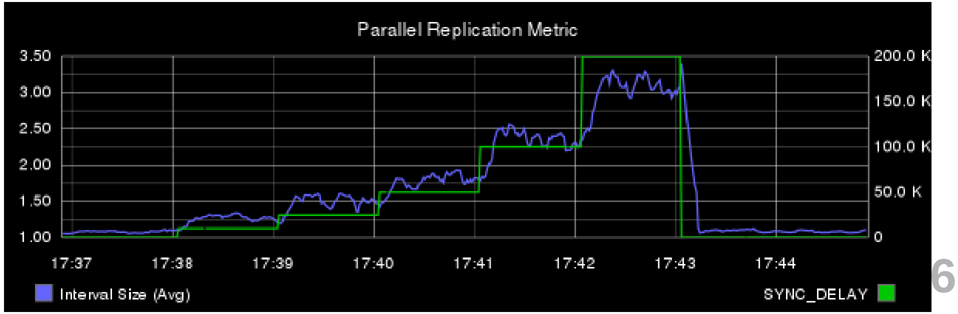
(https://jfg-mysql.blogspot.com/2017/02/metric-for-tuning-parallel-replication-mysql-5-7.html)

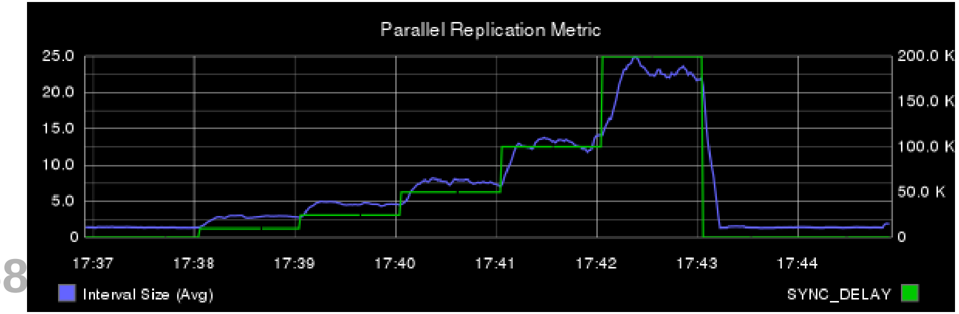
MySQL 5.7: Tuning

- AMIL without and with tuning (delay) on four (4) Booking.com masters:
  (speed-up the slaves by increasing `binlog_group_commit_sync_delay`)









MySQL 5.7: Benchmarks

- I have nothing to show…
- Because MySQL 5.7 does not have something like Slave Group Commit
- And I do not want to risk causing problems in production
- I expect very similar to MariaDB 10.0 (so not great unless a lot of tuning is done)
- And the tuning is not straightforward

- But there is something very interesting in 5.7.22 (released on 2018-04-19)
- And we will talk about this later…

MySQL 8.0: Write Set

- MySQL 8.0.1 introduced 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:
  - `transaction_write_set_extraction =[ OFF | MURMUR32 |` **`XXHASH64`** `]`
  - `binlog_transaction_dependency_history_size` (default to 25000)
  - `binlog_transaction_dependency_tracking =`
    `[` **`COMMIT_ORDER`** `| WRITESET_SESSION | WRITESET ]`

- `WRITESET_SESSION`: no two updates from the same session can be reordered
- `WRITESET`: any transactions which write different tuples can be parallelized

- `WRITESET_SESSION` will not work well for cnx recycling (Cnx Pools or Proxies):
  - Recycling a connection with `WRITESET_SESSION` impedes parallelism identification
  - Unless using the function reset_connection (with Bug#86063 fixed in 8.0.4)

MySQL 8.0: Write Set’’

- To use Write Set on a Master:
  - binlog_transaction_dependency_tracking = [WRITESET_SESSION | WRITESET]
    (if WRITESET, slave_preserve_commit_order can avoid temporary inconsistencies)
    (said otherwise, WRITESET_SESSION is mostly useful without slave_preserve_commit_order)

- To use Write Set on an Intermediate Master (even single-threaded):
  - binlog_transaction_dependency_tracking = WRITESET
    (slave_preserve_commit_order can avoid temporary inconsistencies)

- To stop using Write Set:
  - binlog_transaction_dependency_tracking = COMMIT_ORDER

- Historical note:
  - transaction_write_set_extraction = XXHASH64 not needed as default since 8.0.2
  - Setting it to OFF when not needed might save RAM

MySQL 8.0: Write Set’”

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

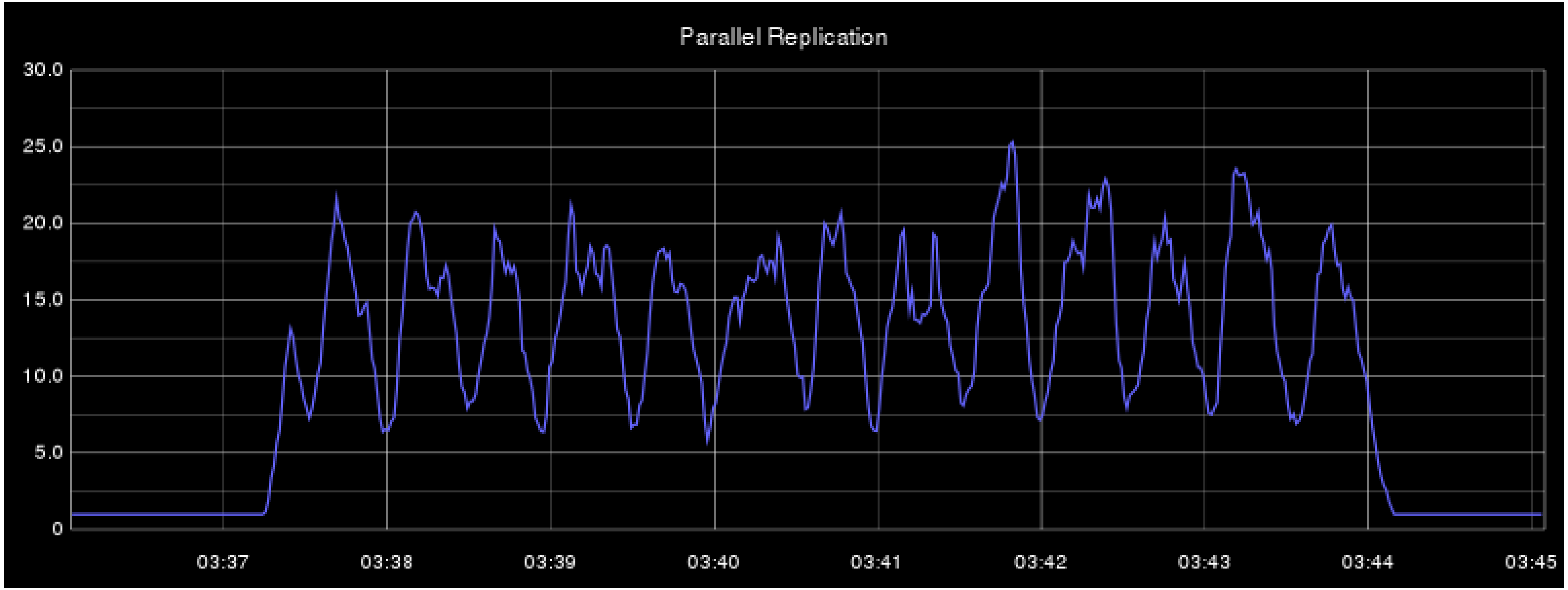
```
#170409  3:37:13  [...] last_committed=6695 sequence_number=6696 [...]
#170409  3:37:14  [...] last_committed=6696 sequence_number=6697 [...]
#170409  3:37:14  [...] last_committed=6697 sequence_number=6698 [...]
#170409  3:37:14  [...] last_committed=6698 sequence_number=6699 [...]
#170409  3:37:14  [...] 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’” ’

```
#170409  3:37:17  [...] last_committed=6700 sequence_number=6766 [...]
#170409  3:37:17  [...] last_committed=6752 sequence_number=6767 [...]
#170409  3:37:17  [...] last_committed=6753 sequence_number=6768 [...]
#170409  3:37:17  [...] last_committed=6700 sequence_number=6769 [...]
[...]
#170409  3:37:18  [...] last_committed=6700 sequence_number=6783 [...]
#170409  3:37:18  [...] last_committed=6768 sequence_number=6784 [...]
#170409  3:37:18  [...] last_committed=6784 sequence_number=6785 [...]
#170409  3:37:18  [...] last_committed=6785 sequence_number=6786 [...]
#170409  3:37:18  [...] last_committed=6785 sequence_number=6787 [...]
[...]
#170409  3:37:22  [...] last_committed=6785 sequence_number=6860 [...]
#170409  3:37:22  [...] last_committed=6842 sequence_number=6861 [...]
#170409  3:37:22  [...] last_committed=6843 sequence_number=6862 [...]
#170409  3:37:22  [...] last_committed=6785 sequence_number=6863
```

MySQL 8.0: AMIL of Write Set

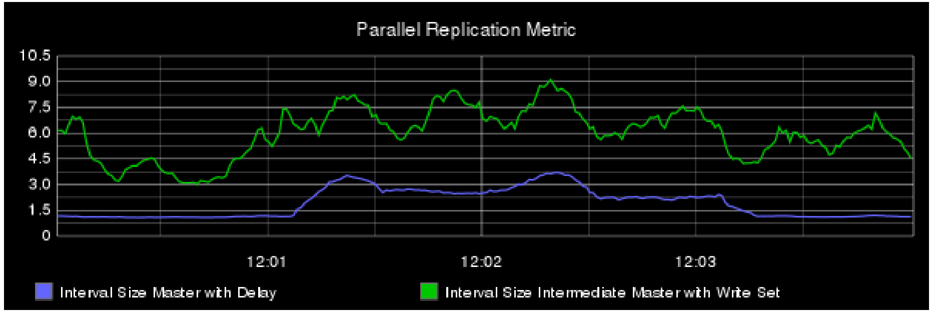
- AMIL on a single-threaded 8.0.1 Intermediate Master (*IM*) without/with Write Set:

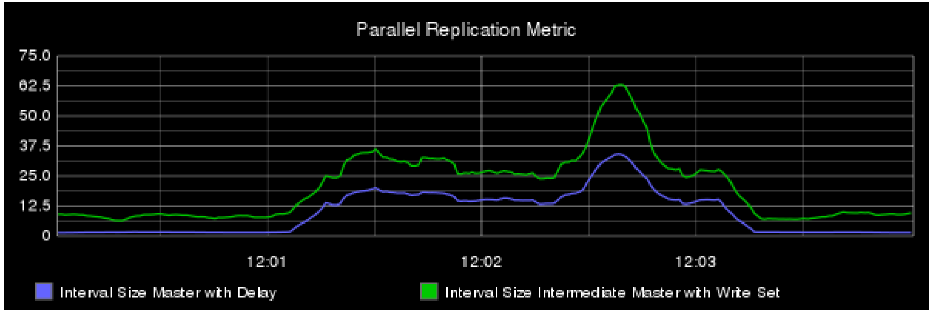


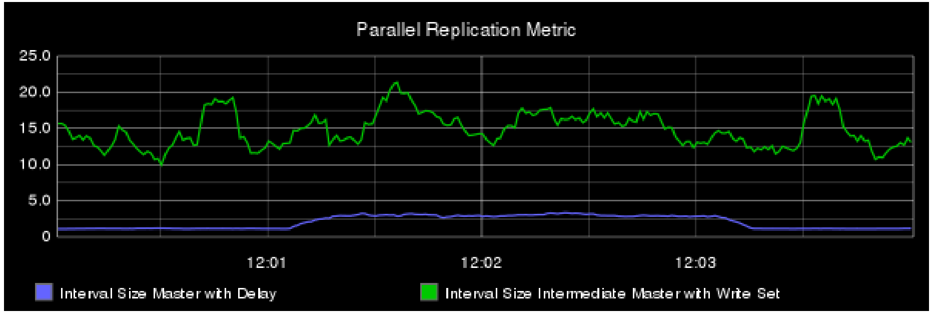
Booking.com

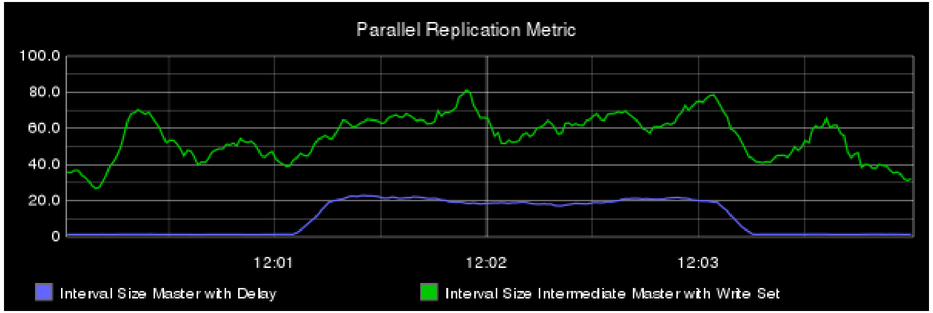
MySQL 8.0: Write Set vs Delay

- AMIL on Booking.com masters with delay vs Write Set on Intermediate Master:



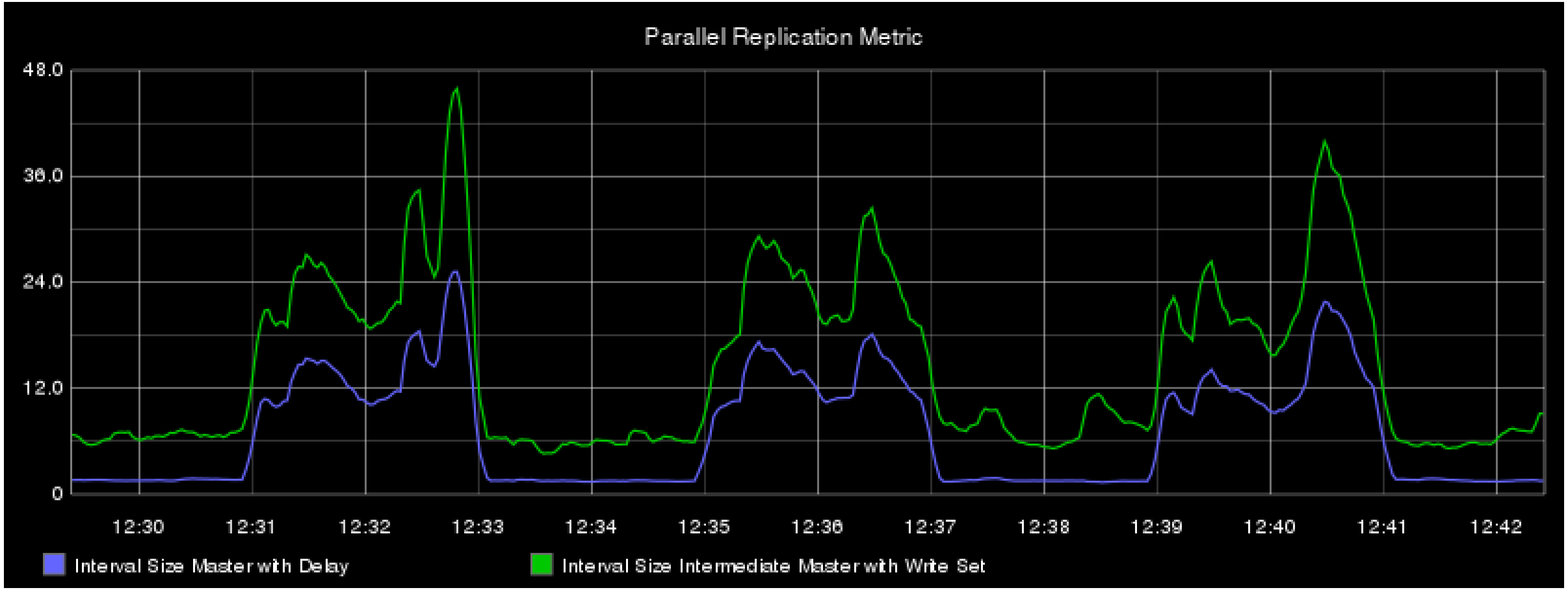






MySQL 8.0: Write Set vs Delay’

- 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'' ''

- Write Set advantages:
  - No need to slowdown the master (but could still be useful in some cases)
  - Will work even at low concurrency on the master
  - Allows to test without upgrading the master (works on an intermediate master)
    (however, this sacrifices session consistency, which might give optimistic results, unless the slave enforce commit order)
  - 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` as Bug#86063 is fixed)
- Write Set drawbacks:
  - Slowdown the master ? Consume more RAM ? Those are not big problems…
  - New technology: there could be surprises…
    (Bug#86078: https://jfg-mysql.blogspot.com/2018/01/an-update-on-write-set-parallel-replication-bug-fix-in-mysql-8-0.html)

MySQL 8.0: Write Set @ B.com

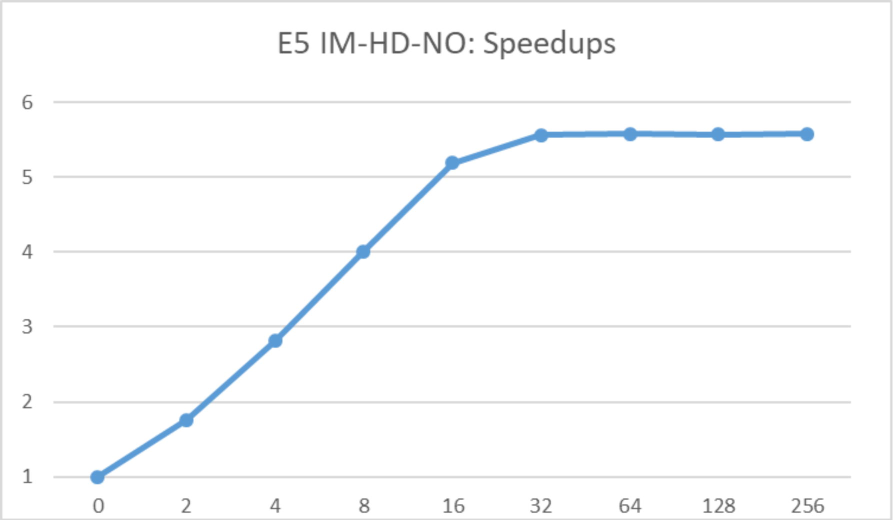
- Tests on eight (8) real Booking.com environments (different workloads):
  - A is MySQL 5.6 and 5.7 masters (1 and 7), some are SBR (4) some are RBR (4)
  - B is MySQL 8.0.3 Intermediate Master with Write Set (RBR)

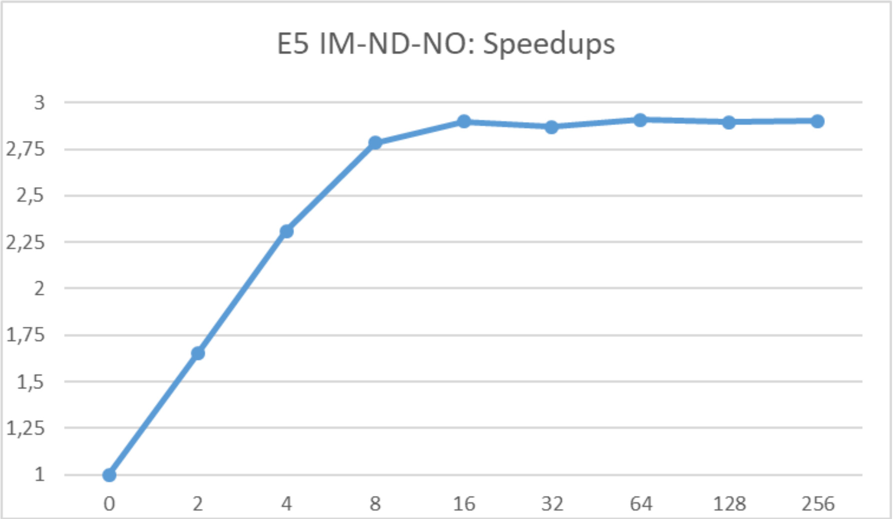
    ```
    set global transaction_write_set_extraction = XXHASH64;
    set global binlog_transaction_dependency_tracking = WRITESET;
    ```
  - C is a slave with local SSD storage

```
+---+             +---+             +---+
| A | -------->  | B | -------->  | C |
+---+             +---+             +---+
```

  - Run with 0, 2, 4, 8, 16, 32, 64, 128 and 256 workers,
    with High Durability (HD - sync_binlog = 1 & trx commit = 1) and No Durability (ND – 0 & 2),
    without and with `slave_preserve_commit_order` (NO and WO)
    with and without `log_slave_updates` (IM and SB)

MySQL 8.0: Write Set Speedups

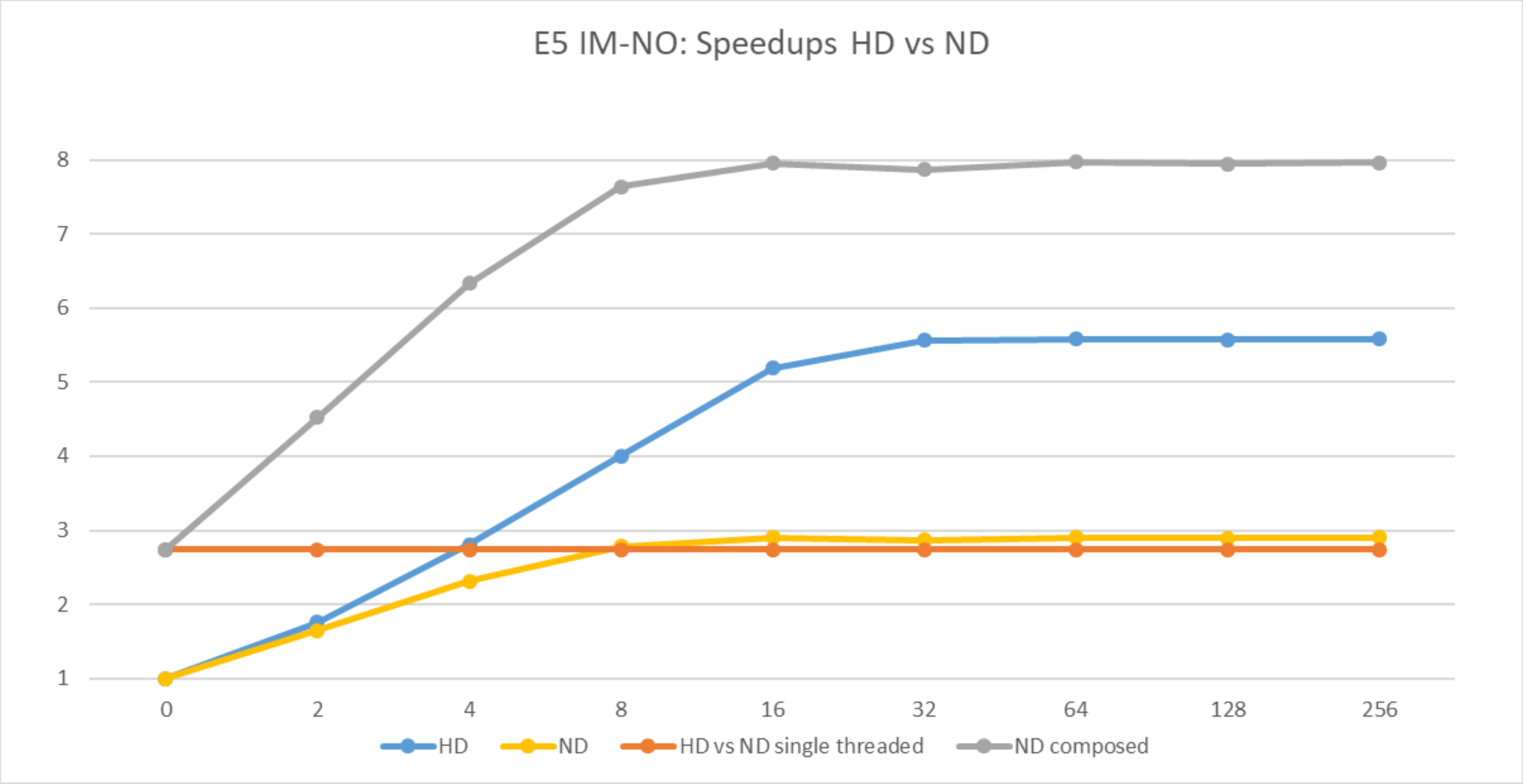




E5 IM-HD Single-Threaded: 6138 seconds

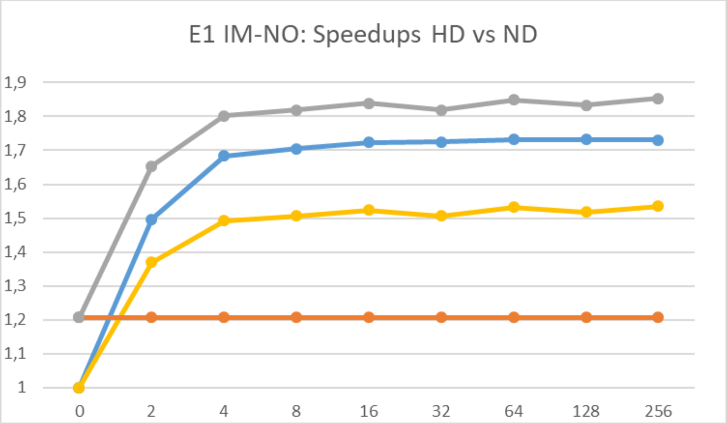
E5 IM-ND Single-Threaded: 2238 seconds

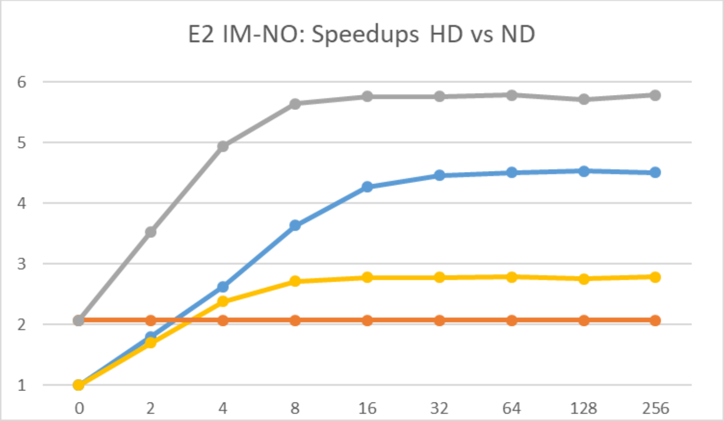
MySQL 8.0: Write Set Speedups

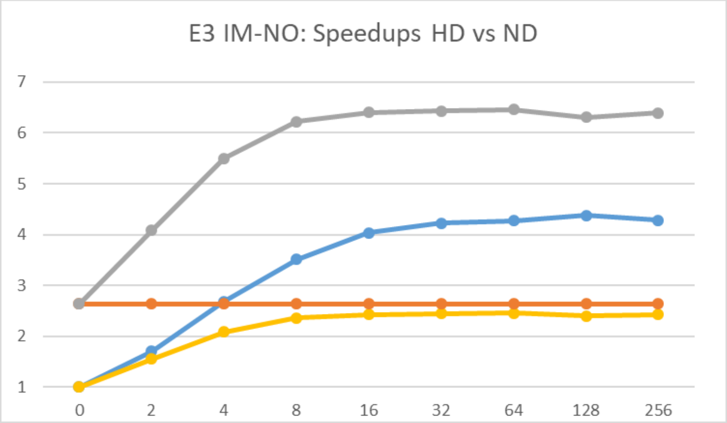


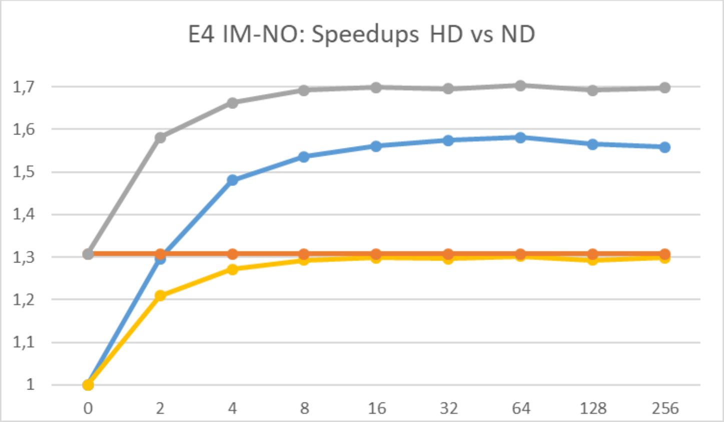
Booking.com

MySQL 8.0: Write Set Speedups’

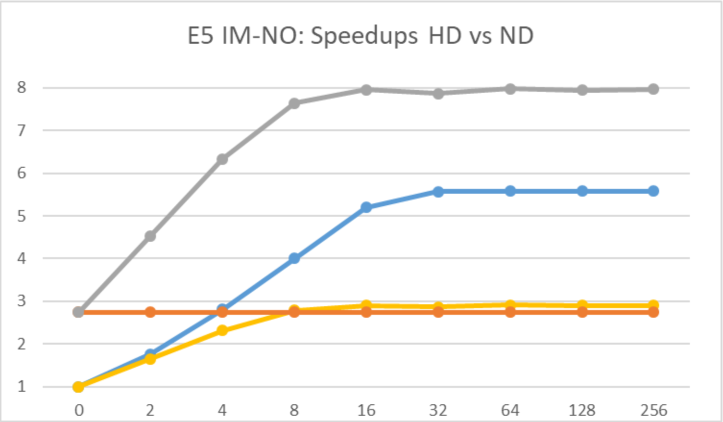

E1 IM-NO: Speedups HD vs ND

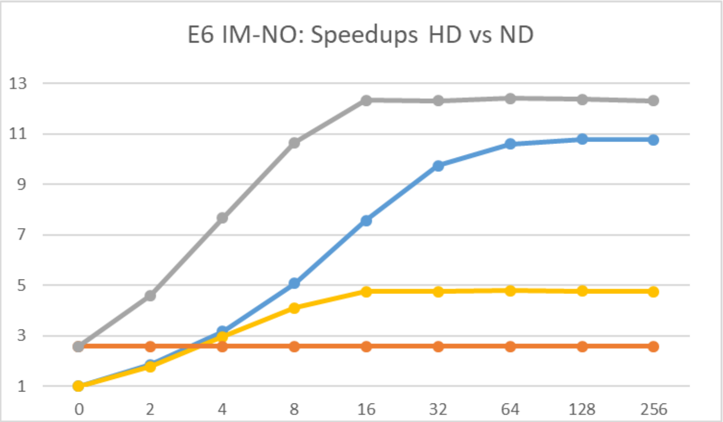

E2 IM-NO: Speedups HD vs ND

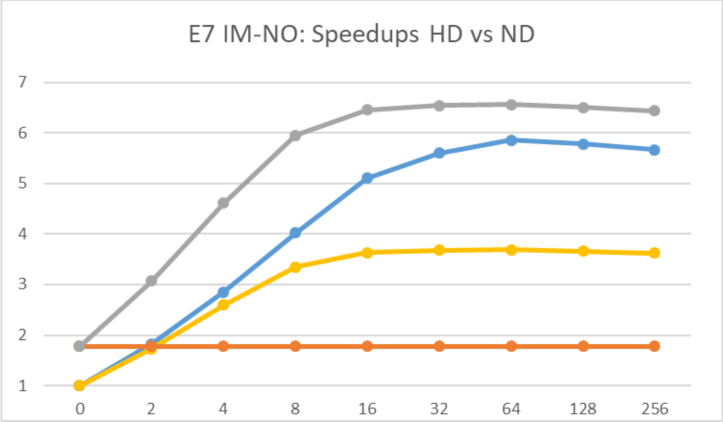

E3 IM-NO: Speedups HD vs ND

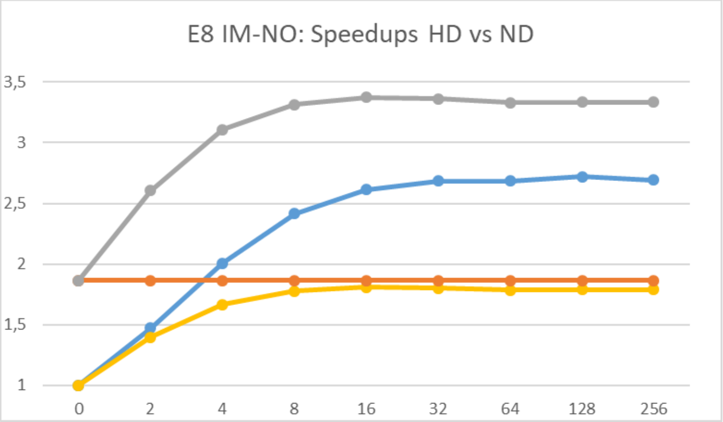

E4 IM-NO: Speedups HD vs ND

MySQL 8.0: Write Set Speedups”





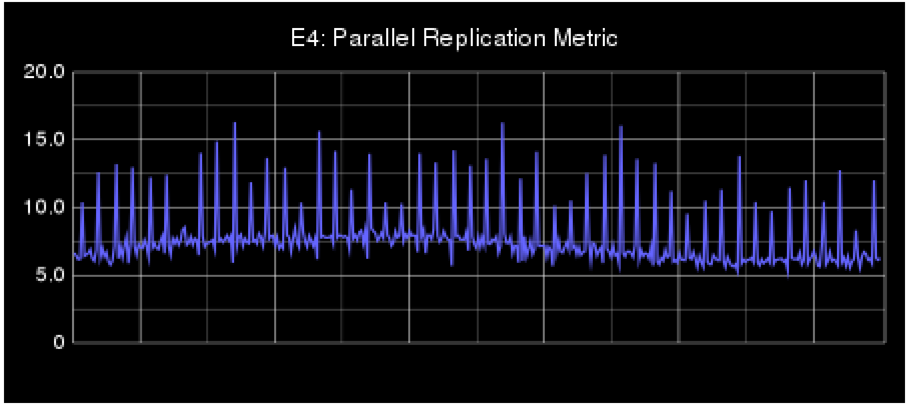


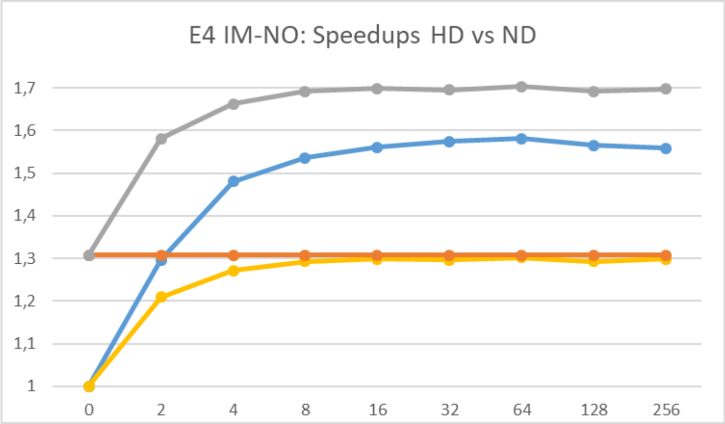


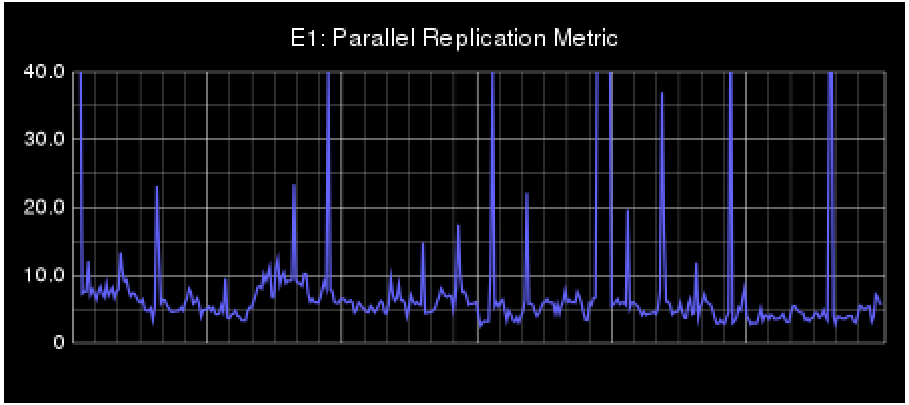
MySQL 8.0: Speedup Summary

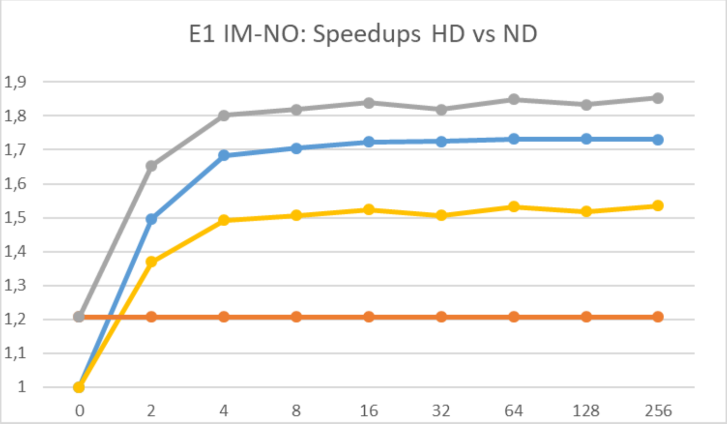
- No thrashing when too many threads !
- For the environments with High Durability:
  - Two (2) “interesting” speedups: 1.6, 1.7
  - One (1) good: 2.7
  - Four (4) very good speedups: 4.4, 4.5, 5.6, and 5.8
  - One (1) **great** speedups: 10.8 !
- For the environments without durability (ND):
  - Three (3) good speedups: 1.3, 1.5 and 1.8
  - Three (3) very good speedups: 2.4, 2.8 and 2.9
  - Two (2) **great** speedups: 3.7 and 4.8 !
- All that without tuning MySQL or the application

MySQL 8.0: Looking at low speedups

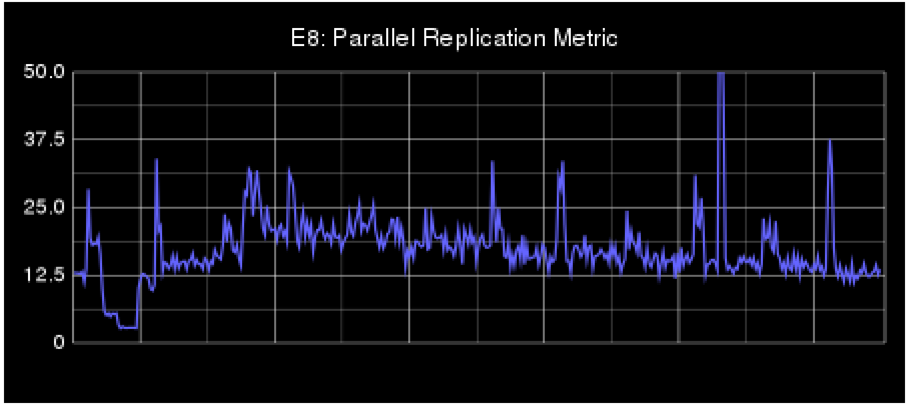


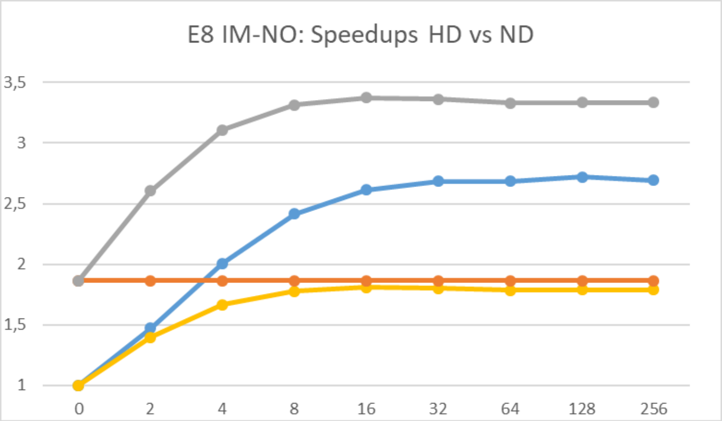




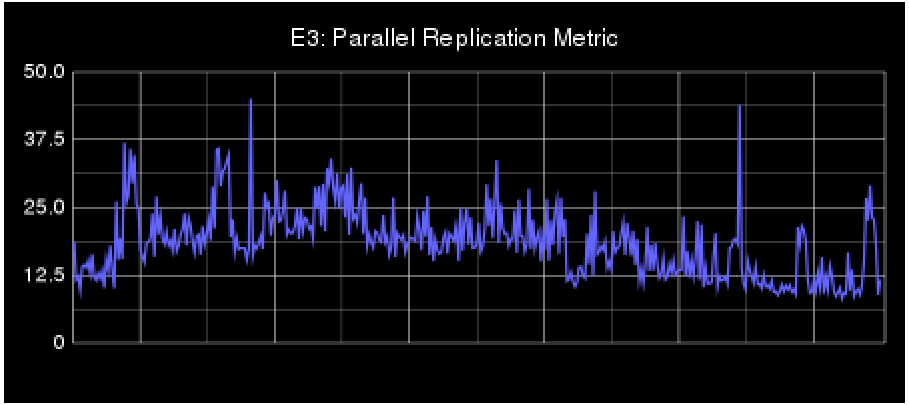


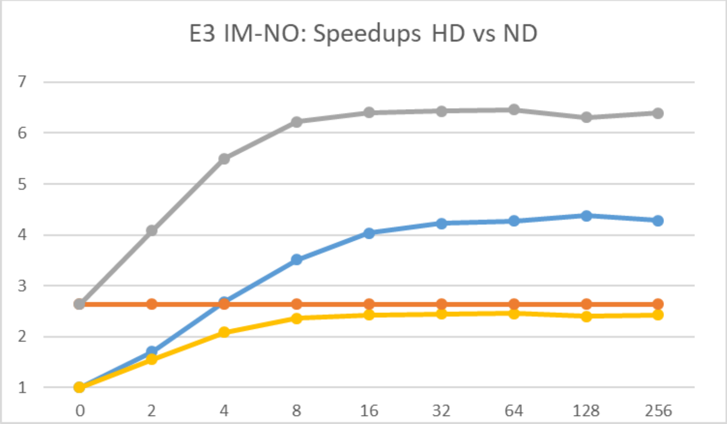
MySQL 8.0: Looking at low speedups'

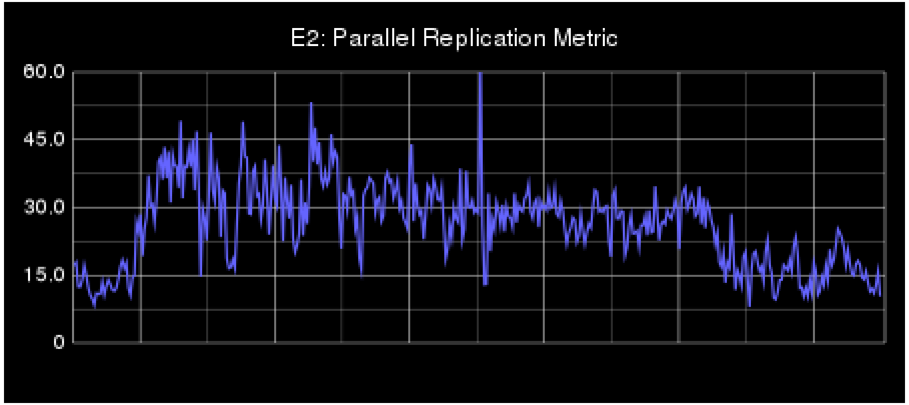


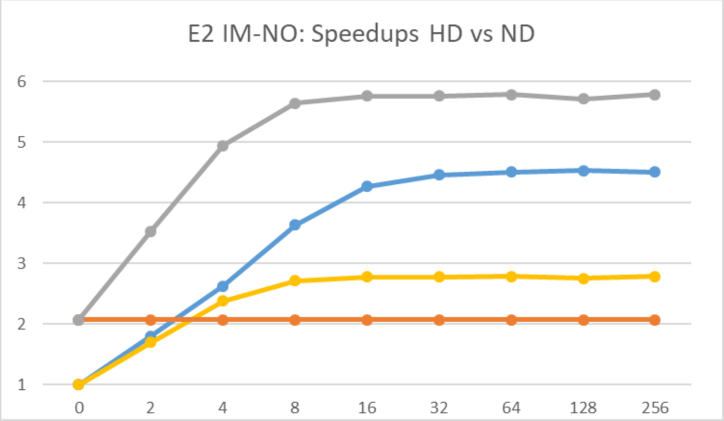


MySQL 8.0: Looking at good speedups

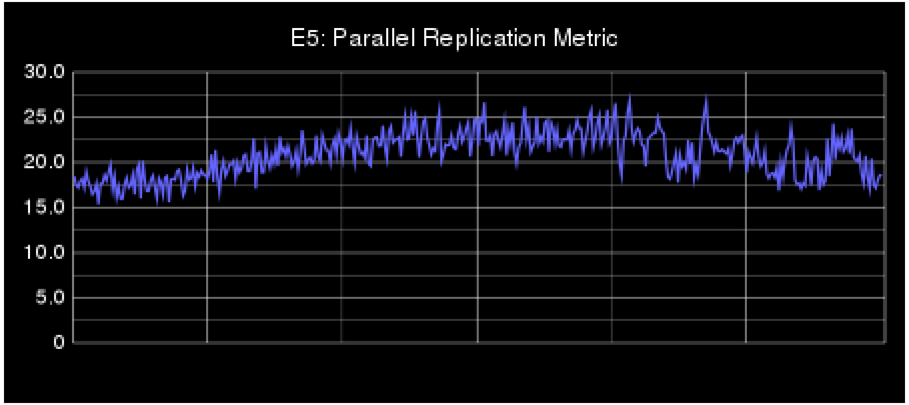


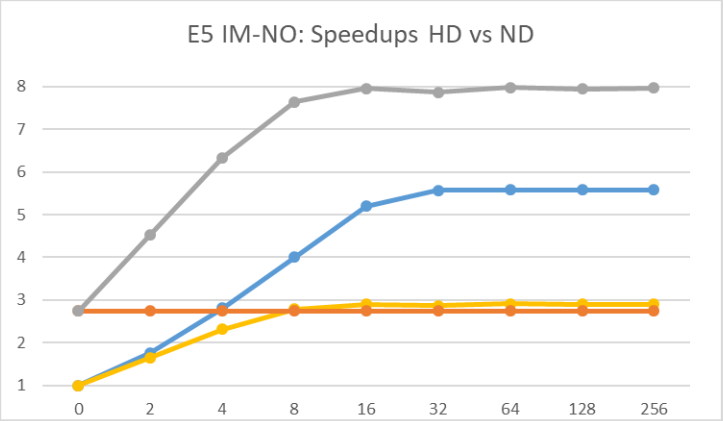


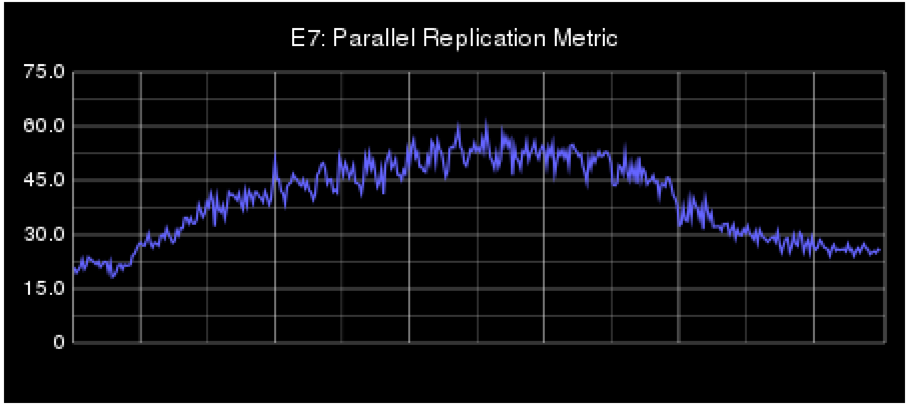


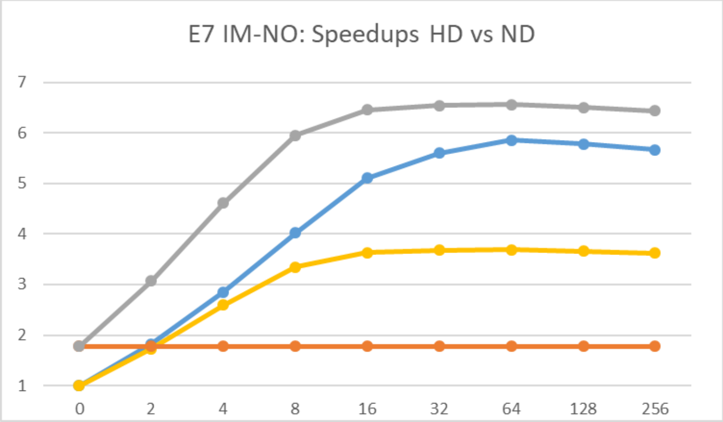


MySQL 8.0: Looking at good speedups’

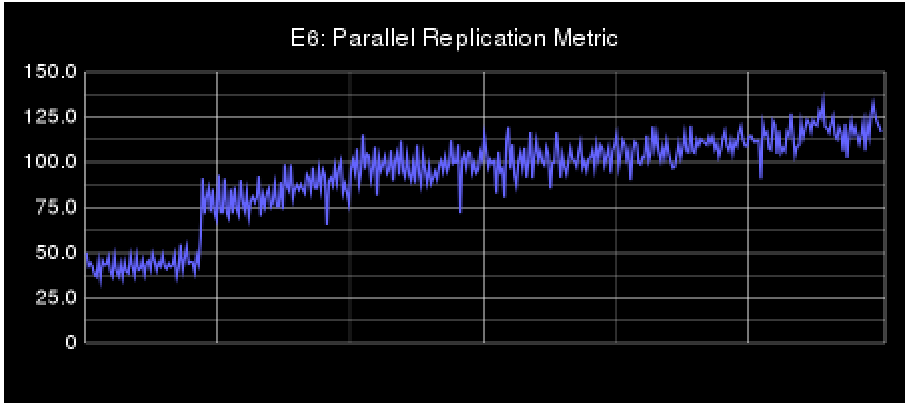


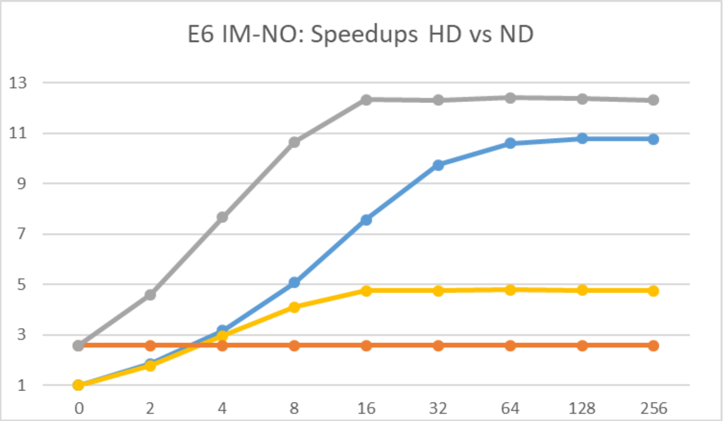




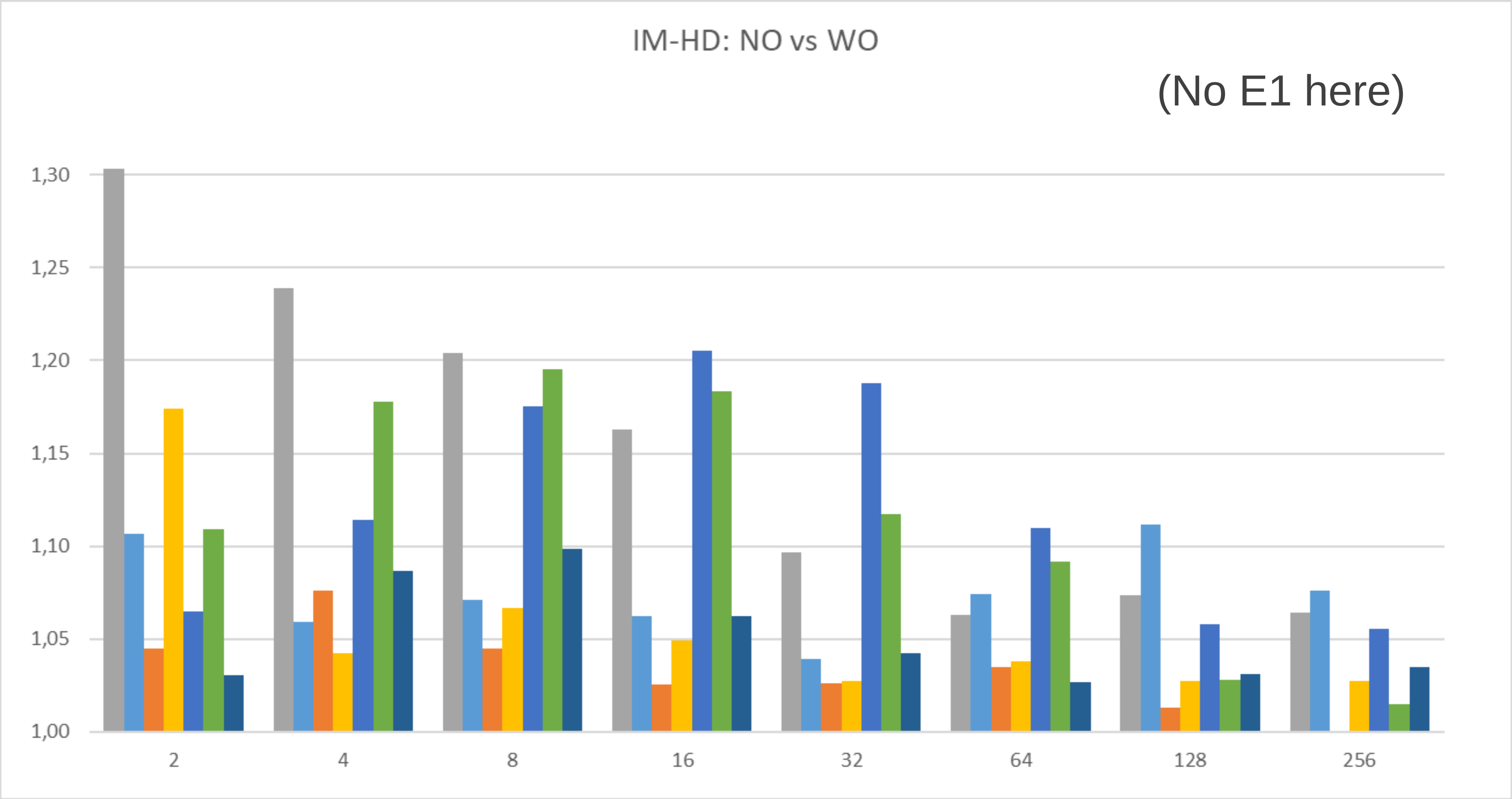


MySQL 8.0: Looking at great speedups



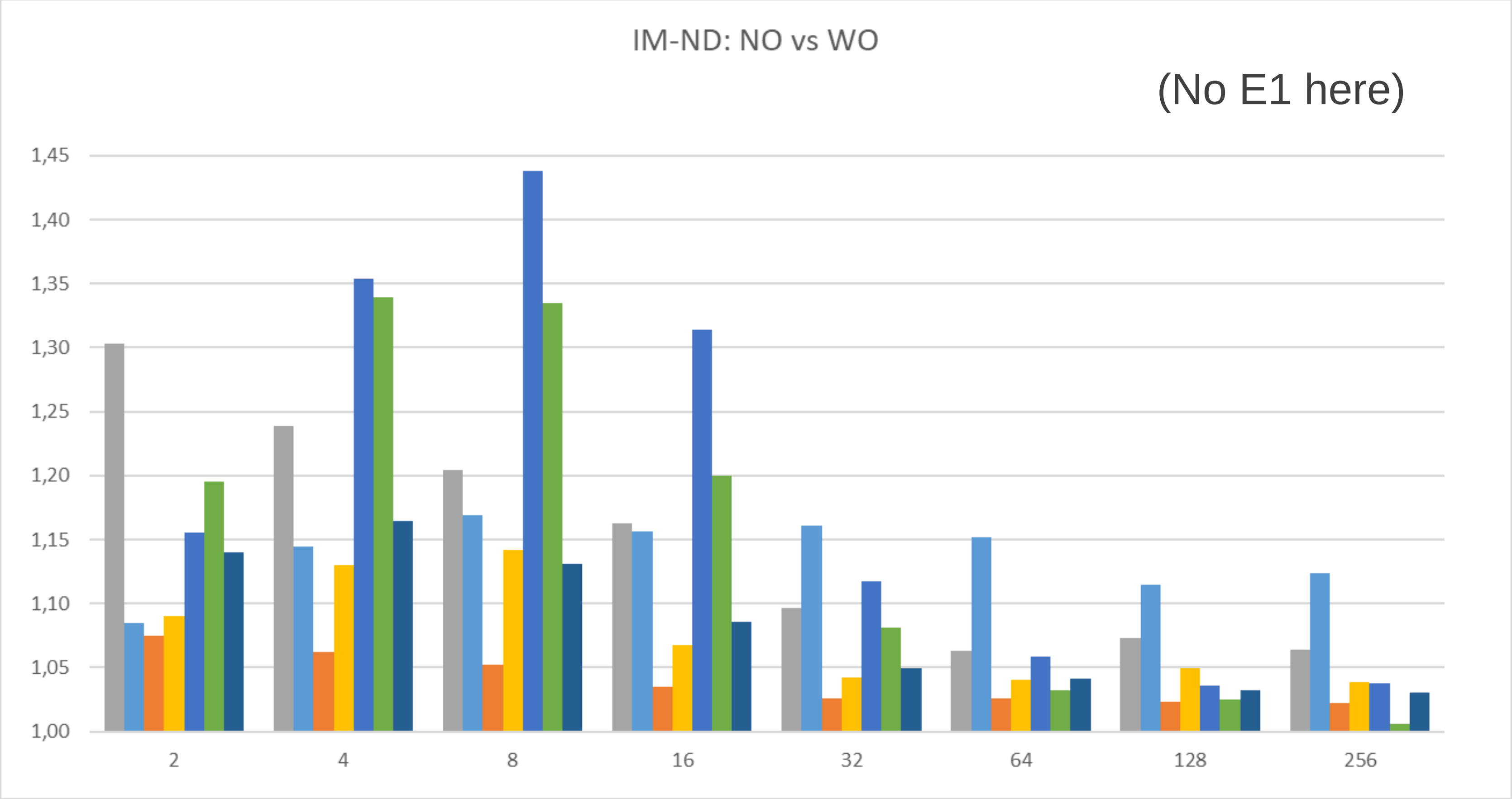


MySQL 8.0: What about commit order ?



(No E1 here)

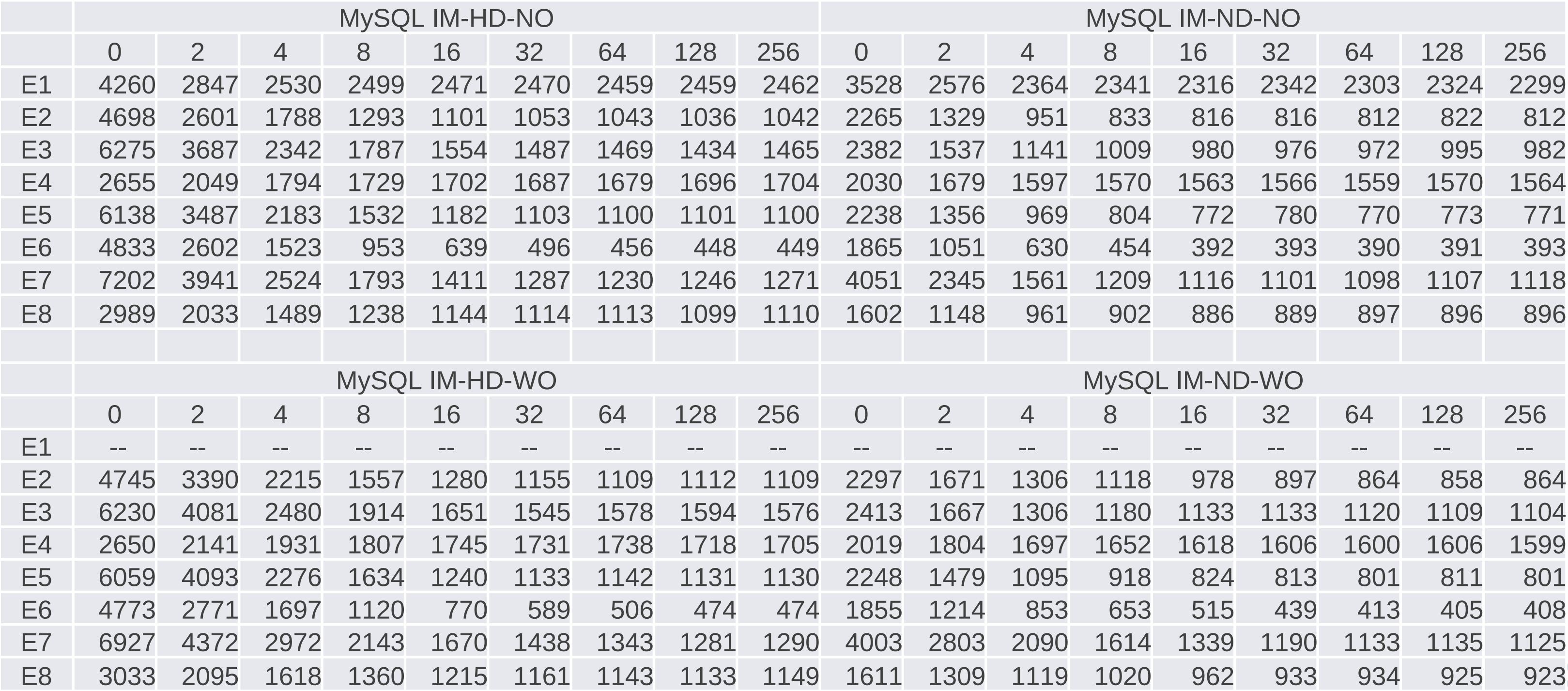
MySQL 8.0: What about commit order ?’



(No E1 here)

MySQL 8.0.3: benchmarks numbers

- IM: Intermediate Master; {H,N}D: {High,No} Durability; {N,W}O: {No,With} Order



	MySQL IM-HD-NO									MySQL IM-ND-NO								
	0	2	4	8	16	32	64	128	256	0	2	4	8	16	32	64	128	256
E1	4260	2847	2530	2499	2471	2470	2459	2459	2462	3528	2576	2364	2341	2316	2342	2303	2324	2299
E2	4698	2601	1788	1293	1101	1053	1043	1036	1042	2265	1329	951	833	816	816	812	822	812
E3	6275	3687	2342	1787	1554	1487	1469	1434	1465	2382	1537	1141	1009	980	976	972	995	982
E4	2655	2049	1794	1729	1702	1687	1679	1696	1704	2030	1679	1597	1570	1563	1566	1559	1570	1564
E5	6138	3487	2183	1532	1182	1103	1100	1101	1100	2238	1356	969	804	772	780	770	773	771
E6	4833	2602	1523	953	639	496	456	448	449	1865	1051	630	454	392	393	390	391	393
E7	7202	3941	2524	1793	1411	1287	1230	1246	1271	4051	2345	1561	1209	1116	1101	1098	1107	1118
E8	2989	2033	1489	1238	1144	1114	1113	1099	1110	1602	1148	961	902	886	889	897	896	896
	MySQL IM-HD-WO									MySQL IM-ND-WO								
	0	2	4	8	16	32	64	128	256	0	2	4	8	16	32	64	128	256
E1	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
E2	4745	3390	2215	1557	1280	1155	1109	1112	1109	2297	1671	1306	1118	978	897	864	858	864
E3	6230	4081	2480	1914	1651	1545	1578	1594	1576	2413	1667	1306	1180	1133	1133	1120	1109	1104
E4	2650	2141	1931	1807	1745	1731	1738	1718	1705	2019	1804	1697	1652	1618	1606	1600	1606	1599
E5	6059	4093	2276	1634	1240	1133	1142	1131	1130	2248	1479	1095	918	824	813	801	811	801
E6	4773	2771	1697	1120	770	589	506	474	474	1855	1214	853	653	515	439	413	405	408
E7	6927	4372	2972	2143	1670	1438	1343	1281	1290	4003	2803	2090	1614	1339	1190	1133	1135	1125
E8	3033	2095	1618	1360	1215	1161	1143	1133	1149	1611	1309	1119	1020	962	933	934	925	923

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

(http://jfg-mysql.blogspot.com/2018/01/write-set-in-mysql-5-7-group-replication.html)

Write Set in 5.7.22

- From MySQL 5.7.22 release notes (2018-04-19):

  It is now possible to specify whether information written into the binary log enables replication slaves to parallelize based on commit timestamps, or on transaction write sets.

- So Write Set has been back-ported from 8.0 to 5.7 !

➢ All the good things about Write Set in 8.0 is available from 5.7.22

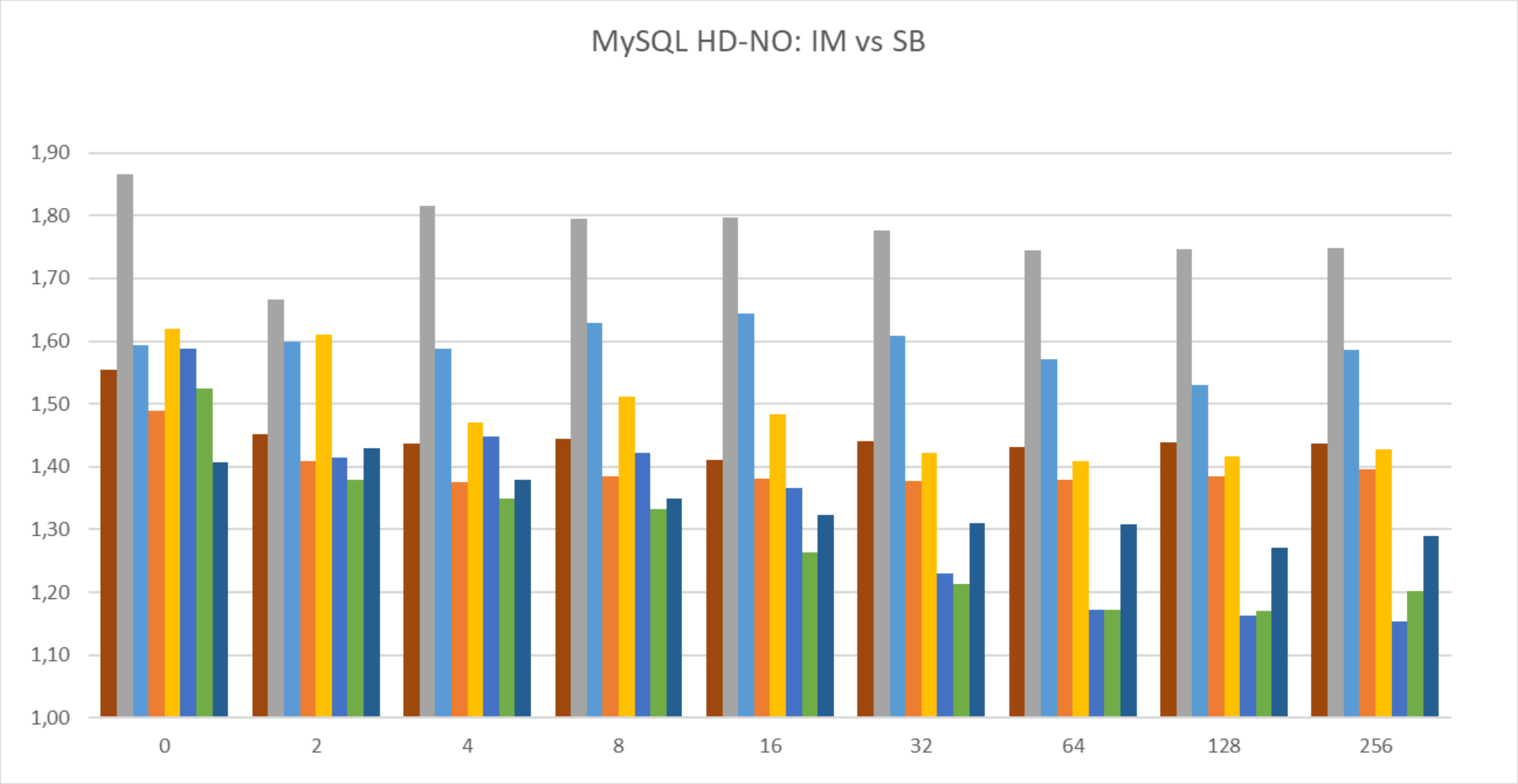
- To enable Write Set parallelism identification in 5.7.22:
  - `transaction_write_set_extraction = XXHASH64` (default OFF)
  - `binlog_transaction_dependency_tracking = [ WRITESET_SESSION | WRITESET ]`

Other ways of improving replication speed

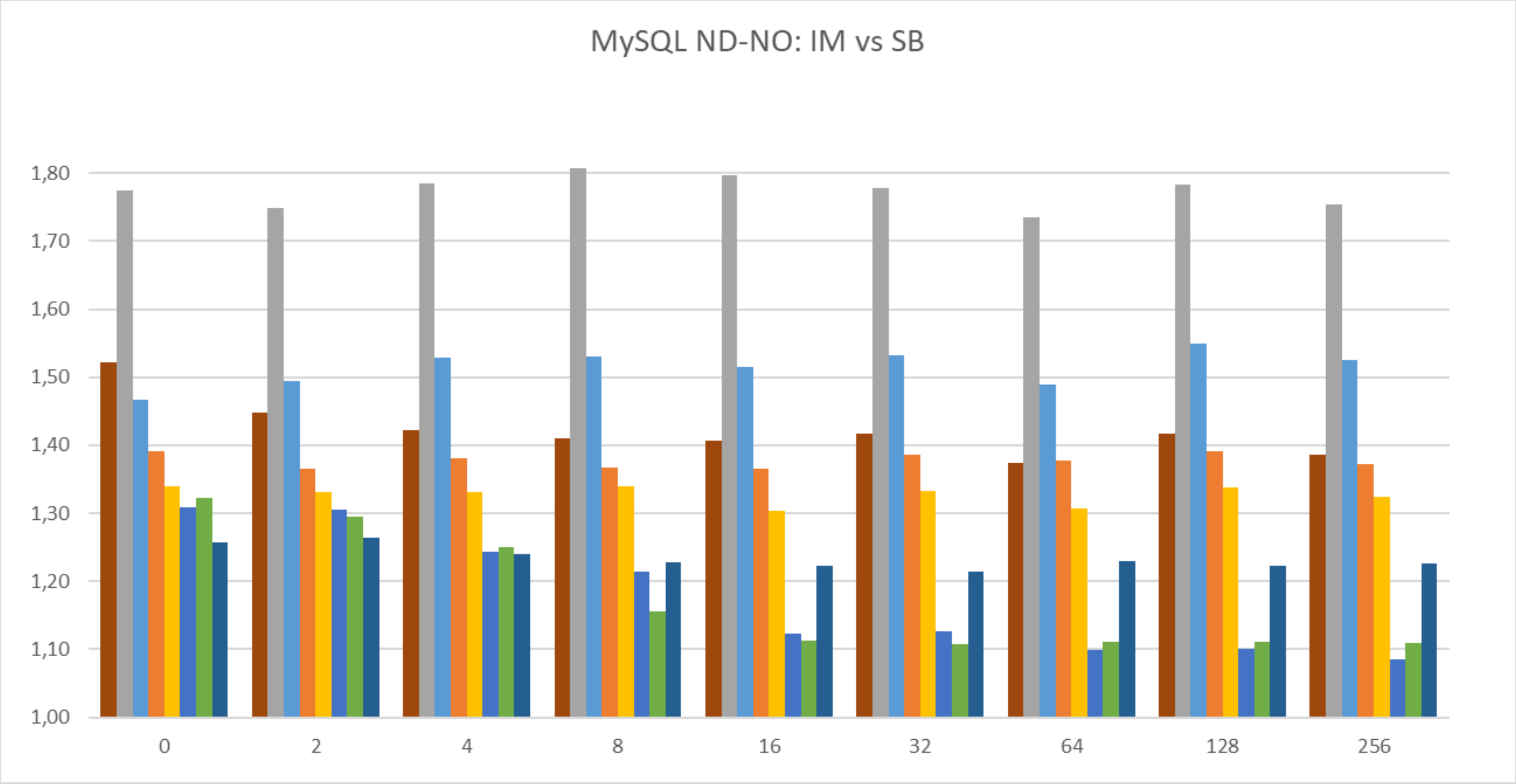
- Optimizing schema and queries was always a way
- Reducing durability was also a well known way

- Enabling Parallel Replication is a more recent way
- But disabling binlogs on slaves really is the thing !

MySQL: no binlogs on slaves



MySQL: no binlogs on slaves’

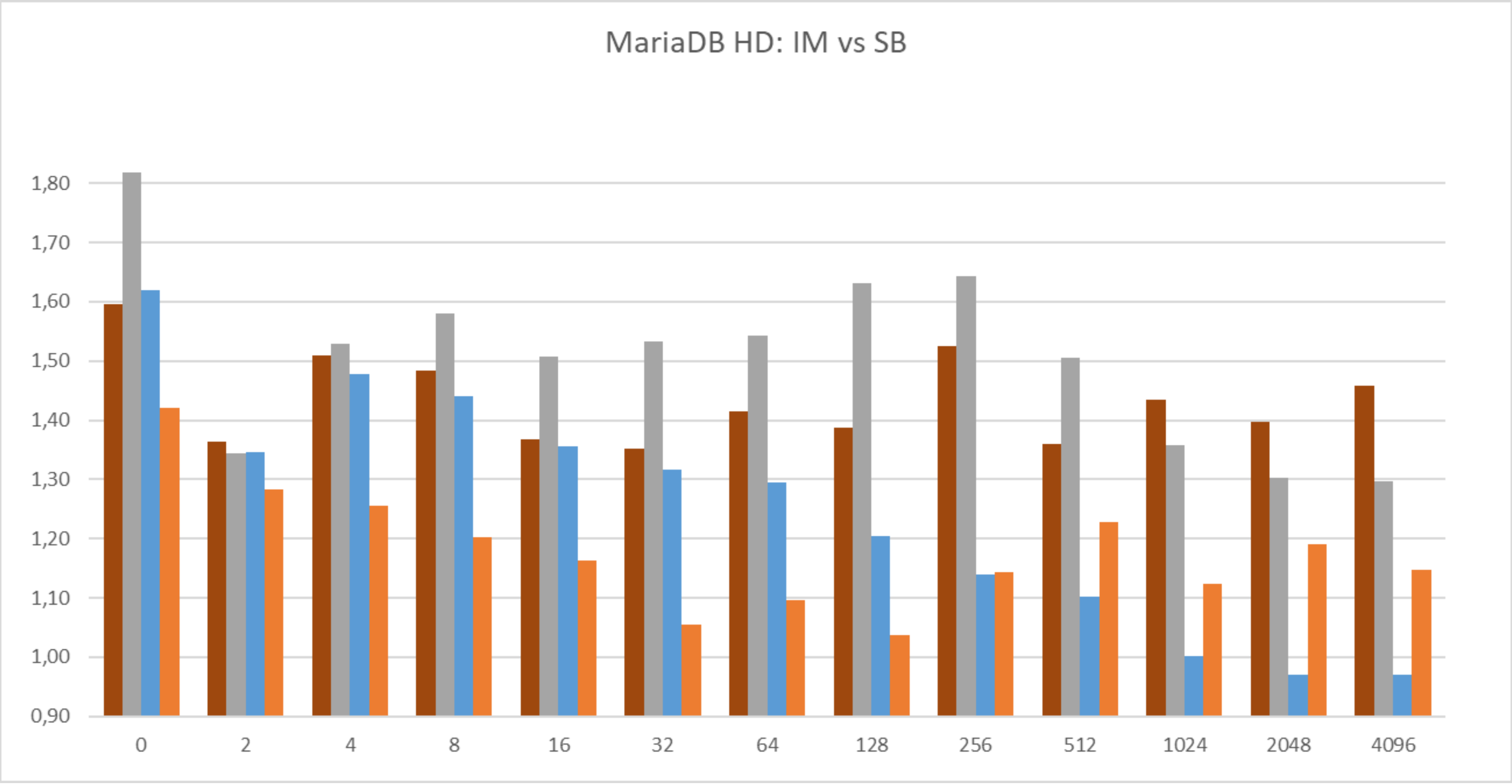


MySQL 8.0.3: benchmarks numbers’

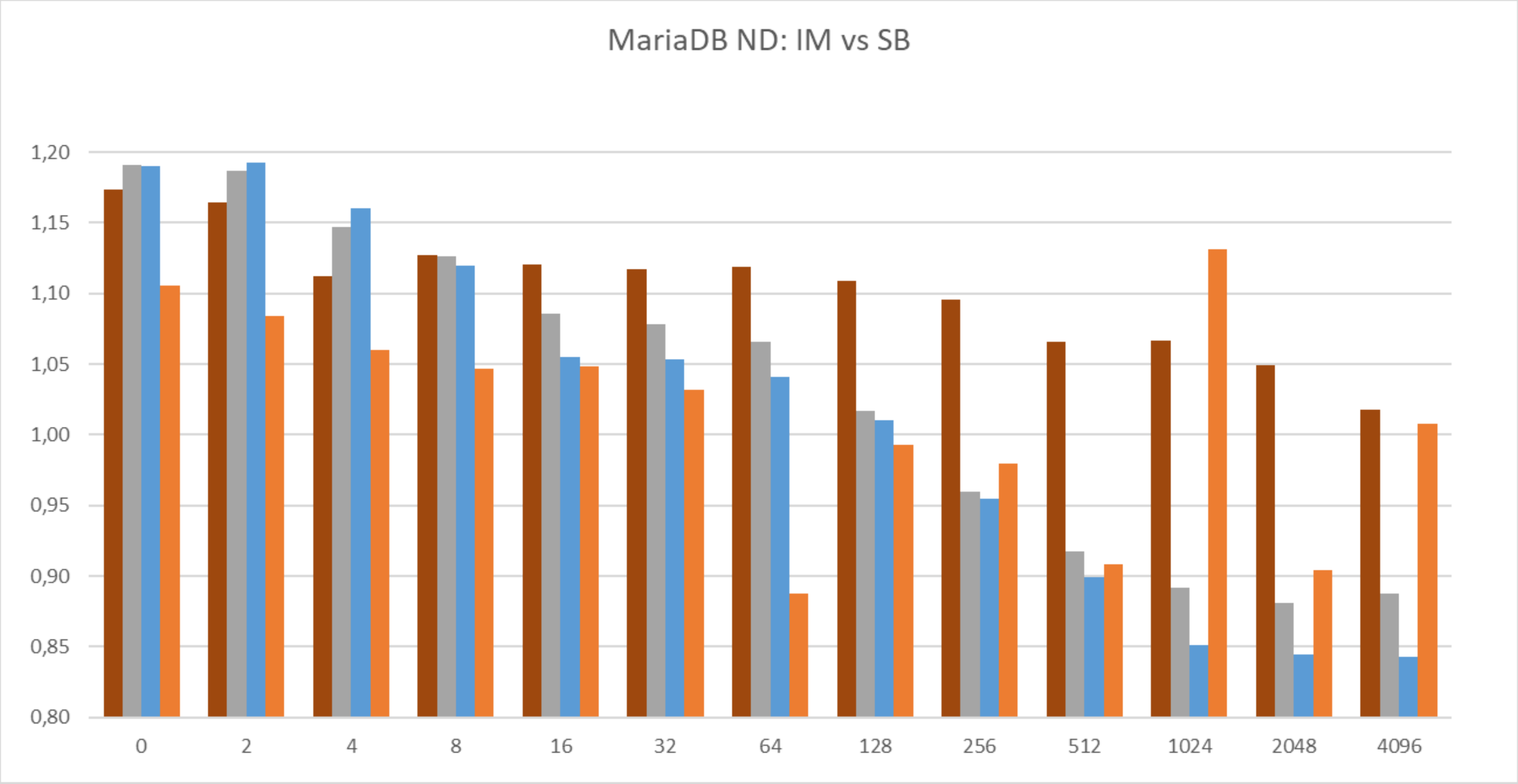
- SB: Slave with Binlogs (but without log-slave-updates); only NO because Bug#75396

	MySQL IM-HD-NO									MySQL IM-ND-NO								
	0	2	4	8	16	32	64	128	256	0	2	4	8	16	32	64	128	256
E1	4260	2847	2530	2499	2471	2470	2459	2459	2462	3528	2576	2364	2341	2316	2342	2303	2324	2299
E2	4698	2601	1788	1293	1101	1053	1043	1036	1042	2265	1329	951	833	816	816	812	822	812
E3	6275	3687	2342	1787	1554	1487	1469	1434	1465	2382	1537	1141	1009	980	976	972	995	982
E4	2655	2049	1794	1729	1702	1687	1679	1696	1704	2030	1679	1597	1570	1563	1566	1559	1570	1564
E5	6138	3487	2183	1532	1182	1103	1100	1101	1100	2238	1356	969	804	772	780	770	773	771
E6	4833	2602	1523	953	639	496	456	448	449	1865	1051	630	454	392	393	390	391	393
E7	7202	3941	2524	1793	1411	1287	1230	1246	1271	4051	2345	1561	1209	1116	1101	1098	1107	1118
E8	2989	2033	1489	1238	1144	1114	1113	1099	1110	1602	1148	961	902	886	889	897	896	896
	MySQL SB-HD-NO									MySQL SB-ND-NO								
	0	2	4	8	16	32	64	128	256	0	2	4	8	16	32	64	128	256
E1	2742	1961	1760	1731	1751	1714	1718	1709	1714	2317	1778	1662	1660	1647	1652	1675	1640	1658
E2	2517	1561	985	720	613	593	598	593	596	1276	760	533	461	454	459	468	461	463
E3	3940	2306	1475	1097	945	924	935	937	924	1623	1029	746	659	647	637	653	642	644
E4	1784	1454	1304	1249	1232	1225	1217	1224	1221	1459	1229	1157	1148	1144	1130	1132	1128	1140
E5	3790	2166	1484	1013	797	776	781	777	771	1670	1019	728	600	592	585	589	578	582
E6	3042	1839	1052	670	468	403	389	385	389	1425	805	507	374	349	349	355	355	362
E7	4724	2856	1871	1345	1116	1060	1050	1065	1058	3064	1810	1249	1046	1003	994	989	997	1008
E8	2124	1422	1079	918	865	850	851	865	861	1274	908	775	735	725	732	729	733	731

MariaDB: no binlogs on slaves



MariaDB: no binlogs on slaves’



MariaDB 10.2.12: benchmarks numbers

- SB: Slave with Binlogs (but without log-slave-updates); only NO because Bug#75396

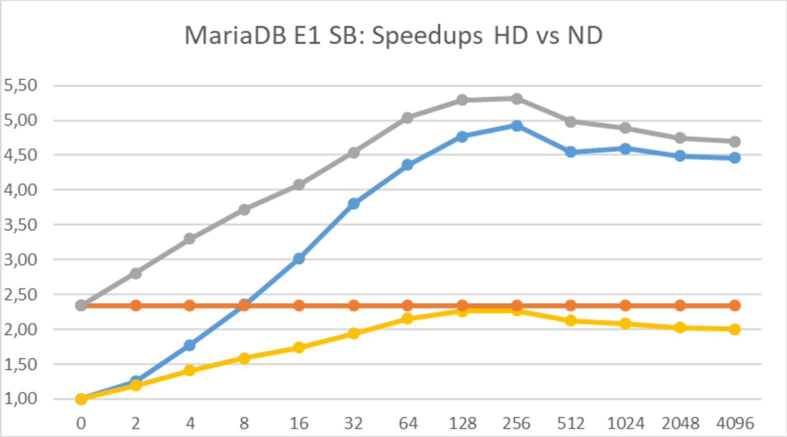
	MariaDB IM-HD													MariaDB IM-ND												
	0	2	4	8	16	32	64	128	256	512	1024	2048	4096	0	2	4	8	16	32	64	128	256	512	1024	2048	4096
E1	5063	3452	2700	2001	1440	1127	1028	923	982	948	990	986	1036	1589	1316	1068	960	873	781	704	664	654	678	692	701	687
E2	7896	4221	2819	1865	1301	1024	926	935	989	1126	1476	1711	1770	1589	1136	860	730	697	664	614	593	620	741	1007	1188	1230
E3	4770	2745	1762	1083	690	494	400	348	361	430	479	477	491	1228	856	616	478	382	336	306	302	316	367	418	424	436
E4	5864	3727	2547	1804	1329	1008	962	849	893	917	933	880	861	2707	2060	1620	1323	1053	906	790	875	760	672	802	720	794
	MariaDB SB-HD													MariaDB SB-ND												
	0	2	4	8	16	32	64	128	256	512	1024	2048	4096	0	2	4	8	16	32	64	128	256	512	1024	2048	4096
E1	3171	2532	1789	1348	1053	834	727	665	644	697	690	706	711	1354	1130	960	852	779	699	629	599	597	636	649	668	675
E2	4341	3143	1843	1181	863	668	600	573	602	748	1087	1313	1365	1334	957	750	648	642	616	576	583	646	808	1129	1349	1386
E3	2945	2041	1193	752	509	375	309	289	317	390	478	492	506	1032	718	531	427	362	319	294	299	331	408	491	502	517
E4	4129	2906	2028	1500	1142	956	878	818	781	747	831	739	751	2449	1900	1528	1264	1004	878	890	881	776	740	709	796	788

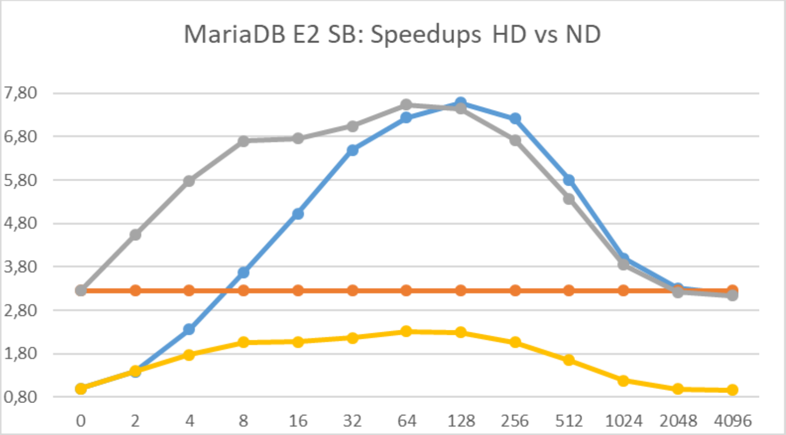
No binlogs on slaves

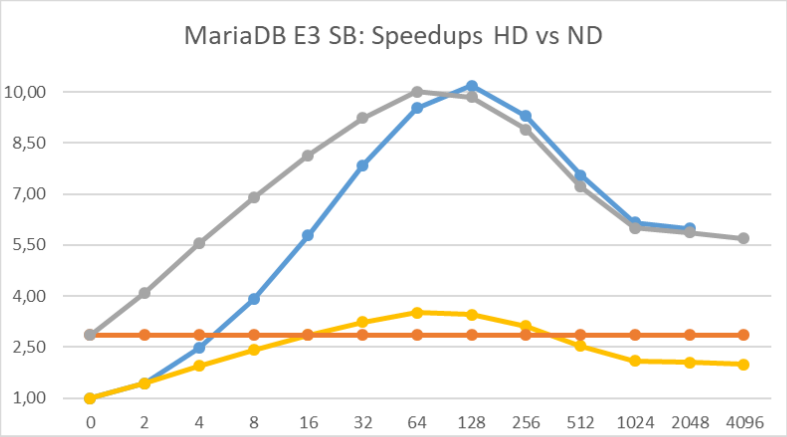
- Disabling `log-slave-updates` is a great way to get faster replication
- But by doing so, “leveling” slaves after the failure of the master is impossible because we have no copies of the binary logs on laves
- Can we have our cake and eat it too ?

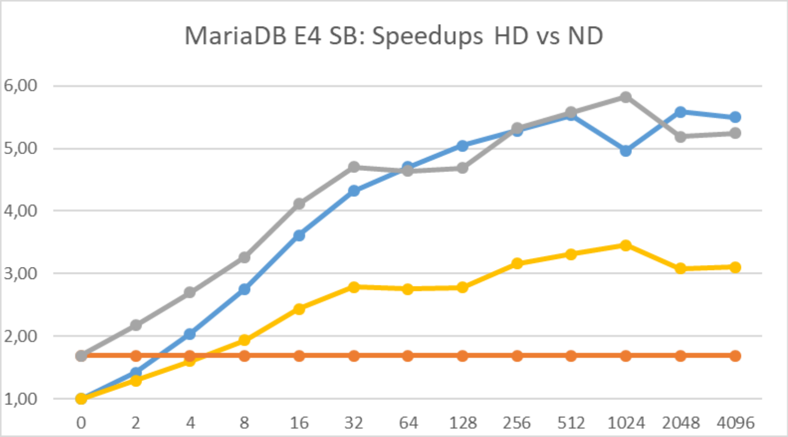
- The solution is Binlog Servers !

Reducing durability is not needed


MariaDB E1 SB: Speedups HD vs ND


MariaDB E2 SB: Speedups HD vs ND


MariaDB E3 SB: Speedups HD vs ND


MariaDB E4 SB: Speedups HD vs ND

Parallel 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

MySQL 5.7 and 8.0 // Repl. Summary

- Parallel replication in MySQL 5.7 is not simple:
  - Need precise tuning
  - Long transactions block the parallel replication pipeline
  - Care about Intermediate masters

- Write Set in MySQL 8.0 gives very interesting results:
  - No problem with Intermediate masters
  - Allows to test with Intermediate Master
  - Some great speedups and most of them very good

- Write Set from MySQL 8.0 has been backported in 5.7.22:
  - Can benefit from this new technology without a major version upgrade !

And please
test by yourself
and share results

Parallel 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
- A Metric for Tuning Parallel Replication in MySQL 5.7 https://jfg-mysql.blogspot.com/2017/02/metric-for-tuning-parallel-replication-mysql-5-7.html

- An update on Write Set (parallel replication) bug fix in MySQL 8.0 https://jfg-mysql.blogspot.com/2018/01/an-update-on-write-set-parallel-replication-bug-fix-in-mysql-8-0.html
- Write Set in MySQL 5.7: Group Replication https://jfg-mysql.blogspot.com/2018/01/write-set-in-mysql-5-7-group-replication.html
- More Write Set in MySQL: Group Replication Certification https://jfg-mysql.blogspot.com/2018/01/more-write-set-in-mysql-5-7-group-replication-certification.html

Parallel Replication: Links’

- Solving MySQL Replication Lag with LOGICAL_CLOCK and Calibrated Delay
  https://www.vividcortex.com/blog/solving-mysql-replication-lag-with-logical_clock-and-calibrated-delay
- How to Fix a Lagging MySQL Replication
  https://thoughts.t37.net/fixing-a-very-lagging-mysql-replication-db6eb5a6e15d

- Binlog Servers:
  - http://blog.booking.com/mysql_slave_scaling_and_more.html
  - 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/
  - https://www.percona.com/blog/2016/02/10/estimating-potential-for-mysql-5-7-parallel-replication/
  - http://mysqlhighavailability.com/mysql-replication-defaults-after-5-7/

Parallel Replication: Links’’

- Bugs:
  - The doc. of slave-parallel-type=LOGICAL_CLOCK wrongly reference Group Commit: Bug#85977
  - MTS with slave_preserve_commit_order not repl. crash safe: Bug#80103 & Bug#81840
  - Deadlock with slave_preserve_commit_order=ON with Bug#86078: Bug#86079 & Bug#89247

  - Add statuses about optimistic parallel replication stalls: MDEV-10664
  - Wrong Seconds_Behind_Master when only starting the SQL_Thread: MDEV-15010
  - SHOW GLOBAL STATUS can deadlock Optimistic Parallel Replication: MDEV-15135
  - Optimistic parallel slave doesn't play well with START SLAVE UNTIL: MDEV-15152
  - MariaDB sometimes crashes on rollback of a transaction with BLOBs: MDEV-15608

Parallel Replication: Links’’’

- Feature requests
  - Avoid overloading the master on relay-log-recovery: Bug#74323
    https://blog.booking.com/better_crash_safe_replication_for_mysql.html
  - Allow slave_preserve_commit_order without log-slave-updates: Bug#75396
  - Expose, on the master/slave, counters for monitoring // info. quality: Bug#85965 & Bug#85966
  - Expose counters for monitoring Write Set barriers: Bug#86060
  - Avoid overloading the master on restarting IO_THREAD: MDEV-8945
    http://jfg-mysql.blogspot.com/2015/10/bad-commands-with-mariadb-gtids-2.html

Parallel Replication: Links’’’ ’

- Fixed bugs:
  - MariaDB deadlocks: MDEV-7326, MDEV-7458, MDEV-10644
  - MariaDB “rewind logic”: MDEV-6589, MDEV-9138 and MDEV-10863
  - Full table scan bug in InnoDB: MDEV-10649, Bug#82968 and Bug#82969
    If you ever noted that your single line UPDATEs by PK worked for a long time, check this:
    https://www.facebook.com/valerii.kravchuk/posts/1073608056064467
  - Message after MTS crash misleading: Bug#80102 (and Bug#77496)
  - Replication position lost after crash on MTS configured slave: Bug#77496
  - 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

Thanks

Eduardo Ortega (MySQL Database Engineer)
eduardo DOT ortega AT booking.com

Jean-François Gagné (System Engineer)
jeanfrancois DOT gagne AT booking.com