Booking.com

MySQL Parallel Replication: inventory, use-cases and limitations

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

Presented at FOSDEM 2016

Booking.com

Booking.com

- Based in Amsterdam since 1996
- Online Hotel and Accommodation Agent:
  - +855.000 properties in 221 countries
  - 42 languages (website and customer service)
  - +21.000.000 bookable rooms
- Part of the Priceline Group
- And we use MySQL:
  - Thousands (1000s) of servers, ~85% replicating
  - >130 masters: ~25 >50 slaves & ~8 >100 slaves

Session Summary

1. Replication Reminders
2. Introducing Parallel Replication
3. MySQL 5.6: schema based
   MariaDB 10.0: out-of-order and in-order
   MariaDB 10.1: +optimistic
   MySQL 5.7: +logical clock
4. Benchmark Results from Booking.com

Replication: Reminders

- Replication: one master / one or more slaves

- The master records all 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(s)
  - Could produce binary logs to be itself a master (log-slave-updates)

- Replication is:
  - Asynchronous → lag
  - Was single-threaded → slower than the master (before MySQL 5.6 and MariaDB 10.0)
  - Was single-master (before MySQL 5.7 and MariaDB 10.0)

// Replication

- Relatively new, because it is hard
- Why is it hard ?
  - Data consistency
- 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 (example: RAID1 has 2 disks → we can do 2 IOs in parallel)

// Replication: History

- Before MySQL 5.6 and MariaDB 10.0, replication is single-threaded (Tungsten had support for parallel replication earlier)
- MySQL 5.6 has support for *schema* based parallel replication
- MariaDB 10.0 has support for *domain id* based parallel replication and also has support for *group commit* based parallel replication
- MariaDB 10.1 adds support for *optimistic* parallel replication
- MySQL 5.7 adds support for *logical clock* parallel replication
  - In early version, the logical clock is group commit based
  - In current version, the logical clock is *interval* based

// 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
- **Deployment**:
  - On the master: nothing to do (except having multiple independent schemas)
  - On the slave: “`SET GLOBAL slave_parallel_workers > 0;`”
- MySQL 5.7 has the same feature:
  - (default for slave-parallel-type = DATABASE)
- MySQL 5.8 defaults might be different:
  - 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 one they appear in the master binary logs

- 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 position are committed
  - This “all committed before” position is called a checkpoint
  - Some transactions might be committed after the SQL thread position
  - But some transactions might still be executing (or queued for execution) → gaps
- Impacts on crash recovery (because gaps):
  - Not as simple as resetting the IO thread position at the SQL thread position
- Impacts on GTIDs:
  - Temporary holes in `@@global.gtid_executed` (because of gaps)
- And more…
  - Skipping transactions, backups, heartbeat, …

// Replication: MySQL 5.6”’

- Removing gaps in transaction execution:
  - “`STOP SLAVE; START SLAVE UNTIL SQL_AFTER_MTS_GAPS;`”
- MySQL 5.6 is not parallel replication crash safe without GTIDs (yet…):
  - http://jfg-mysql.blogspot.co.uk/2016/01/replication-crash-safety-with-mts.html
- For skipping transactions, 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)

// Replication: MySQL 5.6”’ ’

- What is a MTS checkpoint ?
  - After making sure gaps are filled,
    a checkpointing advances the position reported by “`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 transactions small (avoid LOAD DATA INFILE and others…)

// 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;`" (*SPT*)
- Also available in MariaDB 10.1
- Do not mess-up advertising the write domain !
  - MySQL 5.6 will protect you from multi-schema transactions, MariaDB cannot
- Also out-of-order commit of transactions on slaves:
  - There will be gaps, those gaps are managed by MariaDB GTIDs,
  - Impact on binary logs content, SHOW SLAVE STATUS, skipping transactions, backups, heartbeat, …

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

- Difference with MySQL 5.6:
  - "`SHOW SLAVE STATUS`" is the position of the latest committed transactions, there might be gaps before…
  - If the SQL thread stops (or is stopped), its position will "rewind" to a "safe" position
    https://mariadb.atlassian.net/browse/MDEV-6589
    https://mariadb.atlassian.net/browse/MDEV-9138
- Removing gaps:
  - `STOP SLAVE; SET GLOBAL slave_parallel_threads = 0; START SLAVE;`
  - Not re-downloading relay logs, 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
- Dispatching algorithm, its impact, and tuning parameters:
  - Long transactions, big transactions, … we will come back to that after in-order

// Replication: MariaDB 10.0 (in-order)

- **Concept**:
  transactions committing together on the master
  can be executed in parallel on slaves

- **Implementation**:
  - Build on top of the binary log *Group Commit* optimisation:
    the master tags transactions in the binary logs with their Commit ID (*cid*)

- **Deployment**:
  - Needs a MariaDB 10.0 master
  - On slaves: "`SET GLOBAL slave_parallel_thread > 0;`"

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

- Binlog example:

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

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

- Good or bad grouping of transactions on the master:
  - When `sync_binlog = 1`, instead of syncing the binlog after each transaction, MariaDB buffers transactions and sync the binlogs a single time for a group
  - Setting `sync_binlog = 0` or `> 1` might lead to bad grouping
  - When not enough parallelism, or sync very fast, grouping might 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
  - The 1st divided by the 2nd gives the group size
- Grouping optimisation (slowing down the master to speed-up slaves):
  - `BINLOG_COMMIT_WAIT_USEC` (*BCWU*):
    timeout, in microseconds, for waiting more transactions to join the group
  - `BINLOG_COMMIT_WAIT_COUNT` (*BCWC*):
    number of transactions when waiting stops

// 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
```

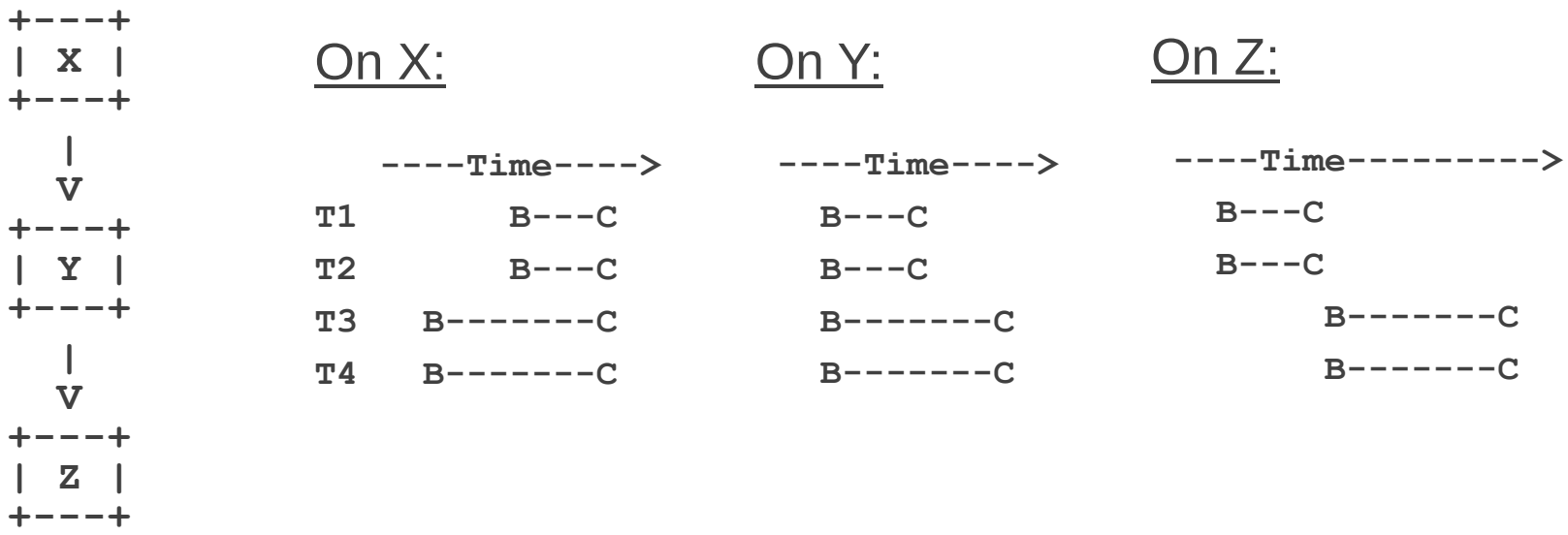
- On the slaves:

```
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 !

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

- Replicating through intermediate master losses grouping
- Four transactions on X, Y and Z:



- To get maximum replication speed on 2nd level slaves, replace intermediate masters by Binlog Servers

Booking.com

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

More graphs and details at:
http://blog.booking.com/better_parallel_replication_for_mysql.html

Booking.com

// Replication: MariaDB 10.0

- Work dispatching algorithm to threads:
  - Work queue, per thread, which contains 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 work pauses (big transactions block scheduling)
  - Once a thread is scheduled work in a domain, it is only queued work for this domain
  - ➢ If all threads are scheduled work, a new domain will starve until a thread has processed all its queue

- Solutions: tuning parameters:
  - slave-parallel-max-queued (default 128KB): buffer, per thread, to queue transactions
  - slave_domain_parallel_threads (default 0): max number of threads a domain can use

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

// 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
T2:         B----C
```

// Replication: Slave Group Commit'

- MariaDB 10.0 implements Slave Group Commit when the master is running MariaDB 10.0, `SPT > 0`, `BCWC > 0` and `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

- Except for DDL where short-circuit is not implemented

// Replication: MariaDB 10.1 (in-order)

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

// Replication: MariaDB 10.1 (in-order)'

- With MariaDB 10.0 (and with MariaDB 10.1 in conservative mode), running in parallel, transactions that group committed on the master, and committing them in-order, can lead to deadlocks
- On the master, T1 and T2 commit together:

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

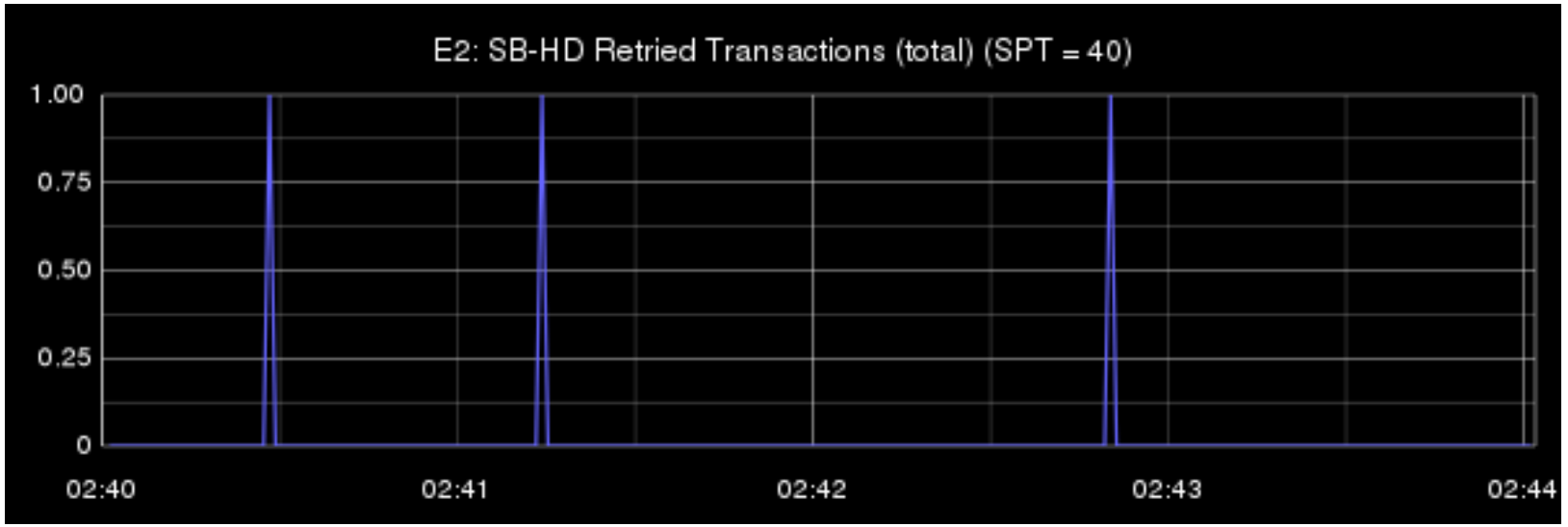
- On the slaves, T2 blocks T1 (because index update, …), but T1 must commit before T2 → deadlock !

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

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

// Replication: MariaDB 10.1 (in-order)”

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



- Retried transactions happen 3 times in 4 minutes → not often at all

Booking.com

// 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
- **Implementation**:
  - Natural evolution from MariaDB 10.0
- **Deployment**:
  - Needs a MariaDB 10.1 master
  - `SET GLOBAL slave_parallel_thread > 0;`
  - `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 run optimistically:
  - DDL blocks the parallel replication pipeline
  - Same for other non-transactional operations

// Replication: MySQL 5.7

- MySQL 5.7 has different slave parallel type:
  - `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 documentation) (the logical clock is implemented using intervals)
- 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

// Replication: MySQL 5.7’

- Binlog example:

```
#160121 15:45:51 ... last_committed=0  sequence_number=9
#160121 15:45:51 ... last_committed=0  sequence_number=10
#160121 15:45:51 ... last_committed=10 sequence_number=11
#160121 15:45:51 ... last_committed=10 sequence_number=12
...
#160121 15:45:51 ... last_committed=10 sequence_number=19
#160121 15:45:51 ... last_committed=10 sequence_number=20
#160121 15:45:52 ... last_committed=20 sequence_number=21
#160121 15:45:52 ... last_committed=20 sequence_number=22
```

// Replication: MySQL 5.7”

- 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.co.uk/2016/01/replication-crash-safety-with-mts.html
  - And also everything else: binary logs content, SHOW SLAVE STATUS, skipping transactions, backups, …

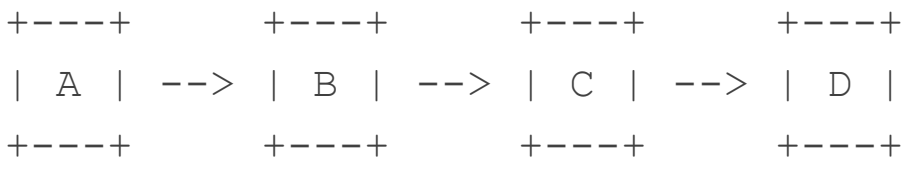
- Using `slave_preserve_commit_order = 1` does what you expect
  - This configuration does not generate gap
  - But it needs log-slave-updates, there is a feature request to remove this limitation:
    https://bugs.mysql.com/bug.php?id=75396
  - And it is still not replication crash safe (surprising because no gap):
    https://bugs.mysql.com/bug.php?id=80103

// Replication: results from B.com

- MariaDB 10.0 tests:
  - On four environments, from MySQL 5.6 masters, thanks to Slave Group Commit
- MariaDB 10.1 tests: conservative vs aggressive
  - Four same environments
- MySQL 5.6 real deployment
- MariaDB 10.0 real deployment
- No results from MySQL 5.7:
  - I guess we can expect similar results to MariaDB 10.0

// 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 a group committing master generates bigger groups than slave group commit, 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 g-commit

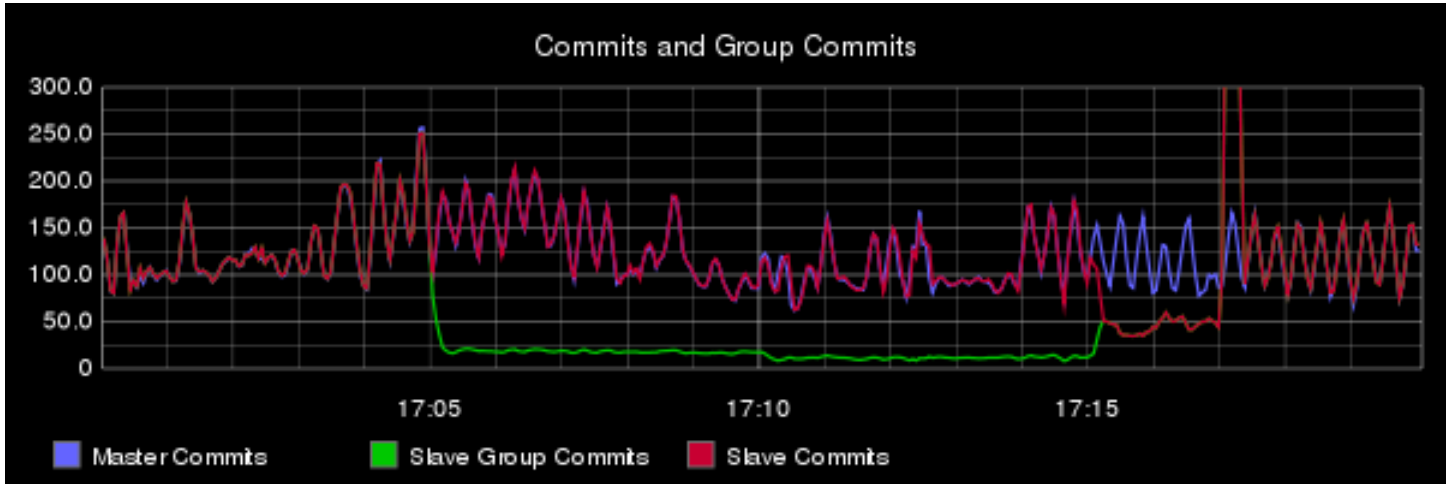
- Upside of group commit: when a sync is expensive on a slave, syncing could become the bottleneck of replication
  - In this case, doing less sync is a win
  - Slave group commit allows that with "`sync_binlog =1`" and "`trx_commit = 1`"

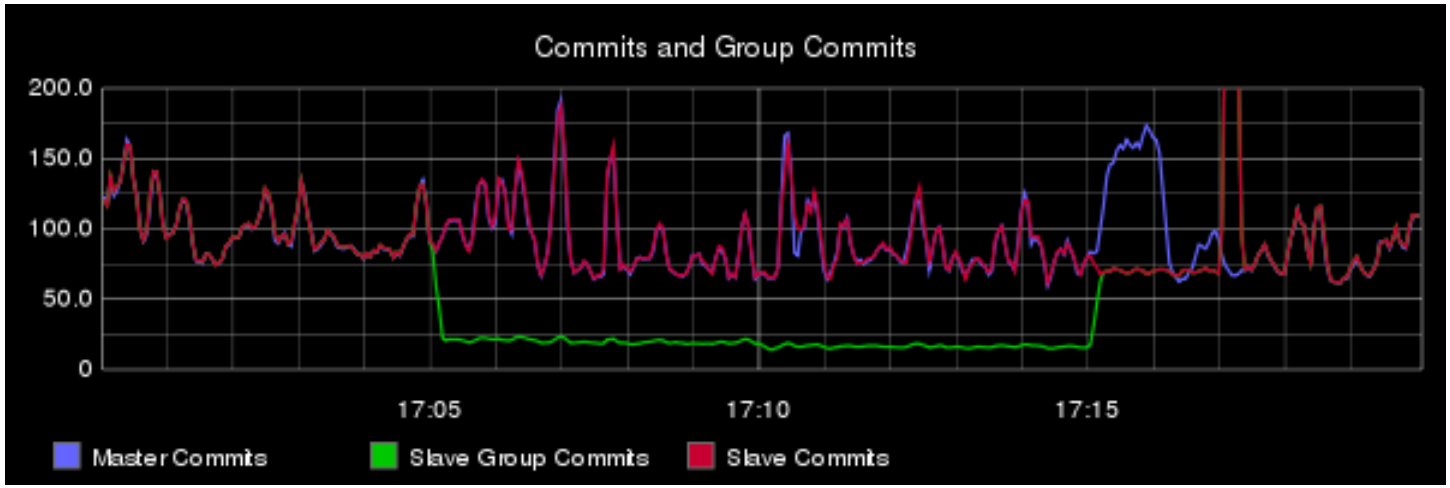
- With B and C having RAID controller write cache (syncs are very fast):
  1. Run B and C without slave group commit (before 17:05)
  2. At 17:05, enable slave group commit on C
  3. At 17:10, disable the write cache on C → syncs become expensive
  4. At 17:15: disable slave group commit on C
- During the test, we monitor commits and group commits

More details at:

http://blog.booking.com/evaluating_mysql_parallel_replication_2-slave_group_commit.html

// Replication: MariaDB 10.0 g-commit'





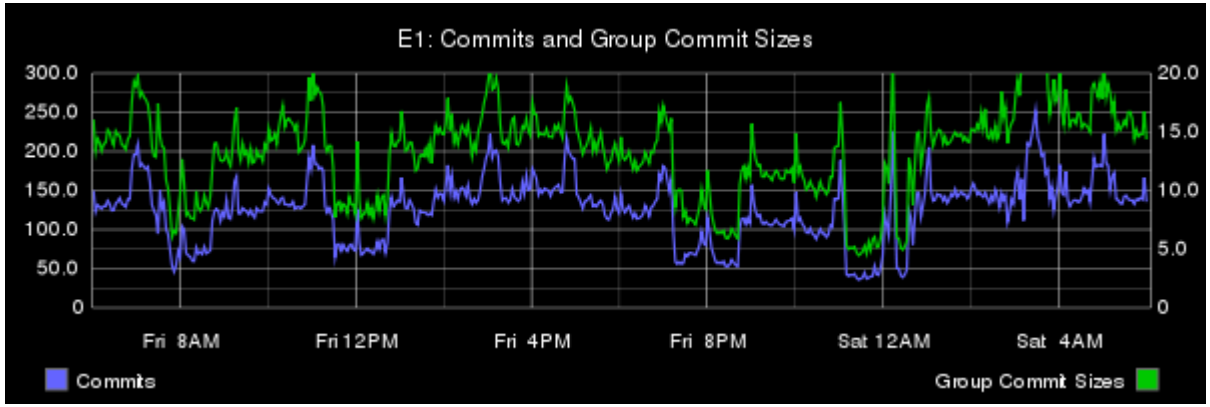
Booking.com

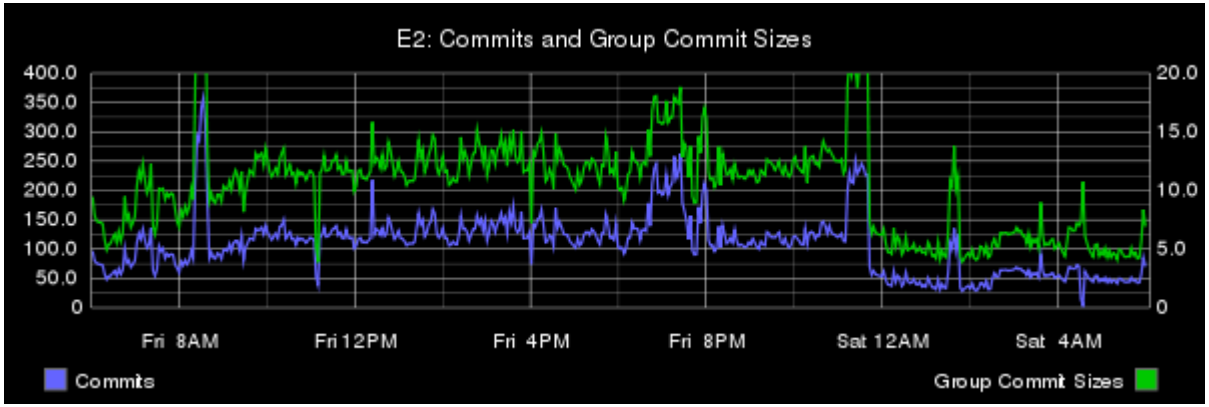
// Replication: MariaDB 10.0 g-commit”

Booking.com

// Replication: MariaDB 10.0 p-tests

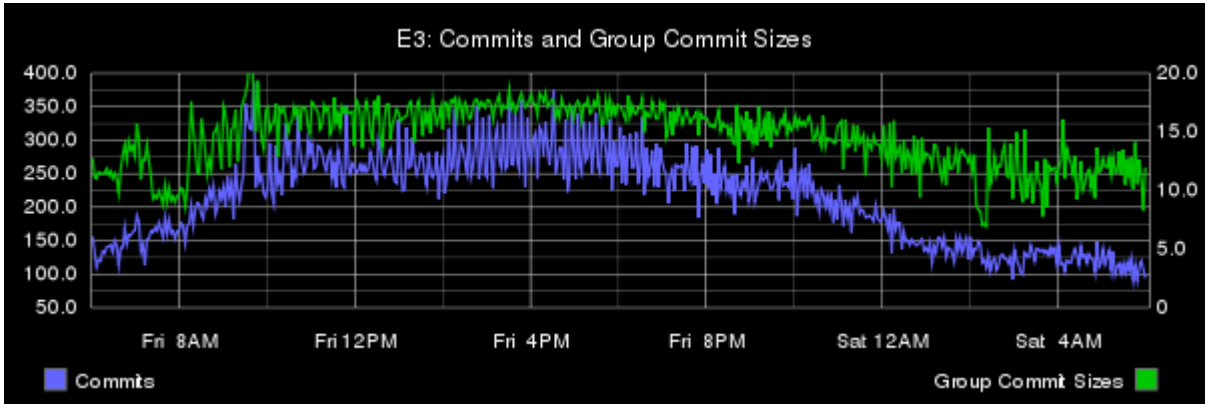
- Parallel replication with MariaDB 10.0 (or with 10.1 conservative):
  - Catching up 24 hours of replication delay with 0, 5, 10, 20 and 40 threads

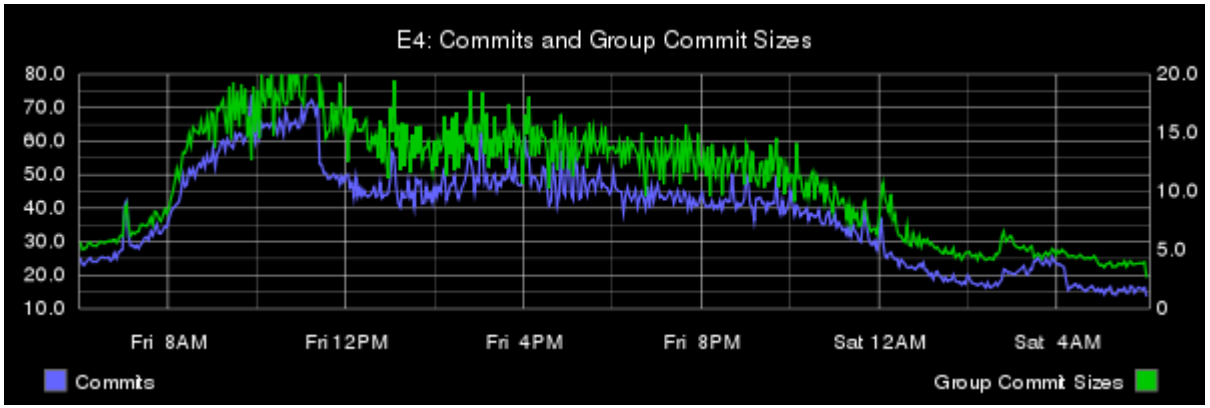




Booking.com

// Replication: MariaDB 10.0 p-tests’





More details at:

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

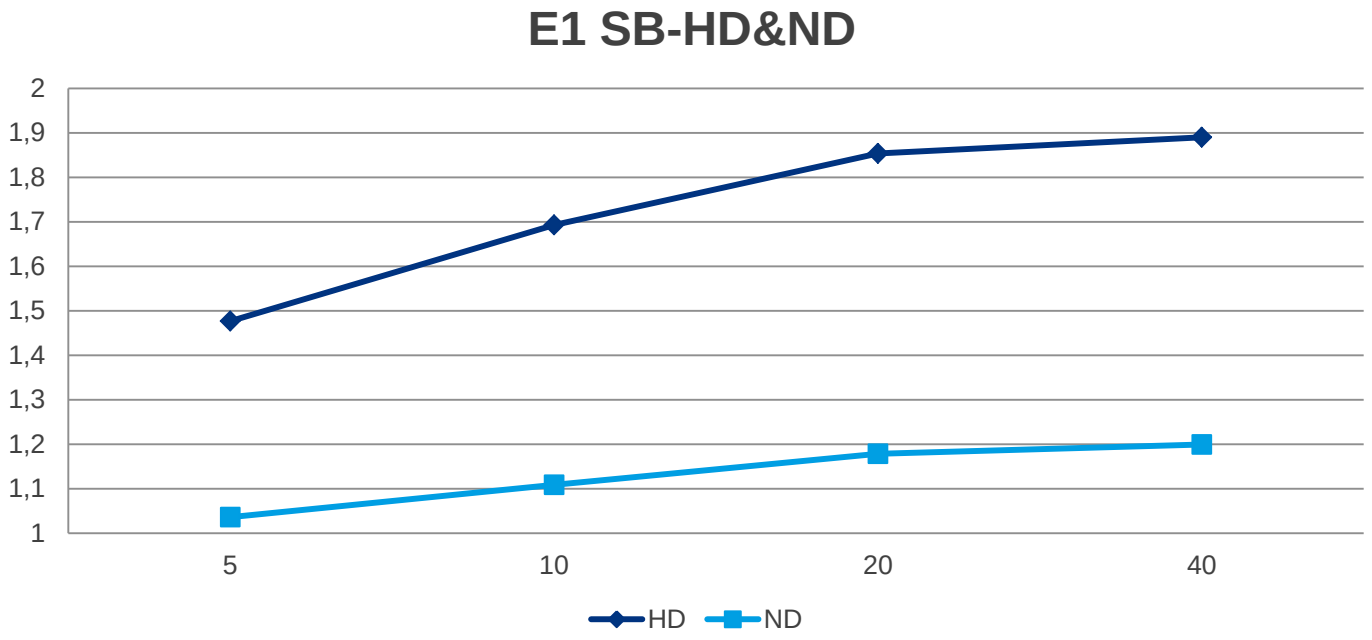
Booking.com

// Replication: MariaDB 10.0 p-tests”

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

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

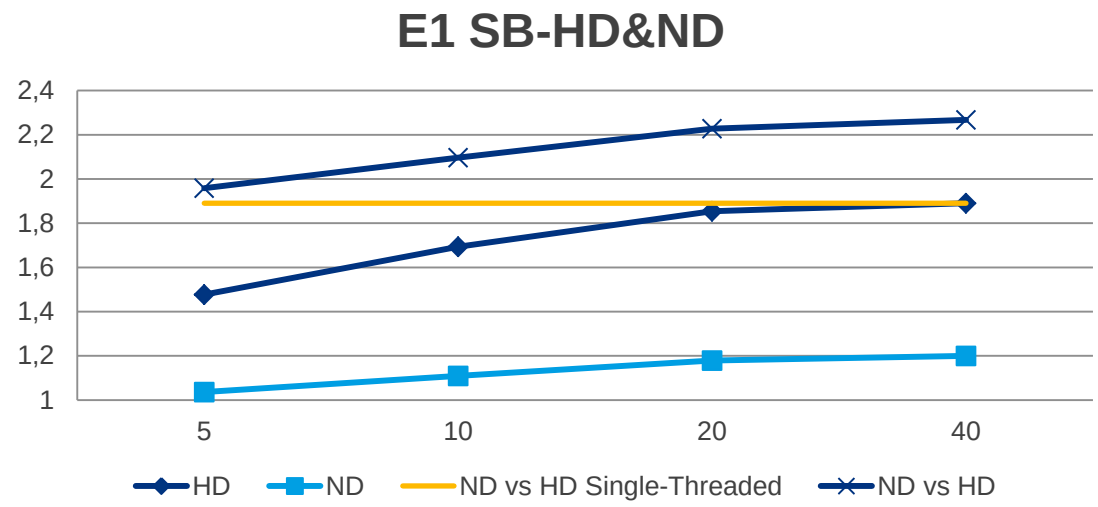
No Durability (*ND*): “`sync_binlog = 0`” + “`trx_commit = 2`”

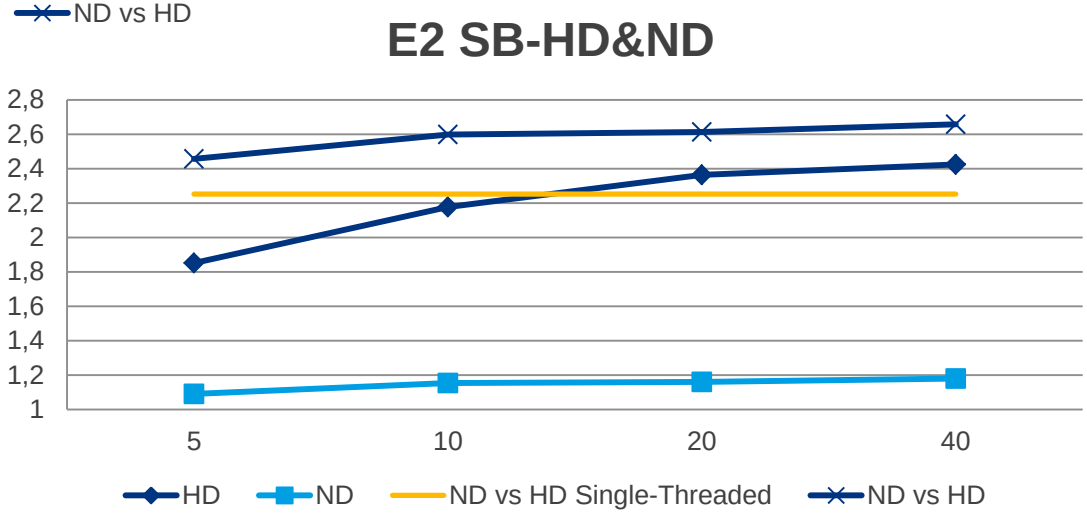


HD Single-Threaded: 3h09.34

ND Single-Threaded: 1h24.09

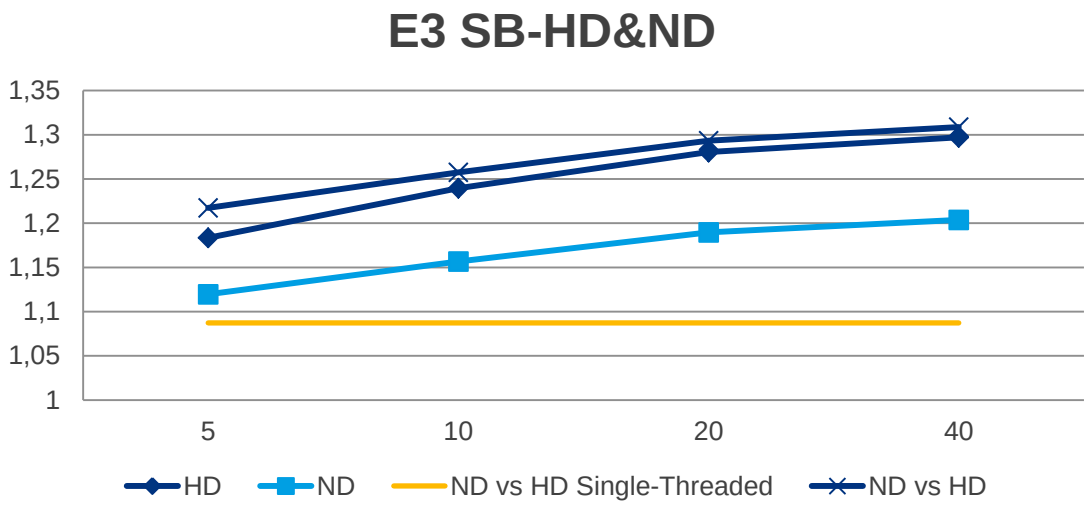
// Replication: MariaDB 10.0 p-tests”’

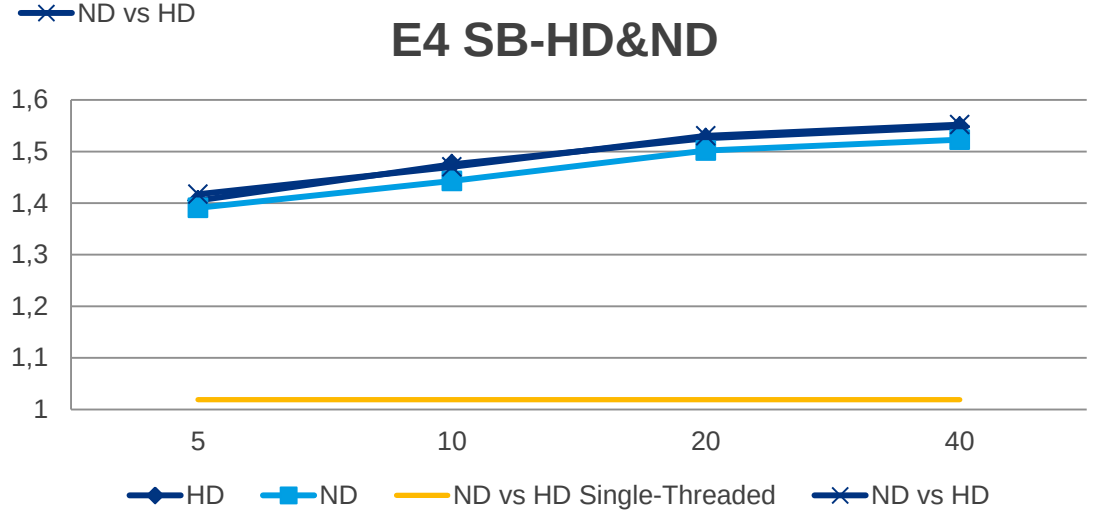




Booking.com

// Replication: MariaDB 10.0 p-tests’’’ ’

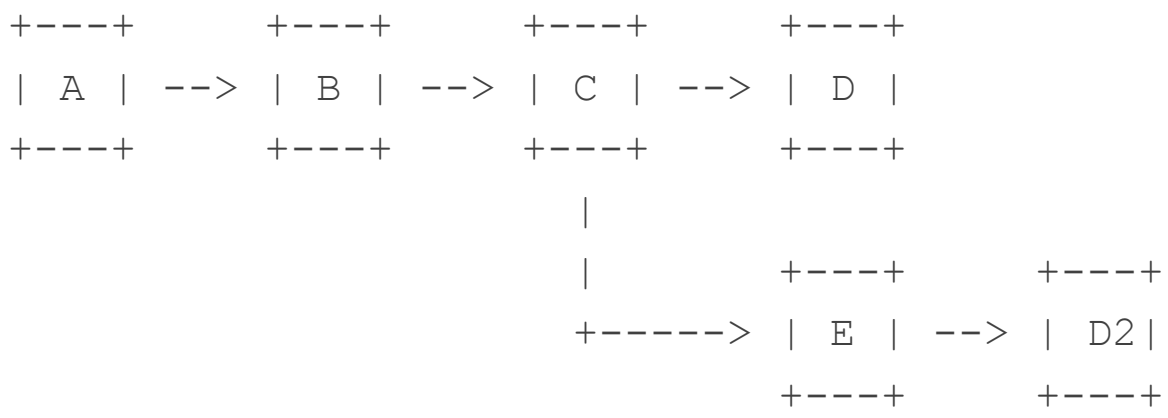




Booking.com

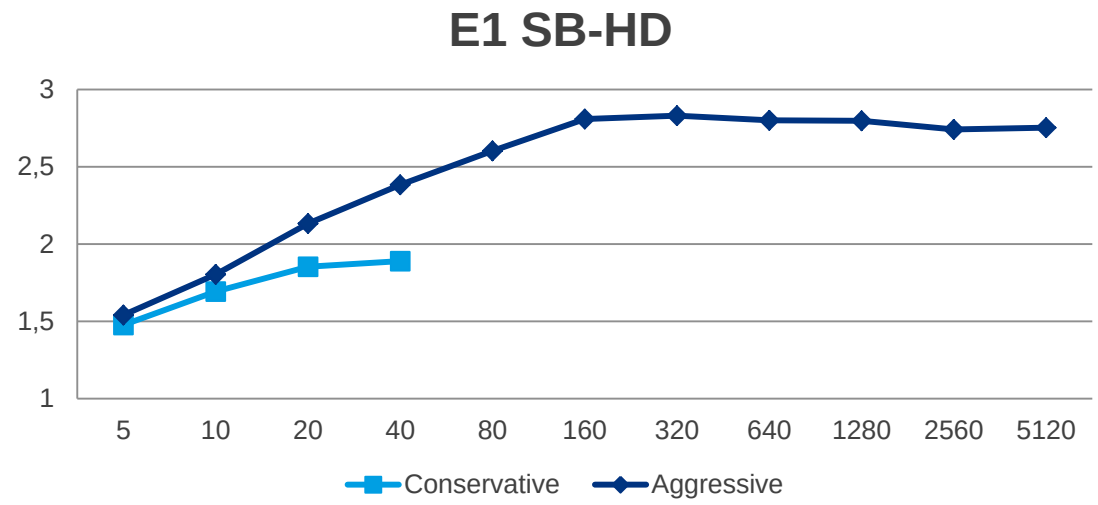
// Replication: MariaDB 10.1 tests

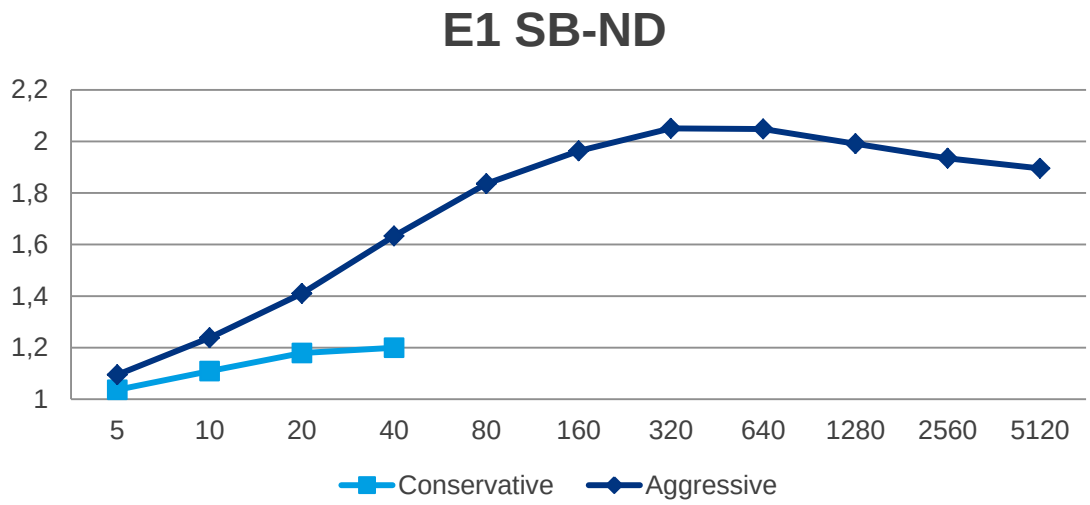
- Four same environments, D now runs MariaDB 10.1, and to take advantage of optimistic parallel replication, we need a 10.1 master → add E



- D and D2 are the same hardware
- D runs with SPT = conservative
- D2 runs with SPT = aggressive

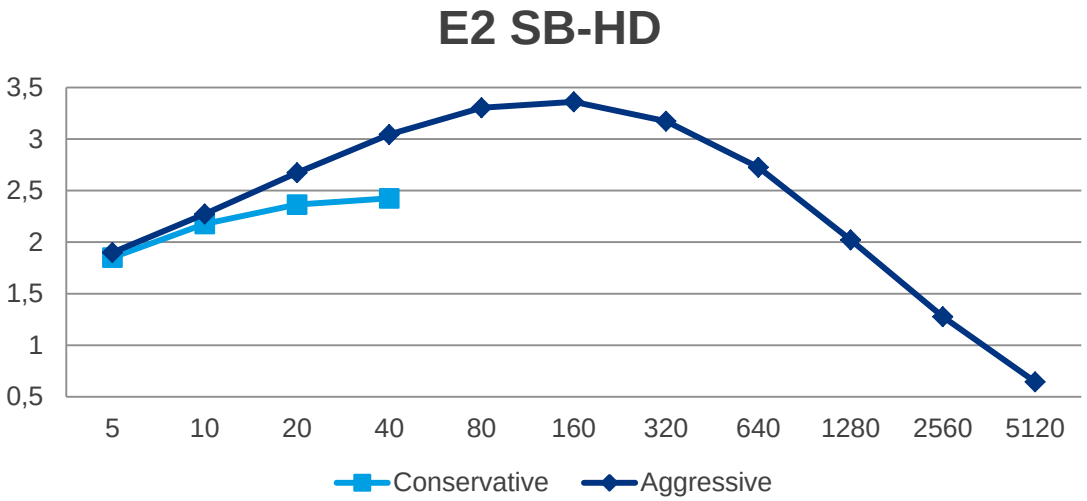
// Replication: MariaDB 10.1 tests'

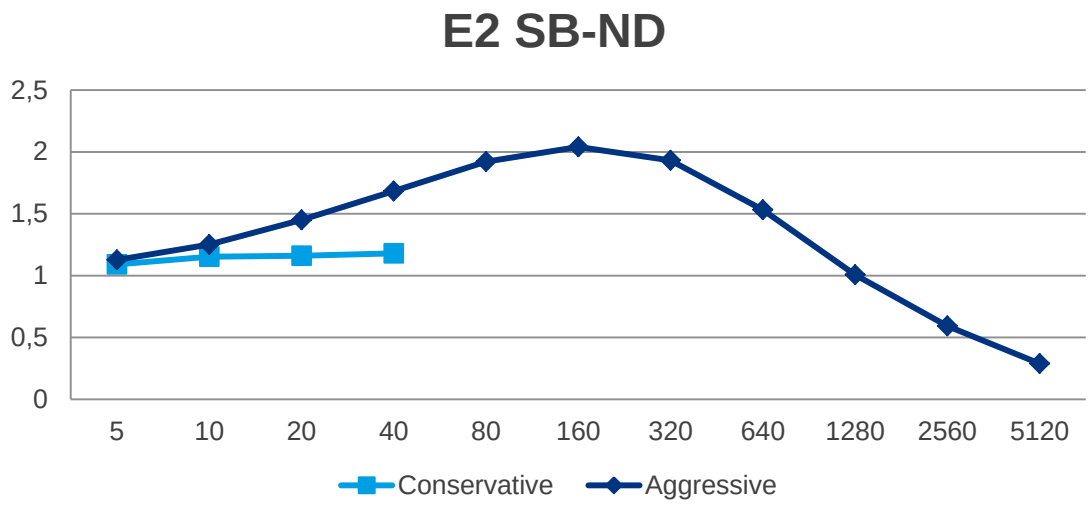




Booking.com

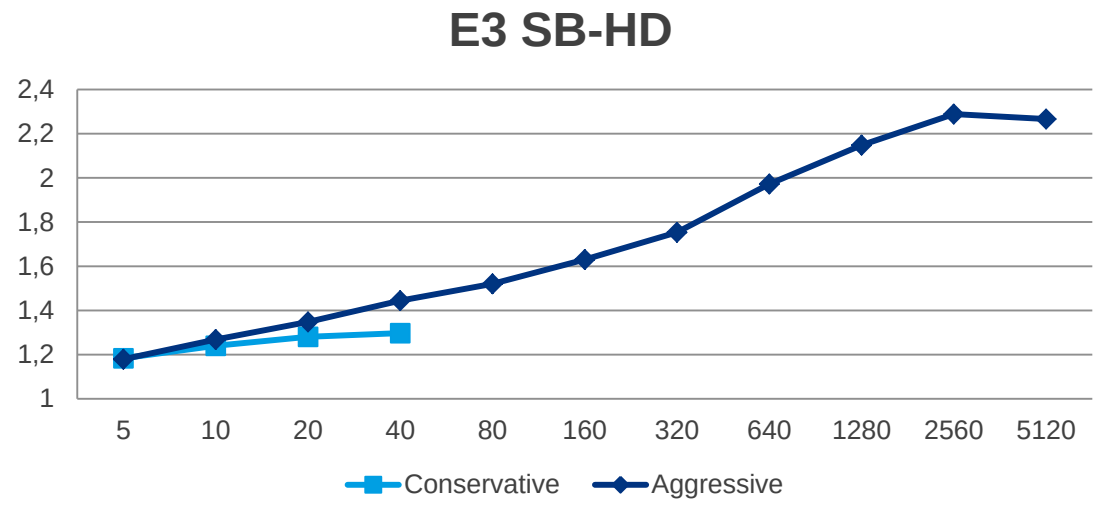
// Replication: MariaDB 10.1 tests”

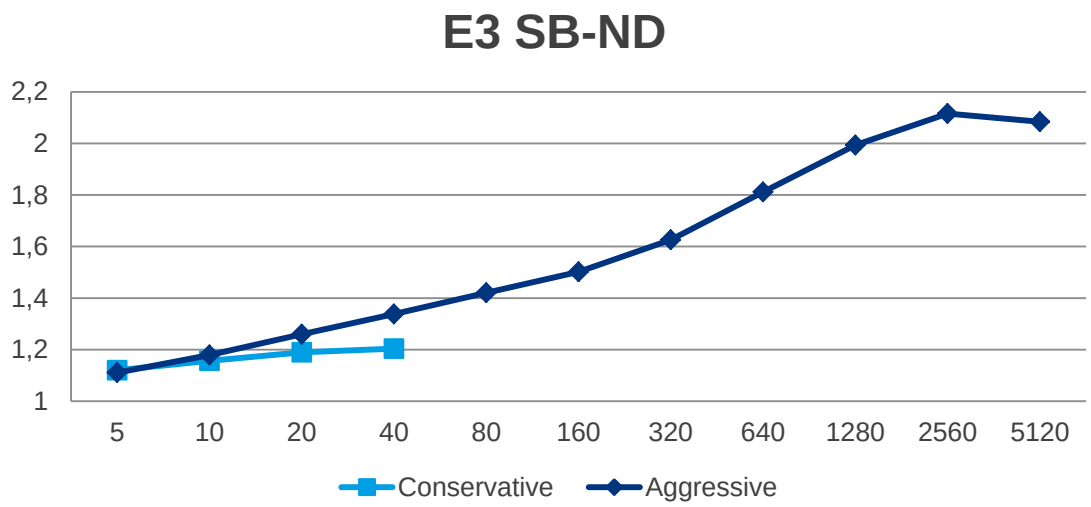




Booking.com

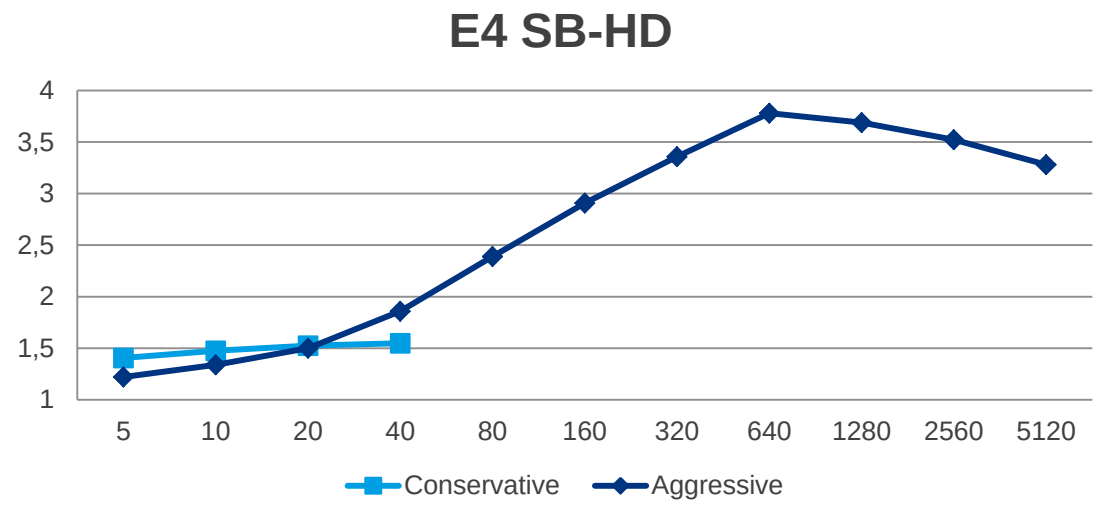
// Replication: MariaDB 10.1 tests”’

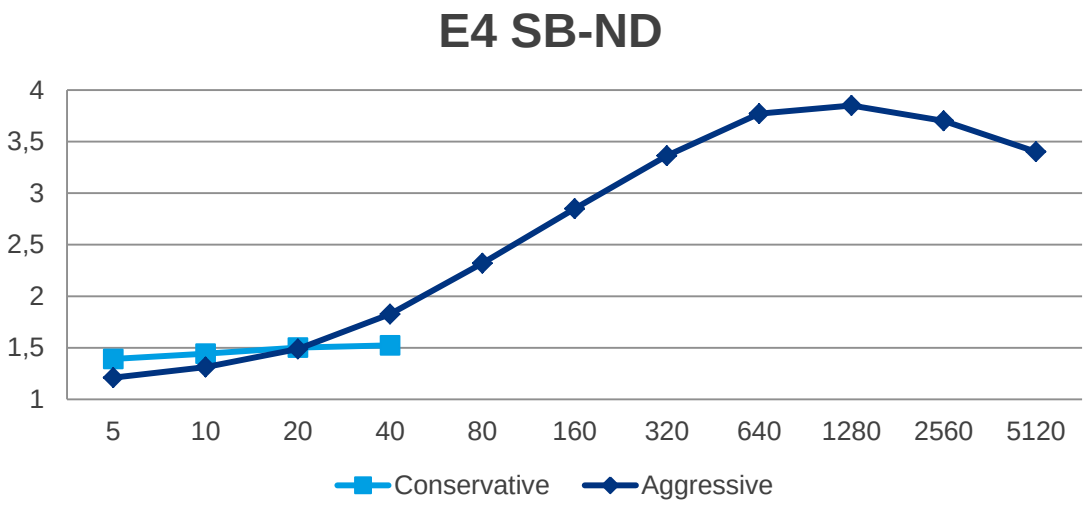




Booking.com

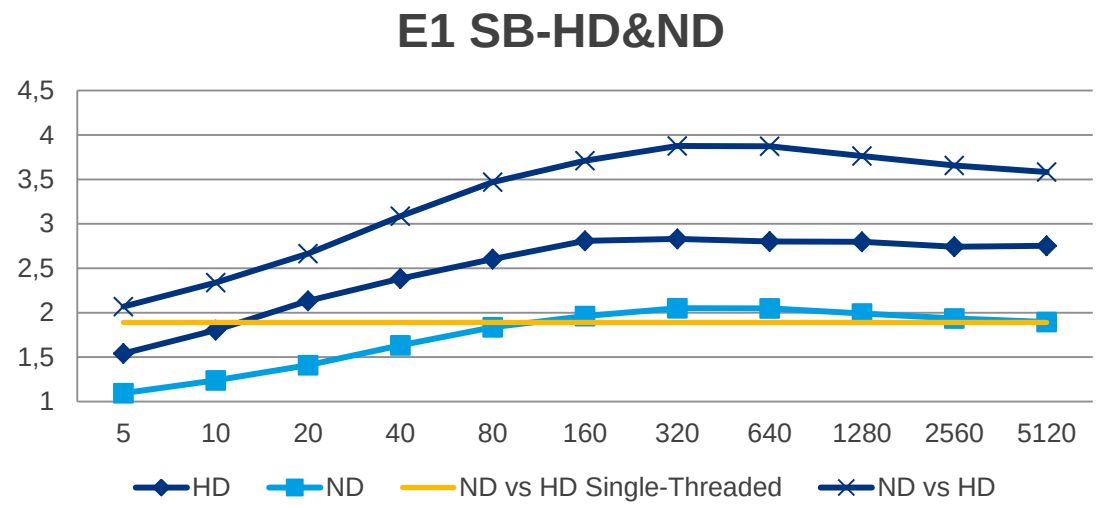
// Replication: MariaDB 10.1 tests”’ ’

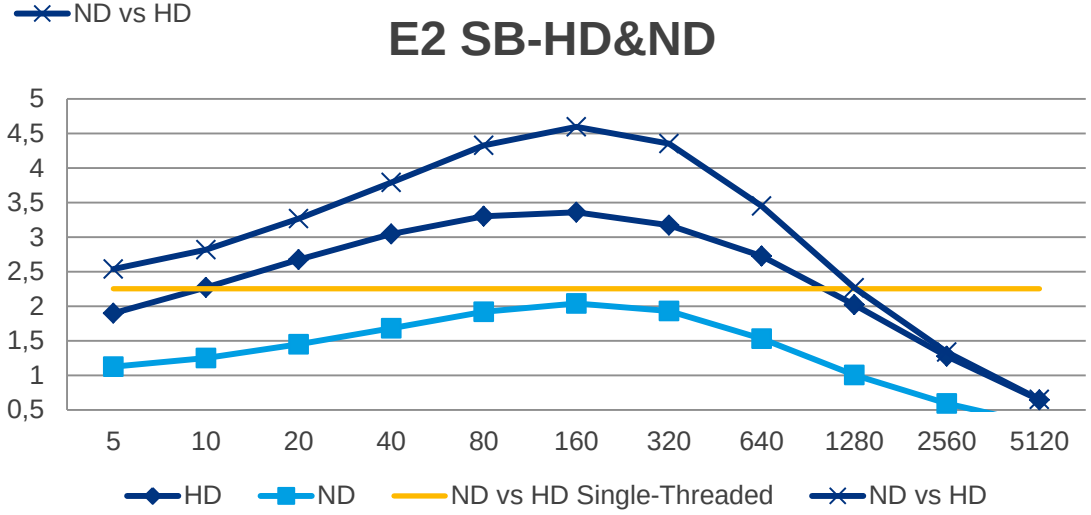




Booking.com

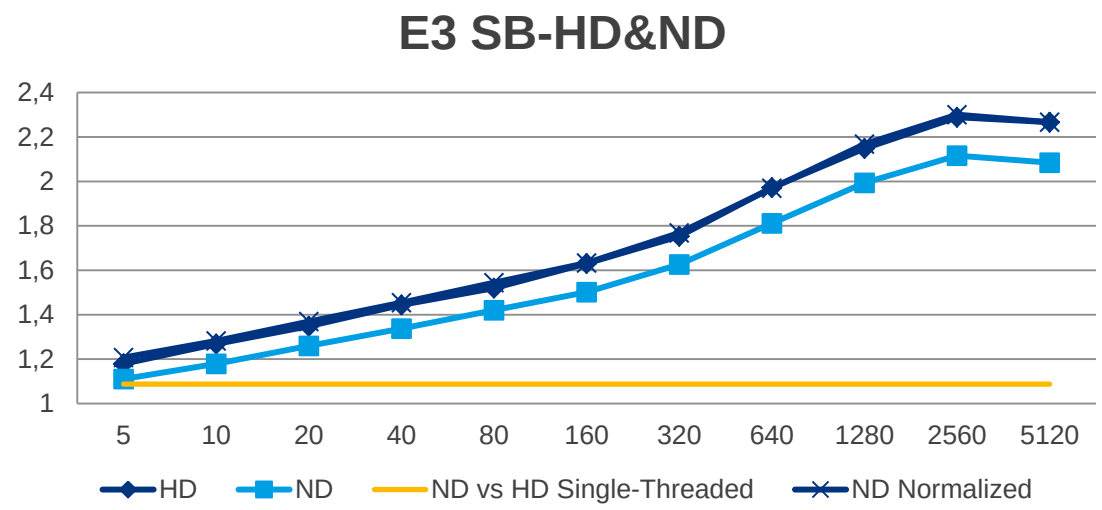
// Replication: MariaDB 10.1 tests”’ ”

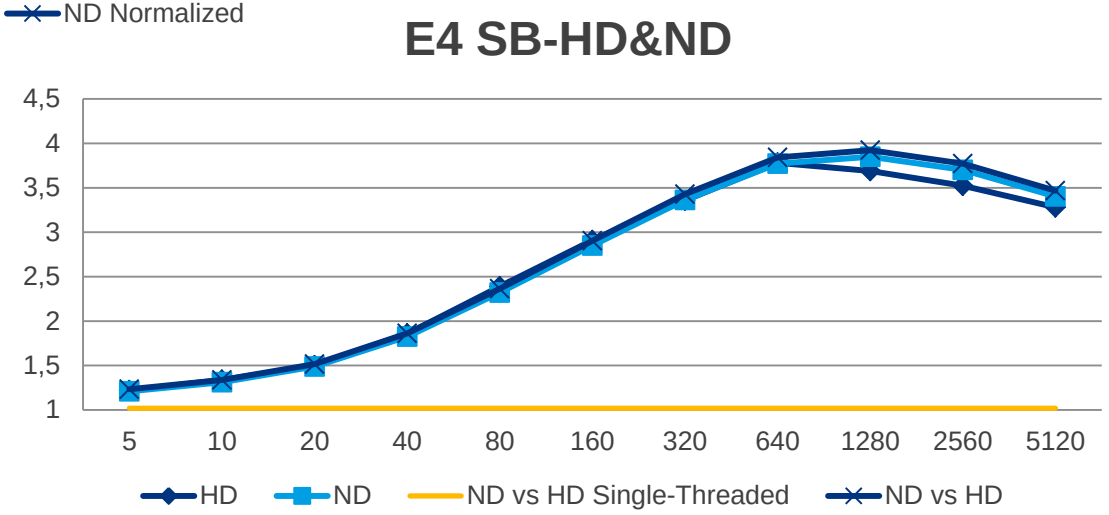




Booking.com

// Replication: MariaDB 10.1 tests”’ ”’

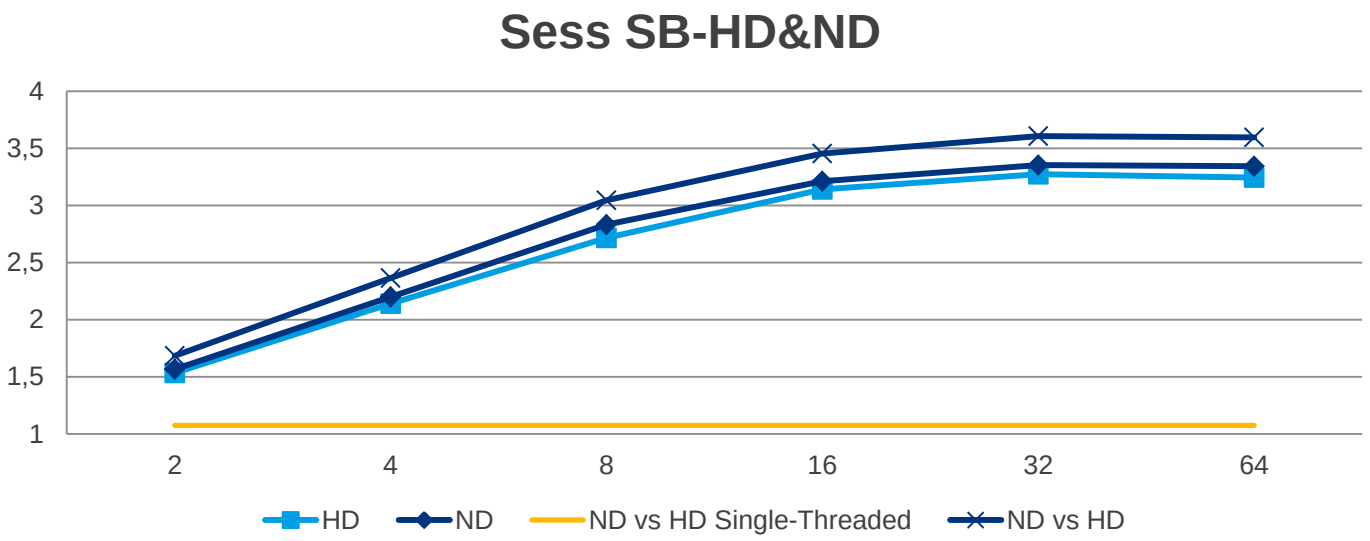




Booking.com

// Replication: MySQL 5.6 real

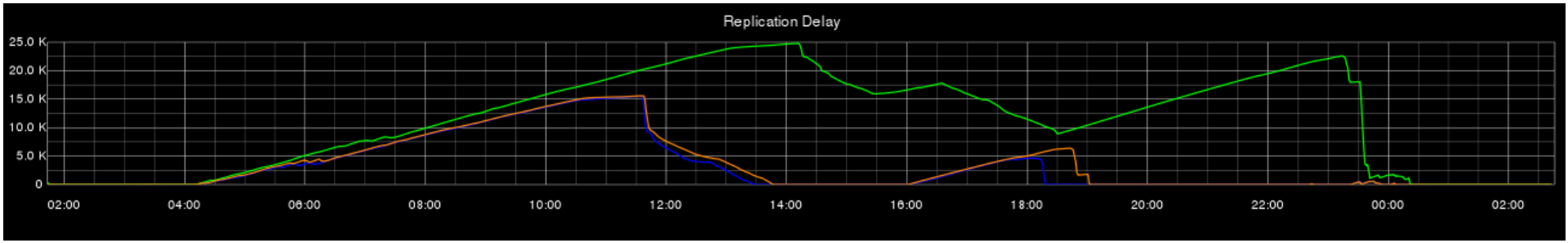
- Booking.com session store is sharded with many schema per database:
  - Running MySQL 5.6, >1 TB per node, 20 schema per node, magnetic disks
  - PLAMS 2015: Combining Redis and MySQL to store HTTP cookie data https://www.percona.com/live/europe-amsterdam-2015/sessions/combining-redis-and-mysql-store-http-cookie-data



Booking.com

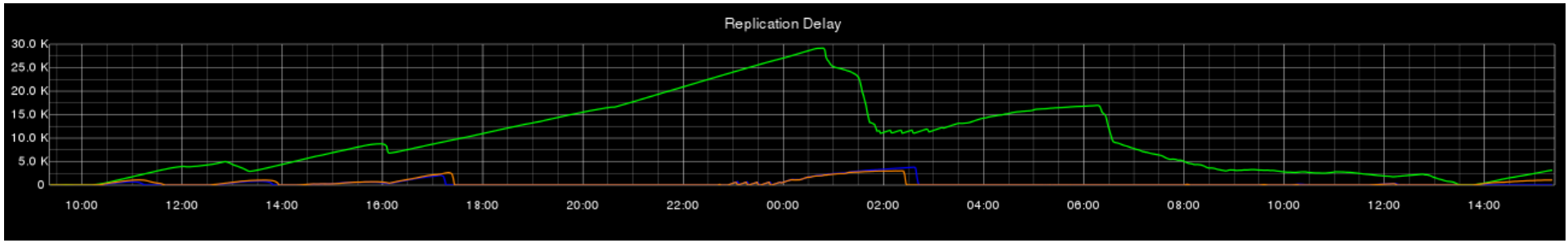
// Replication: MariaDB 10.0 real

- Booking.com is also running MariaDB 10.0.
- One of those system has very high replication load:
  - we optimised group commit on the master and enabled parallel replication on some slaves (blue and orange line below with 30 threads, and yellow with 0)
  - The Y axis is the replication delay (in seconds)

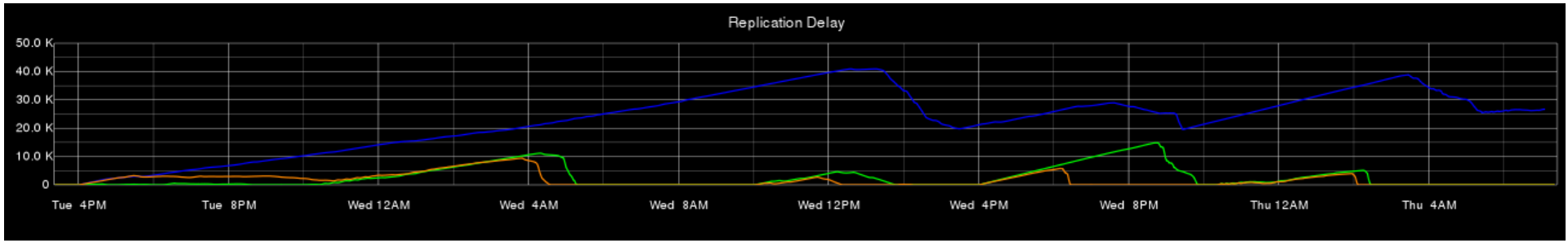


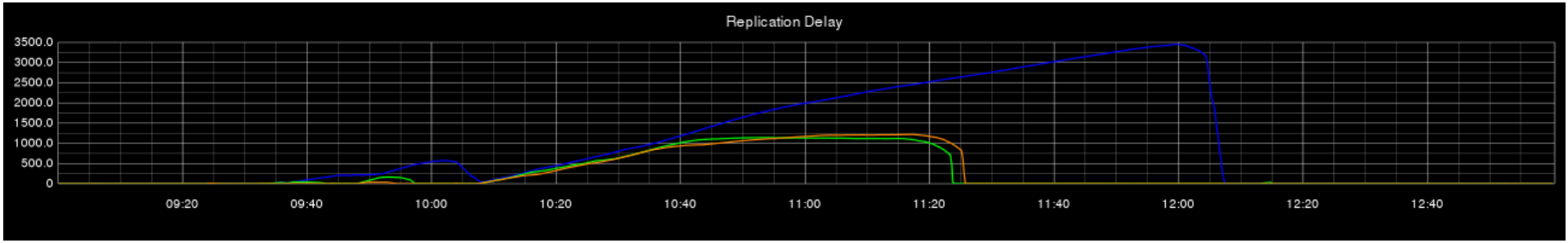
Booking.com

// Replication: MariaDB 10.0 real’



- Swapping parallel replication settings on blue and yellow:





Booking.com

// 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 looses 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 solution needs careful 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

- In all cases, avoid big transactions in the binary logs

// Replication: Links

- Better Parallel Replication for MySQL: http://blog.booking.com/better_parallel_replication_for_mysql.html
- Evaluating MySQL Parallel Replication Part 2: Slave Group Commit: http://blog.booking.com/evaluating_mysql_parallel_replication_2-slave_group_commit.html
- Evaluating MySQL Parallel Replication Part 2: Slave Group Commit: http://blog.booking.com/evaluating_mysql_parallel_replication_2-slave_group_commit.html
- Evaluating MySQL Parallel Replication Part 3: Benchmarks in Production: http://blog.booking.com/evaluating_mysql_parallel_replication_3-benchmarks_in_production.html
- Evaluating MySQL Parallel Replication Part 4: More Benchmarks in Production: about optimistic // replication, to be published eventually on http://blog.booking.com
- Evaluating MySQL Parallel Replication Part 5: Event more Benchmarks in Production: about MySQL 5.6 and maybe more, to be published on http://blog.booking.com
- Replication crash safety with MTS in MySQL 5.6 and 5.7: reality or illusion? http://jfg-mysql.blogspot.co.uk/2016/01/replication-crash-safety-with-mts.html
- Do not run those commands with MariaDB GTIDs http://jfg-mysql.blogspot.fr/2015/10/bad-commands-with-mariadb-gtids-2.html

// Replication: Links’

- 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
- Bugs/feature requests:
  - Message after MTS crash misleading: https://bugs.mysql.com/bug.php?id=80102 (and https://bugs.mysql.com/bug.php?id=77496)
  - MTS LOGICAL_CLOCK with slave_preserve_commit_order=1 not repl. crash safe: http://bugs.mysql.com/bug.php?id=80103
  - Relay log pos. corrupted with p-replication after interrupt. LOAD DATA on master https://mariadb.atlassian.net/browse/MDEV-9138 (related to https://mariadb.atlassian.net/browse/MDEV-6589)
- 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/

Thanks

Jean-François Gagné
jeanfrancois DOT gagne AT booking.com