The consequences of `sync_binlog != 1`

by Jean-François Gagné
Senior Infrastructure Engineer / System and MySQL Expert
jeanfrancois AT messagebird DOT com / @jfg956 #FOSDEM #MySQLDevRoom

The full title of the talk should be

The consequences of `sync_binlog != 1` and of `innodb_flush_log_at_trx_commit = 2`

as one without the other does not make much sense

(I will use `trx_commit` for short in this talk)

Summary

(The consequences of `sync_binlog != 1` – MySQL.FOSDEM.2020)

- Faster but by how much ?
- Overview of replication and zoom in `sync_binlog` & `trx_commit`
- Avoiding `sync_binlog != 1`
- The consequences of `sync_binlog != 1`
- Mitigating `sync_binlog != 1`
- Closing, links and questions

This talk applies mostly to MySQL 5.6 and 5.7 (unless explicitly mentioned)
some content will apply to MariaDB 10.0+ (and will be explicitly mentioned)

Faster but by how much ?

(The consequences of `sync_binlog != 1` – MySQL.FOSDEM.2020)

Sysbench, single-thread insert benchmark, without secondary index,
vm on Google Cloud Platform (*GCP),* SSD persistent-disks, MySQL 5.7.26

With `sync_binlog = 1` & `trx_commit = 1`:

- ~200 Transactions Per Second (*TPS*) on the master
- ~230 TPS on a slave

With `sync_binlog = 0` & `trx_commit = 2`:

- ~3770 TPS on the master (~18x faster)
- ~7050 TPS on a slave (~30x faster)

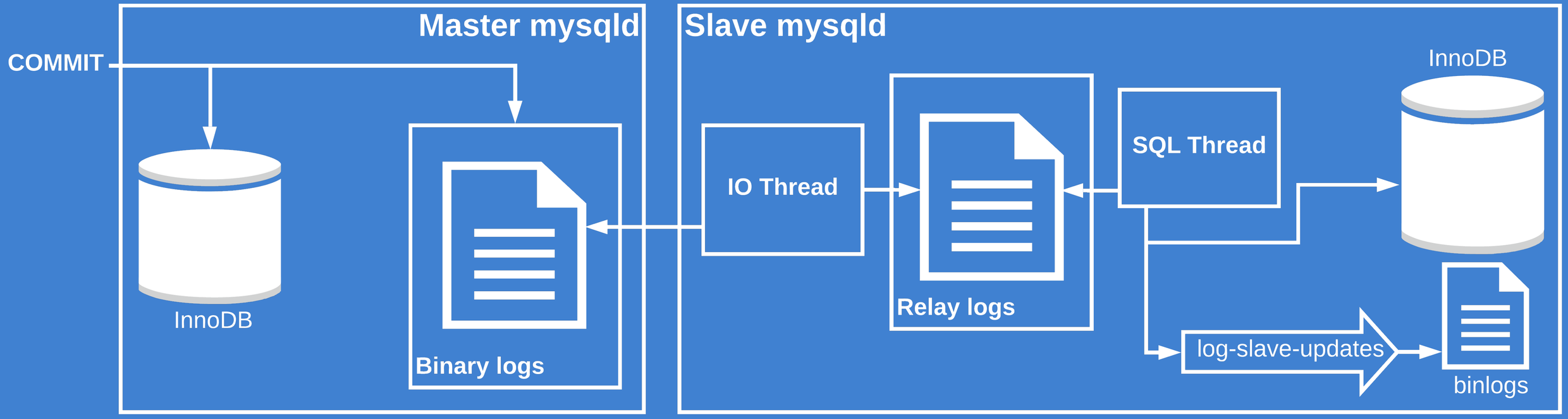
https://jfg-mysql.blogspot.com/2019/07/master-replication-crash-safety-part-4-benchmarks-of-high-n-low-durability.html

Overview of MySQL Replication

(The consequences of `sync_binlog != 1` – MySQL.FOSDEM.2020)

One master node with one or more slave nodes:

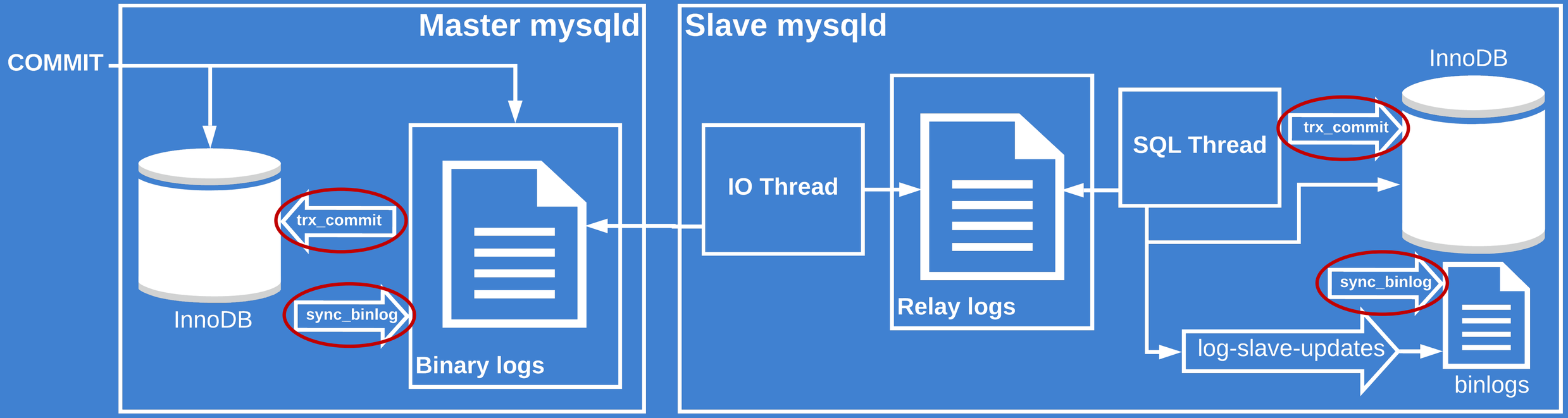
- The master records transactions in a journal (binary logs); each slave:
  - Downloads the journal and saves it locally in the relay logs (IO thread)
  - Executes the relay logs on its local database (SQL thread)
  - Could also produce binary logs to be a master (**`log-slave-updates`**)



Zoom in `sync_binlog` & `trx_commit` [1 of 6]

(The consequences of `sync_binlog != 1` – MySQL.FOSDEM.2020)

- `sync_binlog = N`: the binlogs are flushed to disk every N trx group
- `trx_commit = {1,2,0}`
  - 1: the InnoDB Redo Log (*IRL*) is flushed to disk after each transaction
  - 2: the IRL is written to after each trx (OS RAM buffer) but flushed every second
  - 0: the IRL is written to and flushed every second (not covered in this talk)



Zoom in `sync_binlog` & `trx_commit` [2 of 6]

(The consequences of `sync_binlog != 1` – MySQL.FOSDEM.2020)

Flushing to disk is not fast !

- Local disks [1]:
  - Desktop and Enterprise grade magnetic SATA: between 18 and 25 ms
  - Consumer grade SSD SATA and NVMe: between 0.5 ms and 10 ms
  - High-end enterprise-grade NVMe: ~0.15 ms
  - Dell Perc with BBU: ~0.04 ms
- Network disks (FC or iSCSI → network round-trip): between 0.5 ms and 1 ms
- Cloud environments:
  - GCP Persistent Disks (network): ~0.6 ms [2]
  - Amazon Web Services Local SSD: ~0.05 ms [3]
    (careful about local SSDs in the cloud as they are not always persistent between reboots)

[1]: https://www.percona.com/blog/2018/02/08/fsync-performance-storage-devices/
[2]: https://jfg-mysql.blogspot.com/2019/07/master-replication-crash-safety-part-4-benchmarks-under-the-hood.html
[3]: https://jfg-mysql.blogspot.com/2019/07/master-replication-crash-safety-part-5a-faster-wo-reducing-durability-hardware.html

Zoom in `sync_binlog` & `trx_commit` [3 of 6]

(The consequences of `sync_binlog != 1` – MySQL.FOSDEM.2020)

What does this mean ?

- **`sync_binlog = 1` & `trx_commit = 1`**: trx are on disk after COMMIT
  - Everything is fully durable (ACI**D**), nothing lost in the case of an OS crash
- **`= 0` & `= 2`**: trx are in an OS RAM buffer after COMMIT, but not on disk
  - No data lost after a mysqld crash (data from OS RAM buffer is not lost)
  - But things are lost in the case of an OS crash
  - And after an OS crash, InnoDB and the binary logs are probably out-of-sync

If those transactions are run on the master: …, D, **E**, F, G, H, I, J, **K**, L, …

- On an OS crash, binlog could be flushed up to E, and InnoDB up to K
- So after recovery, InnoDB will have data up to K (L and after is lost)
  and the transactions F and after are lost from the binary logs

(Note: the scenario where InnoDB looses less than the binlog is more likely
as the Redo Logs are flushed every second, but the opposite might also happen)

Zoom in `sync_binlog` & `trx_commit` [4 of 6]

(The consequences of `sync_binlog != 1` – MySQL.FOSDEM.2020)

Another thing about `sync_binlog != 1`:

- Binary logs are flushed to disk at each binlog rotation
  and putting up to 1 GB on disk this might take time…
- The **associated stalls** have been described by Vadim Tkachenko in [1]

[1]: https://www.percona.com/blog/2018/05/04/how-binary-logs-and-filesystems-affect-mysql-performance/

And with MySQL 5.6+ and MariaDB 5.3+, the *Binary Log Group Commit* optimization allows to persist many transactions to disk in a single flush (InnoDB already had Group Commit for some time):

- Running trx in parallel on the master increases TPS with `sync_binlog = 1`
- MariaDB 10.0+ *Slave Group Commit* [2] allows the same on slaves
- And so does *Parallel Replication* in MySQL 5.6+ and MariaDB 10.0+

[2]: https://medium.com/booking-com-infrastructure/evaluating-mysql-parallel-replication-part-2-slave-group-commit-459026a141d2

Zoom in `sync_binlog` & `trx_commit` [5 of 6]

(The consequences of `sync_binlog != 1` – MySQL.FOSDEM.2020)

Defaults in different versions:

	MySQL 5.5	5.6	5.7	8.0	MariaDB 10.0	10.1	10.2	10.3	10.4
`sync_binlog`	0 😐[1]	0 😐[1]	1 ☺ 😐[2]	1 ☺ 😐[2]	0 😐[3]	0 😐[3]	0 ☹[3]	0 ☹[3]	0 ☹[3]
`trx_commit`	1	1	1	1	1	1	1	1	1
`binlog_order_commits`	N/A[4]	ON	ON	ON	N/A[4]	N/A[4]	N/A[4]	N/A[4]	N/A[4]
`innodb_support_xa`	ON	ON	ON	N/A[5]	ON	ON	ON	N/A[5]	N/A[5]

(The `binlog_order_commits` and `innodb_support_xa` parameters are also this discussion, but their defaults are decent, so they are only briefly mention here.)

Zoom in `sync_binlog` & `trx_commit` [6 of 6]

(The consequences of `sync_binlog != 1` – MySQL.FOSDEM.2020)

Notes from previous page:

1) MySQL introduced binary log group commit (*BLGC*) in 5.6, so up to, and arguably including 5.6, sync_binlog needed to be to 0 for performance reasons, but this was unsafe.
2) As MySQL had GLGC since 5.6, it was possible to make sync_binlog = 1 the default without penalizing performance on the master (☺), but this change made replication the bottleneck and MySQL does not have slave group commit (*SGC*) nor does it enable parallel replication (*// rpl*) by default (😐). Still good to be safe by default (☺).
3) MariaDB introduced BLGC in 5.3, so arguably, they should have made sync_binlog = 1 the default in 10.0, but as this might have impacted replication performance (// rpl just had been introduced in 10.0 and SGC explicitly only in 10.1) this is only a 😐 up to 10.1. But from 10.2, this is a ☹ as databases should be safe by default (sync_binlog = 1) and fast (// rpl or SGC enabled).
4) The binlog_order_commits was introduced in MySQL 5.6, probably as part of the binary group commit implementation, and it never was introduced in MariaDB.
5) The innodb_support_xa parameter was deprecated in MySQL 5.7 and MariaDB 10.2 and removed in 8.0 and 10.3.

Avoiding `sync_binlog != 1` [1 of 5]

(The consequences of `sync_binlog != 1` – MySQL.FOSDEM.2020)

Ideally, we would always run with `sync_binlog = 1` (and `trx_commit = 1`)

When reaching the transaction throughput limit of `sync_binlog = 1`,
because of disk flush latencies becoming the bottleneck,
and before setting `sync_binlog` to `0`, we can:

1. Get faster disks [1] (reducing the latency of a flush)
2. Run transactions in parallel on the master
   (persisting many trx with a single flush thanks to Binary Log Group Commit)
3. Use parallel replication (hoping to persist many trx with a single flush)
   (including if running MariaDB 10.0+, use Slave Group Commit)

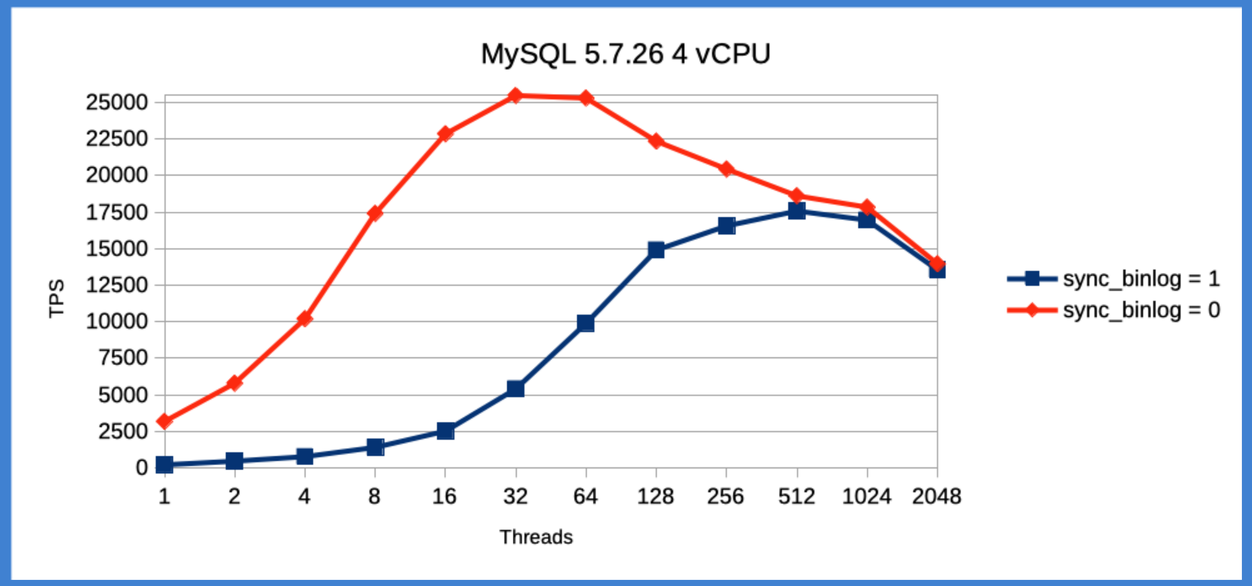
We will quickly explore #2 and 3 in the next slides (#1 is described in [1])

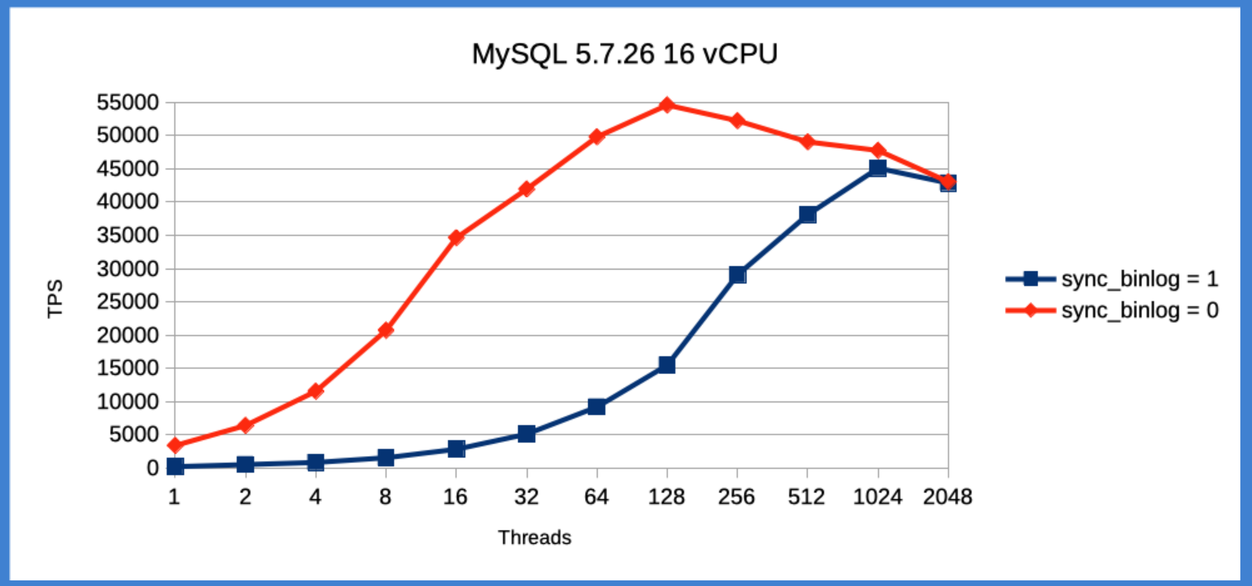
[1]: https://jfg-mysql.blogspot.com/2019/07/master-replication-crash-safety-part-5a-faster-wo-reducing-durability-hardware.html

Avoiding `sync_binlog != 1` [2 of 5]

(The consequences of `sync_binlog != 1` – MySQL.FOSDEM.2020)

Running transactions in parallel on the master (Binary Log Group Commit):





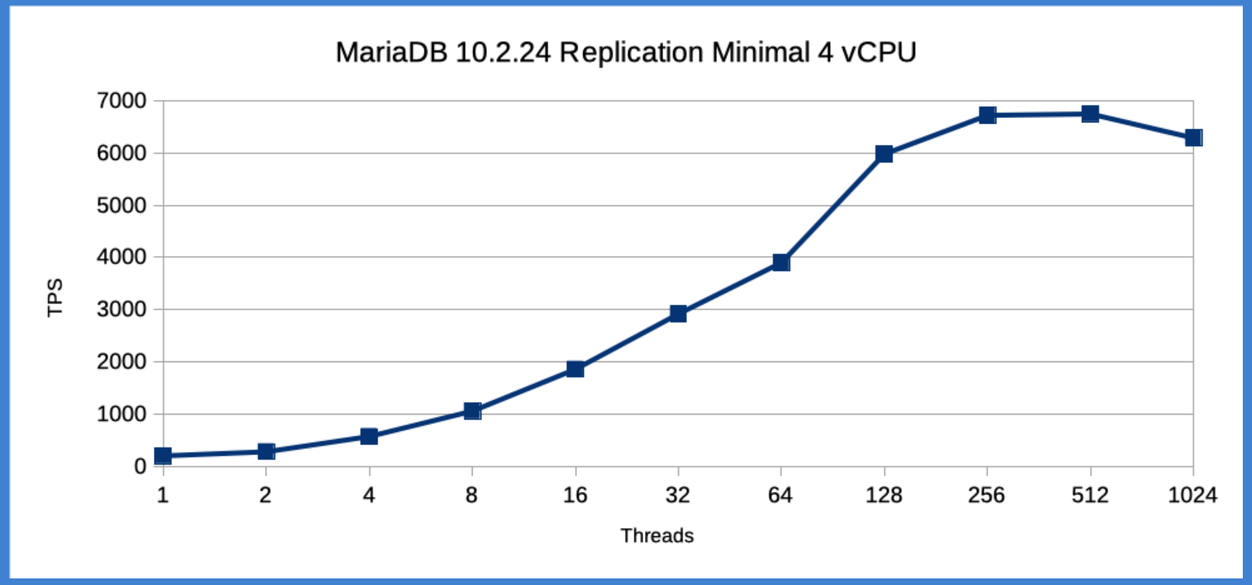
- Very nice increase in TPS with `sync_binlog = 1`, but needs a lot of threads
- And `sync_binlog = 0` can do much more TPS with less threads

Note: benchmarks done in GCP, so not as reliable as dedicated server.
(sysbench insert bench. with SSD persistent disks without secondary index)

Avoiding `sync_binlog != 1` [3 of 5]

(The consequences of `sync_binlog != 1` – MySQL.FOSDEM.2020)

MariaDB 10.0+ Slave Group Commit [1] (named minimal in 10.1+):



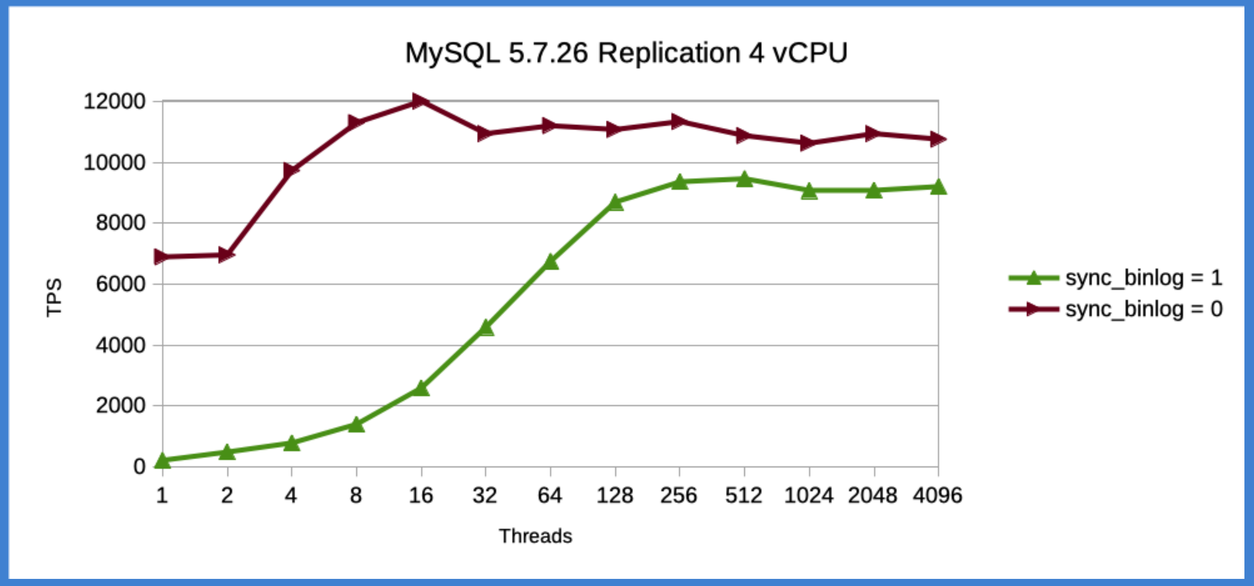
- Nice increase in TPS (without allocating too much extra CPU resources)
- Almost as fast as single-threaded `sync_binlog = 0` !
- But not as good as multi-threaded on the master
  (obviously as transactions are serialized and not run in parallel)

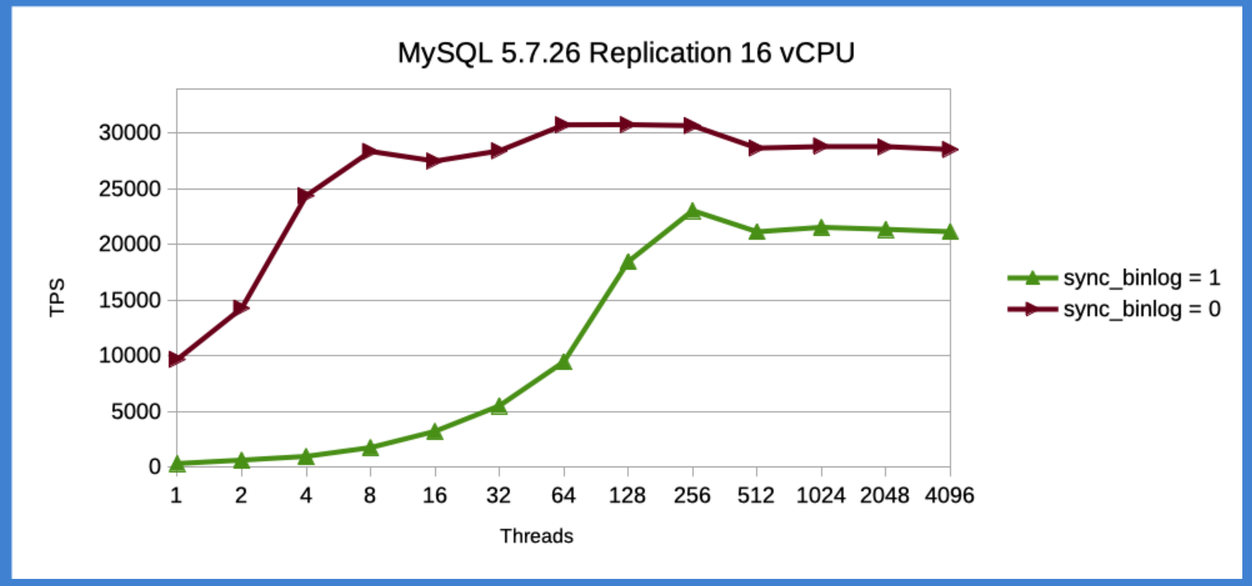
[1]: https://medium.com/booking-com-infrastructure/evaluating-mysql-parallel-replication-part-2-slave-group-commit-459026a141d2

Avoiding `sync_binlog != 1` [4 of 5]

(The consequences of `sync_binlog != 1` – MySQL.FOSDEM.2020)

Parallel Replication (this is a complex subject, not covered in detail here, see [1]):

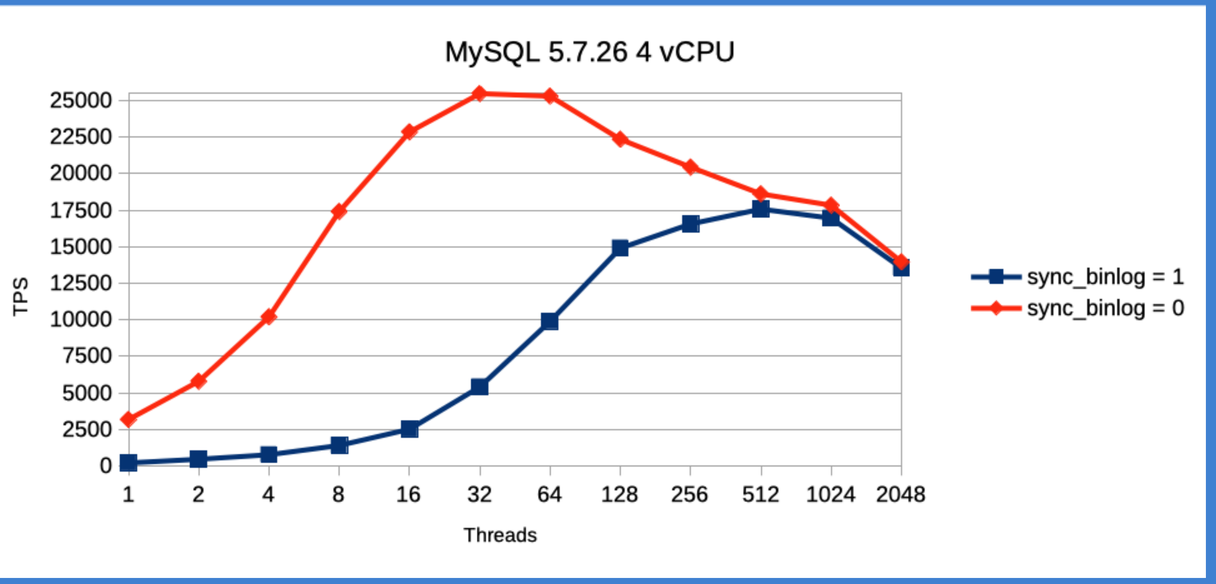


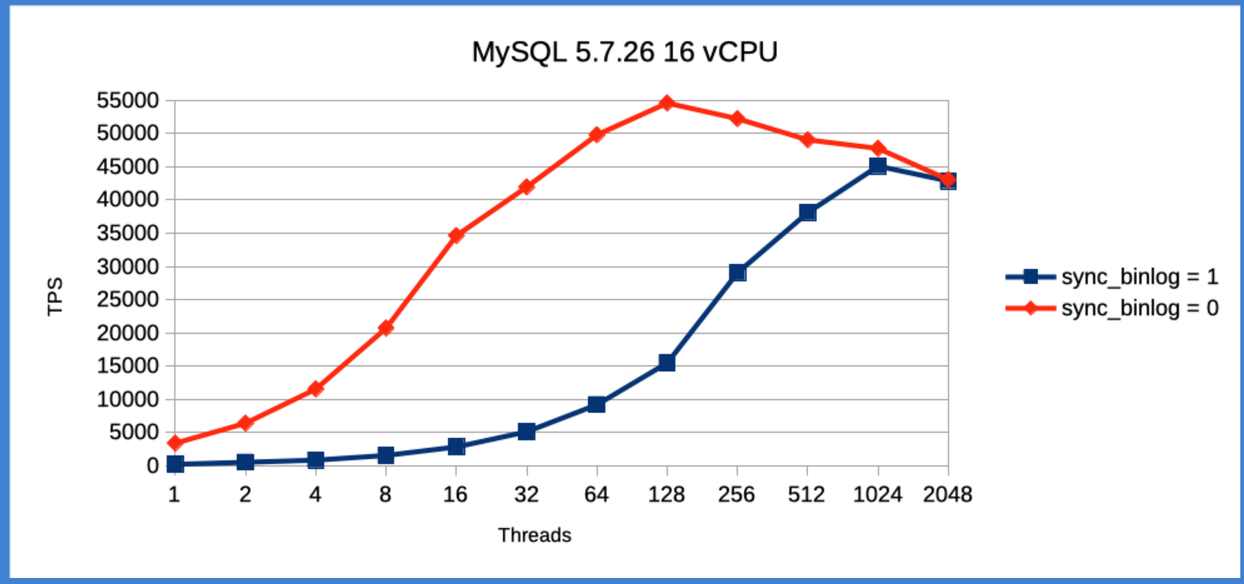


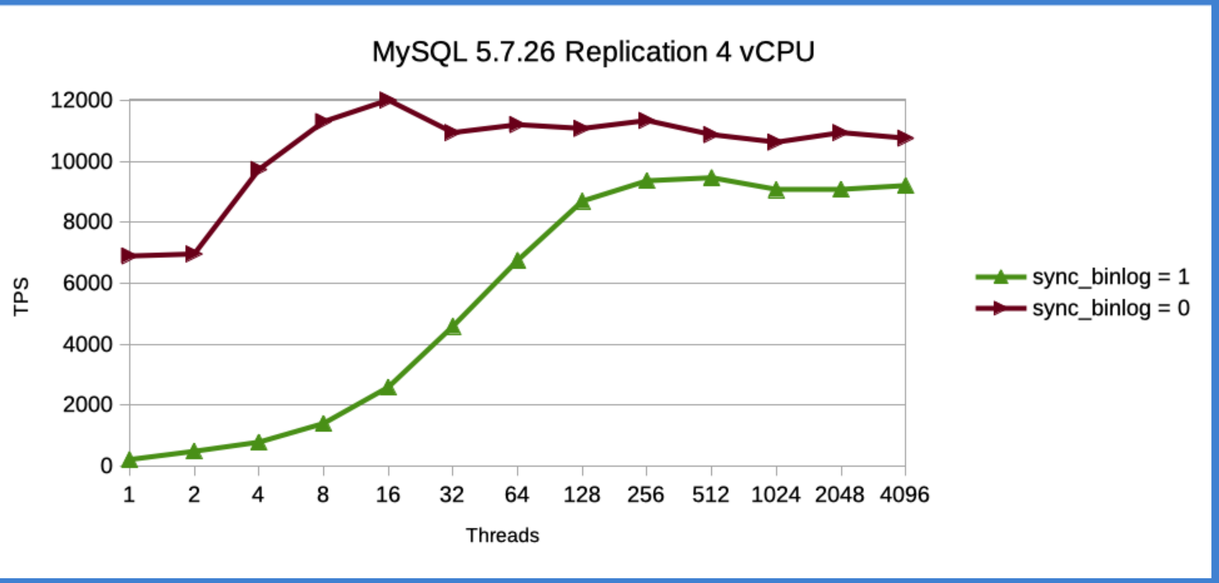
- Nice increase in TPS with `sync_binlog = 1`, but needs a lot of threads
- And `sync_binlog = 0` can do more TPS with less threads
- Sadly, not as good as the master in this case
  (Note that the insert benchmark is probably the worse for Parallel Repl.)

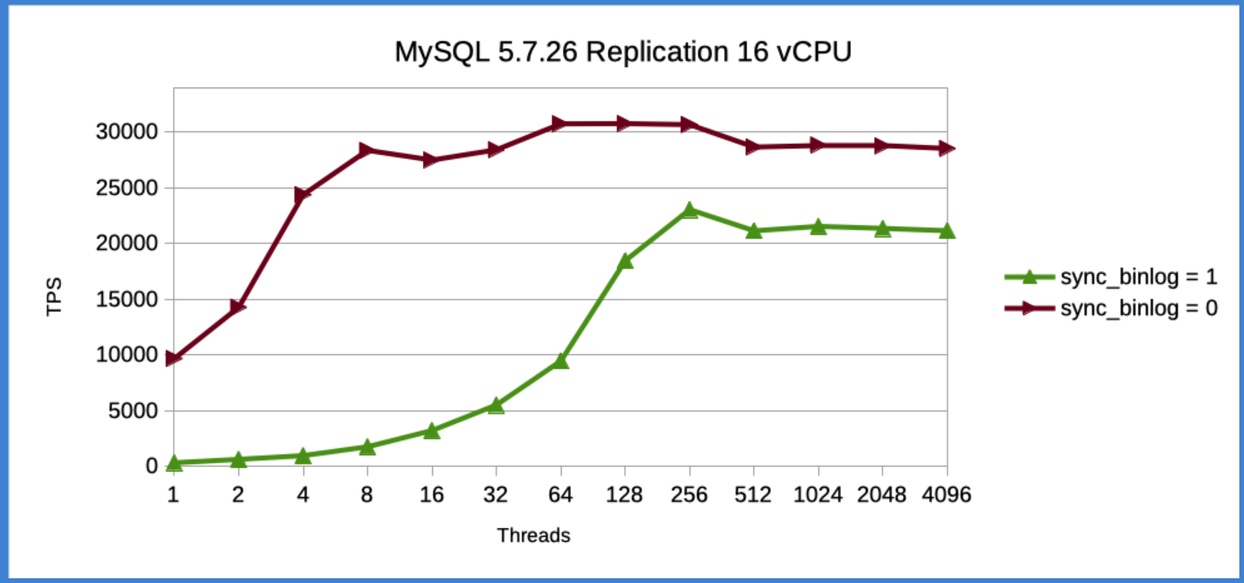
[1]: https://www.slideshare.net/JeanFranoisGagn/the-full-mysql-and-mariadb-parallel-replication-tutorial

Avoiding `sync_binlog != 1` [5 of 5]









Consequences of `sync_binlog != 1` [1 of 6]

(The consequences of `sync_binlog != 1` – MySQL.FOSDEM.2020)

Trx are in an OS RAM buffer after COMMIT, but they are not on disk:

- No data lost after a mysqld crash (data from OS RAM buffer is not lost)
- But loosing data in case of an OS crash
- And after an OS crash, InnoDB and the binary logs are not in sync

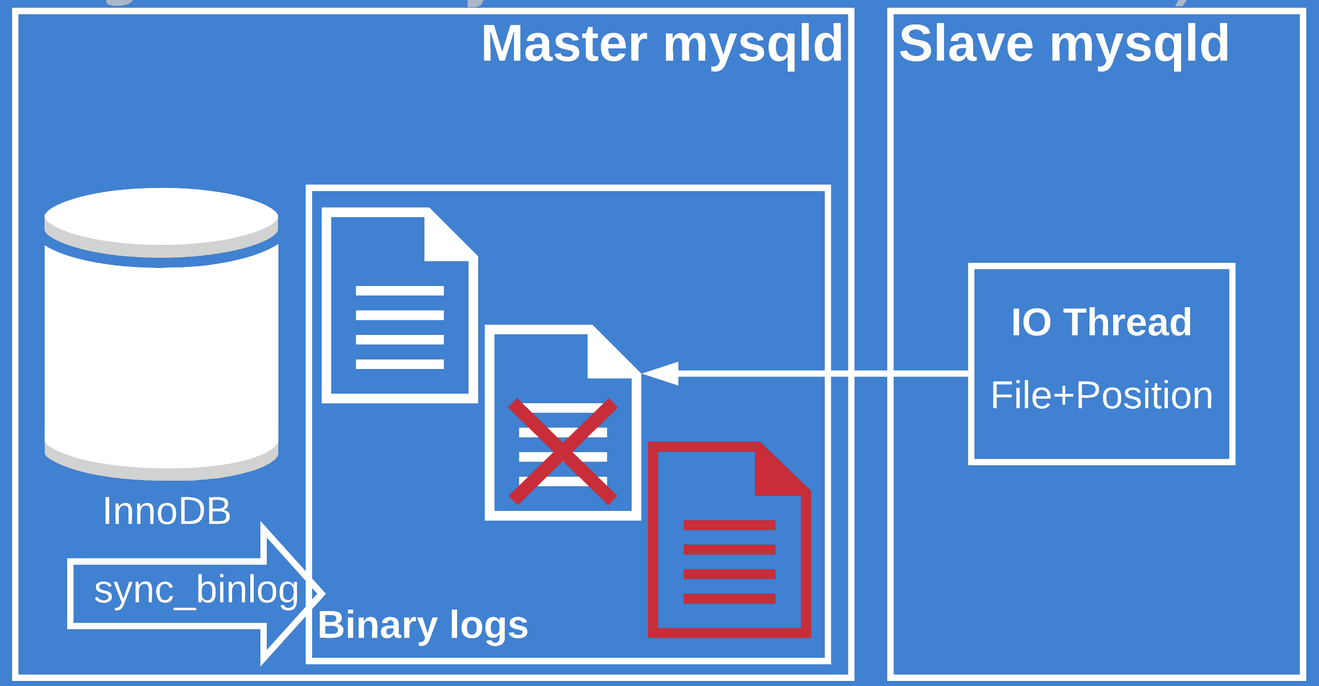
If those transactions are run on the master: …, D, **E**, F, G, H, I, J, **K**, L, …

- On an OS crash, binlog could be flushed up to E, and InnoDB up to K
- So after recovery, InnoDB will have data up to K (L and after is lost) and the transactions F and after are lost from the binary logs

(Note that the scenario where InnoDB looses less than the binlog is more likely as the Redo Logs are flushed every seconds, but the opposite of above might also happen.)

Consequences of `sync_binlog != 1` [2 of 6]

(The consequences of `sync_binlog != 1` – MySQL.FOSDEM.2020)



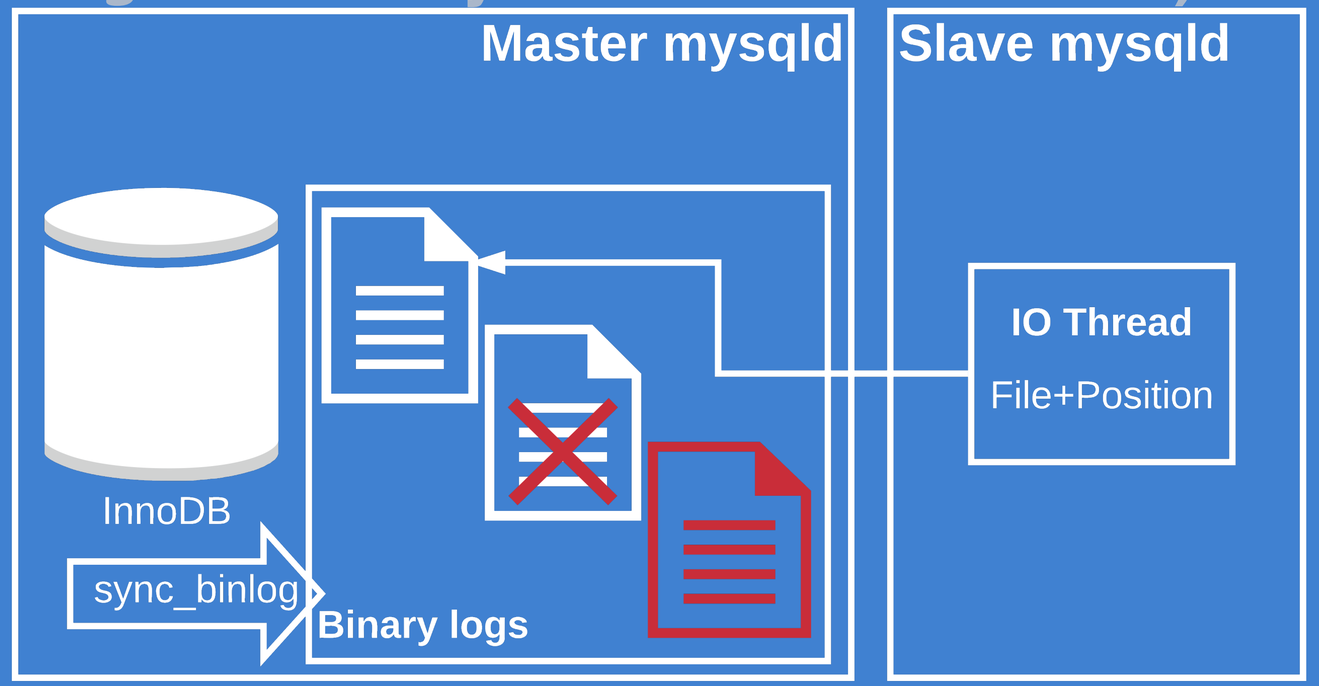
Master OS crash

and rpl. with file+position:

- IO Thread in vanished binlogs
- So slaves executed phantom trx
  (ghost in binlogs, maybe not in InnoDB)
- When the master is restarted:
  - It records trx in new binlog file
  - Most slaves are broken (pointing in vanished binlog), and they might be out-of-sync with each-others
  - Some lagging slave might skip vanished binlogs

Consequences of `sync_binlog != 1` [2 of 6]

(The consequences of `sync_binlog != 1` – MySQL.FOSDEM.2020)



Master OS crash

and rpl. with file+position:

- IO Thread in vanished binlogs
- So slaves executed phantom trx
  (ghost in binlogs, maybe not in InnoDB)
- When the master is restarted:
  - It records trx in new binlog file
  - Most slaves are broken (pointing in vanished binlog),
    and they might be out-of-sync with each-others
  - Some lagging slave might skip vanished binlogs

➢ Broken slaves have more data than the master (→ data drift)

➢ With different data drift on “*lucky*” lagging slaves that might not break

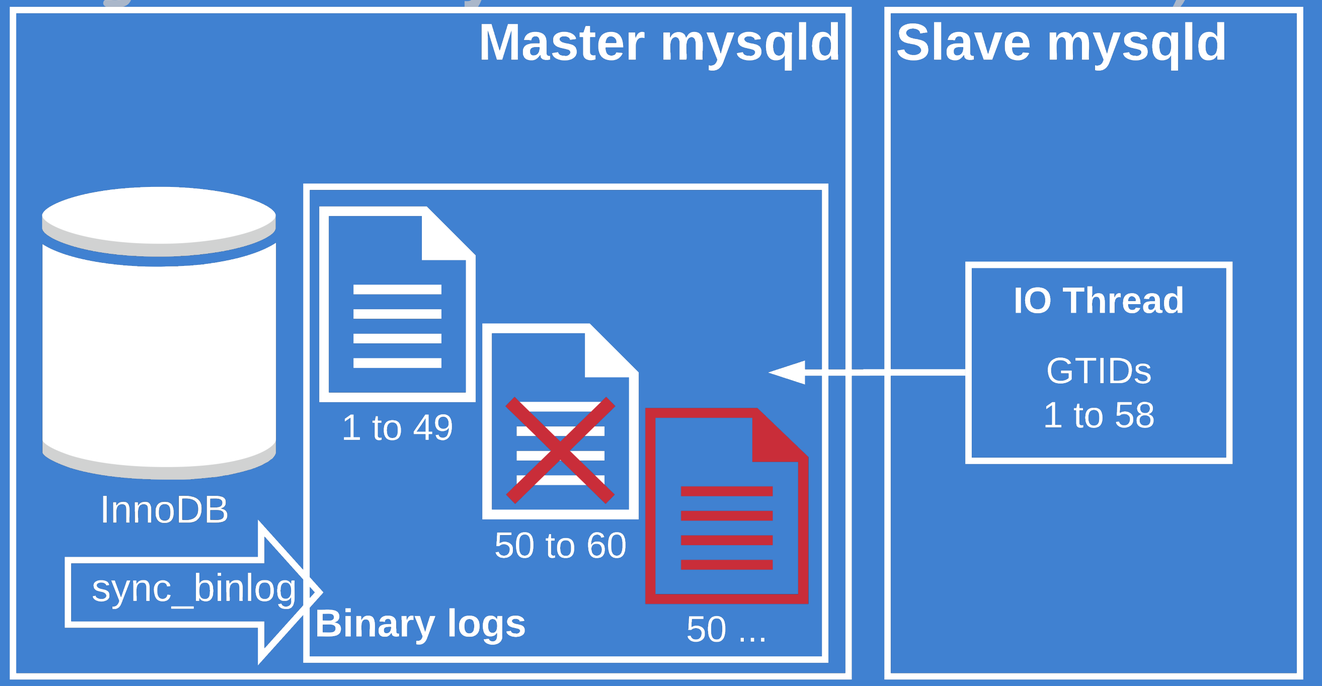
Consequences of `sync_binlog != 1` [3 of 6]

(The consequences of `sync_binlog != 1` – MySQL.FOSDEM.2020)

Master OS crash

and replication with GTID:

- Slave also executed ghost trx vanished from binlogs
- But those are in their GTID state
- A recovered master reuses GTIDs of the vanished trx



- Slaves *magically* reconnect to the master (`MASTER_AUTO_POSITION = 1`)
  1. If master has not reused all ghost GTIDs, then the slave breaks
  2. If it has, then the slave skips the new transactions → more data drift (in illustration, the slave will skip new 50 to 58 as it has the old one)

Consequences of `sync_binlog != 1` [4 of 6]

(The consequences of `sync_binlog != 1` – MySQL.FOSDEM.2020)

Slave OS crash and replication with file+position:

- If using replication crash-safe configuration [1] → no problem !
  (well, there is still the corrupted binlog problem, but this is covered in master crashes)
  - With crash-safe replication, the position is stored in an InnoDB table
  - So replication will resume at the exact place where InnoDB rewinded
  - Obviously, if this slave has binlog enabled, they will be corrupted (same problems as for a master crash)

[1]: https://medium.com/booking-com-infrastructure/better-crash-safe-replication-for-mysql-a336a69b317f

Slave OS crash and replication with GTID:

- If binlogs disabled on the slave (only possible with 5.7+) → no problem !
  - With binlogs disabled, the GTID state is stored in an InnoDB table
  - So replication will resume at the exact place where InnoDB rewinded

Consequences of `sync_binlog != 1` [5 of 6]

(The consequences of `sync_binlog != 1` – MySQL.FOSDEM.2020)

Slave OS crash and replication with GTID:

- If binary logs are enabled → **problems as the GTID state is lost** !
  - With binlogs enabled, the GTID state is stored in the binary logs
  - As the binlogs rewinded, replication will resume at this past position
  - And as InnoDB is not in sync with the binlogs, replication will probably break
  - Disappointing: the GTID table is not updated after each trx (only on binlog rotation)
    Bug#92109: Please make replication crash-safe with GITD and less durable setting (bis)

(Note: this is not a problem with MariaDB as its GTID state is stored in a table.)

Consequences of `sync_binlog != 1` [6 of 6]

(The consequences of `sync_binlog != 1` – MySQL.FOSDEM.2020)

It looks like MySQL 8.0.17 has the GTID position in InnoDB Logs [1]

- But unclear if this solves the problem
  (no communication from Oracle on this subject including no bugs closed)
- And GTID position in InnoDB does not work for MyRocks or other Storage Engines

➢ Updating the GTID table after each trx would be better IMHO

[1]: WL#9211: InnoDB: Clone Replication Coordinates: https://dev.mysql.com/worklog/task/?id=9211

Mitigating `sync_binlog != 1` [1 of 2]

(The consequences of `sync_binlog != 1` – MySQL.FOSDEM.2020)

When running with `sync_binlog != 1`:

- The binlogs – of the master or slave – cannot be trusted after an OS crash

- On a master, mysqld restarting after OS crash leads to broken slaves and data drift
- ➢ After an OS crash, make sure no slaves reconnect to the recovered master (`OFFLINE_MODE = ON` in conf file - **failing-over to a slave is the way forward**)

- On slaves, having mysqld restarts after such a crash leads to truncated binlogs
- ➢ After an OS crash, purge all binlogs on the recovered slave (`RESET MASTER`) (A little more complicated with GTID, see next slide)

- Intermediate Masters (*IM*) are both master and slaves
- ➢ After an OS crash of an intermediate master, make sure no slaves reconnect to it
- ➢ And purge all its binary logs

Mitigating `sync_binlog != 1` [2 of 2]

(The consequences of `sync_binlog != 1` – MySQL.FOSDEM.2020)

When running a GTID slave with `sync_binlog != 1`:

- The GTID state is corrupted after an OS crash → you should restore a backup
- But file+pos. in a table can be trusted, so there is hope to avoid restoring a backup if single-threaded replication or no gap in relay log execution (always the case if `slave_preserve_commit_order = ON`)

This is how to salvage a GTID slave after an OS crash (this is voodoo):

- Make sure mysql starts with replication disabled (`skip-slave-start`)
- Note the GTID position (`gtid_executed`) and wipe the binlogs (`RESET MASTER`)
- Start replication with file+position (`CHANGE MASTER TO AUTO_POSITION=0; START SLAVE`)
- After running a few transactions, stop replication and restore the GTID state (this is the voodoo part: you will have to figure it out on your own and this uses the noted GTID position above)

Conclusion [1 of 2]

(The consequences of `sync_binlog != 1` – MySQL.FOSDEM.2020)

- We saw why someone would run with `sync_binlog != 1`
- We understood the consequences, and saw how to avoid and mitigates them
- Using `sync_binlog != 1` will be very common in the Cloud
  (because of the high latency of network disks)
- To be fully Cloud Ready,
  MySQL should make running with `sync_binlog != 1` easier:
  - Auto OFFLINE_MODE after an OS crash when `sync_binlog != 1` ?
  - Until MySQL is replication crash-safe with GTID and `sync_binlog != 1`:
    Auto `skip-slave-start` after an OS crash when `sync_binlog != 1` ?
  - Bug#98448: Please make running MySQL with sync_binlog != 1 safer
- Rant: MySQL implementation of GTID makes things complicated !
- Rant2: GTID state in a table, in the binlogs and in the InnoDB Logs
  → solving problems by patching things, no consistent vision, cleanup needed !

Conclusion [2 of 2]

(The consequences of `sync_binlog != 1` – MySQL.FOSDEM.2020)

- More GTID rant: gtid_executed shared by a DBA in a MySQL Slack channel:
  (how to clean this up is left as an exercise for the audience)
  (this was reported to Oracle years ago, no good way of dealing with this provided yet)

```
000d4a8b-ee59-11e9-ab79-008cfa5440e4:1-18448,0048b71c-d850-11e9-bc30-008cfa542442:1-18563,005f87ce-ade3-11e9-9c87-008cfa542c9e:1-82355,008367c9-971f-
11e9-95d9-008cfa544064:1-80594,00f2a2a9-ab92-11e9-ac53-008cfa544064:1-81871,0100879f-e8fc-11e9-b340-008cfa5440e4:1-18531,0103609f-d53a-11e9-828e-
008cfa542442:1-18206,011f5cb1-a92c-11e9-9fdb-008cfa542c9e:1-83848,018c0de3-ba7e-11e9-a890-008cfa542c9e:1-83934,02a5faa1-98b6-11e9-97cd-008cfa542c9e:1-
85046136,037f8134-c89f-11e9-8e3a-008cfa542c9e:1-18332,03ba38a5-a6db-11e9-8af2-008cfa5440e4:1-74794,03de556d-eb30-11e9-84df-008cfa542c9e:1-
18466,03eb1177-cd50-11e9-9949-008cfa542442:1-1818897,04d72700-b5c7-11e9-a0a9-008cfa542c9e:1-85470,05a10dbe-d5ff-11e9-8e87-008cfa544064:1-18386,068e6393-
c323-11e9-ac17-008cfa542442:1-103482,069bb1f8-caf4-11e9-919f-008cfa542c9e:1-18420,069eb739-e1c0-11e9-b03d-008cfa5440e4:1-18496,06cde7cb-989c-11e9-bd1e-
008cfa5440e4:1-935909,07035800-e036-11e9-9775-008cfa5440e4:1-18497,071c672a-be70-11e9-a302-008cfa542442:1-138571,075883a2-d211-11e9-a3a2-008cfa542c9e:1-
18225,07d8cfd0-a79e-11e9-9fbb-008cfa544064:1-84475,08241557-db7f-11e9-9801-008cfa542442:1-18287,08eee297-e4ed-11e9-8eb3-008cfa542c9e:1-18230,090590ae-
e5ac-11e9-8b4f-008cfa544064:1-18527,0939cfc6-d462-11e9-b8f1-008cfa542c9e:1-18275,09d48f25-c971-11e9-99ec-008cfa542c9e:1-18493,0a121d94-cee6-11e9-b5ab-
008cfa544064:1-18595,0a1ae309-d6c4-11e9-b732-008cfa542442:1-18168,0aa0e300-ca2d-11e9-a81b-008cfa542c9e:1-18347,0b3d7eed-b29a-11e9-87d6-008cfa5440e4:1-
77023,0b7c58fd-cb01-11e9-bfcd-008cfa544064:1-18508,0bb517b6-cd52-11e9-8903-008cfa542c9e:1-18173,0bcfd1ac-b488-11e9-89b8-e4434b27ec50:1-39442,0c34eca3-
ad1e-11e9-bf76-008cfa542c9e:1-82933,0c46cff1-b823-11e9-bbb4-008cfa5440e4:1-7001796,0c72d5c7-ee57-11e9-9cde-a0369f9432f4:1-50694858,0d36c194-a7a2-11e9-
965a-008cfa542c9e:1-7572323,0d534018-ade1-11e9-934e-008cfa544064:1-83481,0e751980-cc8f-11e9-92ef-008cfa5440e4:1-17609,0ea04e41-b757-11e9-a049-
008cfa542442:1-80864,0f0e919e-b1d5-11e9-b22a-008cfa5440e4:1-61341,0f7c6956-bc0a-11e9-8628-008cfa542442:1-130513,10528142-95c0-11e9-8218-008cfa5440e4:1-
38346,10636758-bccf-11e9-bdf4-008cfa542c9e:1-133216,1076c777-b110-11e9-8dfa-008cfa542c9e:1-85695,110d67aa-d603-11e9-b24d-008cfa542442:1-18271,11d66846-
d601-11e9-b3da-008cfa542c9e:1-1671806,11fb613f-aeaa-11e9-87ad-008cfa542c9e:1-81172,1202852e-d854-11e9-bb27-008cfa542c9e:1-18331,12259ccd-a867-11e9-868f-
008cfa5440e4:1-7005845,128e5b58-ac59-11e9-a987-008cfa542c9e:1-84165,128ff97f-d2d8-11e9-85ff-008cfa542442:1-1668330,1395d36e-d39b-11e9-9b67-
008cfa542c9e:1-18186,1398ce80-e73e-11e9-b6d6-008cfa544064:1-18295,13aa8365-ca2f-11e9-9b9c-008cfa544064:1-18393,13b2a7f8-fefc-11e9-8d3c-b4969136e8e0:1-
3,13ca371a-d919-11e9-9e7b-008cfa5440e4:1-18470,13d4cd9c-c0c1-11e9-8a19-008cfa5440e4:1-121684,13fa0ac6-b04b-11e9-ab02-008cfa544064:1-83899,14c9fee1-bb45-
11e9-bc9b-008cfa542c9e:1-136184,1509d718-9a40-11e9-bca9-008cfa542442:1-961253,15a5c07d-ddd0-11e9-b050-008cfa542442:1-18432,15c06864-b361-11e9-8e4c-
008cfa542c9e:1-81510,15da65c1-a6dd-11e9-b919-008cfa542c9e:1-83624,1695a57f-a9f3-11e9-b8d9-008cfa542c9e:1-81772,1730b2c1-a63f-11e9-a9a7-008cfa542442:1-
4112383,176edb77-c96a-11e9-a4ee-008cfa5440e4:1-18533,178da034-9b07-11e9-a1bc-008cfa542442:1-962266,17c75d9f-ab94-11e9-8d5e-008cfa542c9e:1-
84025,18bcec4e-b818-11e9-ac66-008cfa5440e4:1-78637,1936ceb2-c327-11e9-aead-008cfa542442:1-104809,196ee579-e1c2-11e9-8f96-008cfa542c9e:1-18151,19e72316-
ea67-11e9-b4cf-a0369f9432f4:1-18255,1a094a28-ea6b-11e9-a501-008cfa542c9e:1-18296,1a5f9739-ba80-11e9-9bbf-008cfa542442:1-138749,1a88b452-d6ca-11e9-ab67-
008cfa5440e4:1-18288,1ac1bdef-b753-11e9-84b5-008cfa544064:1-35746,1b425cb0-d78f-11e9-8ed5-008cfa542442:1-18153,1b730452-d852-11e9-8190-008cfa5440e4:1-
18538,1b7c329b-df6f-11e9-9906-008cfa544064:1-1663592,1bb9e21f-e679-11e9-ab63-008cfa5440e4:1-18516,1bea1785-c8a5-11e9-a768-008cfa542c9e:1-1671715
```

Links [1 of 3]

(The consequences of `sync_binlog != 1` – MySQL.FOSDEM.2020)

- On the consequences of sync_binlog != 1 (part #1)
  https://jfg-mysql.blogspot.com/2018/10/consequences-sync-binlog-neq-1-part-1.html
- Evaluating MySQL Parallel Replication Part 2: Slave Group Commit
  https://medium.com/booking-com-infrastructure/evaluating-mysql-parallel-replication-part-2-slave-group-commit-459026a141d2
- Better Crash-safe replication for MySQL
  https://medium.com/booking-com-infrastructure/better-crash-safe-replication-for-mysql-a336a69b317f

- Bug#70659: Make crash safe slave work with gtid + less durable settings
- Bug#92109: Please make repl. crash safe with GITD and less durable setting (bis)
- Bug#98448: Please make running MySQL with sync_binlog != 1 safer

- WL#9211: InnoDB Clone Replication Coordinates

Links [2 of 3]

(The consequences of `sync_binlog != 1` – MySQL.FOSDEM.2020)

- MySQL Master High Availability and Failover: more thoughts
  https://jfg-mysql.blogspot.com/2019/02/mysql-master-high-availability-and-failover-more-thoughts.html
- MySQL Scalability and Reliability for Replicated Environment
  https://www.slideshare.net/JeanFranoisGagn/mysql-scalability-and-reliability-for-replicated-environment-150876075

- The Full MySQL and MariaDB Parallel Replication Tutorial
  https://www.slideshare.net/JeanFranoisGagn/the-full-mysql-and-mariadb-parallel-replication-tutorial
- Evaluating MySQL Parallel Replication Part 4: More Benchmarks in Production
  https://medium.com/booking-com-infrastructure/evaluating-mysql-parallel-replication-part-4-more-benchmarks-in-production-49ee255043ab

Links [3 of 3]

(The consequences of `sync_binlog != 1` – MySQL.FOSDEM.2020)

- Pseudo-GTID and Orchestrator:
  https://github.com/github/orchestrator/blob/master/docs/pseudo-gtid.md
  https://speakerdeck.com/shlominoach/pseudo-gtid-and-easy-mysql-replication-topology-management

- How Binary Logs (and Filesystems) Affect MySQL Performance
  https://www.percona.com/blog/2018/05/04/how-binary-logs-and-filesystems-affect-mysql-performance/
- Fsync Performance on Storage Devices
  https://www.percona.com/blog/2018/02/08/fsync-performance-storage-devices/

MessageBird is hiring

messagebird.com/en/careers/

Thanks !

by Jean-François Gagné
Senior Infrastructure Engineer / System and MySQL Expert
jeanfrancois AT messagebird DOT com / @jfg956 #FOSDEM #MySQLDevRoom