Booking.com

How B.com avoids and deals with replication lag

Jean-François Gagné (System Engineer)

April 9, 2017 – MariaDB Developer Meeting 2017

Booking.com

- Based in Amsterdam since 1996
- Online Hotel/Accommodation/Travel Agent (OTA):
  - +1.202.000 properties in 227 countries
  - +1.200.000 room nights reserved daily
  - +40 languages (website and customer service)
  - +13.000 people working in 187 offices worldwide
- Part of the Priceline Group

- And we use MySQL:
  - Thousands (1000s) of servers, ~90% replicating
  - >150 masters: ~30 >50 slaves & ~10 >100 slaves

Session Summary

1. MySQL replication at Booking.com
2. Replication lag: what/how/why
3. Bad solutions to cope with lag
4. Booking.com solution to cope with lag
5. Improving Booking.com solution
6. Galera and Group Replication

MySQL replication at Booking.com

- Typical Booking.com MySQL replication deployment:

```
+----+
|  M |
+----+
  |
  +------+--  ...  --+---------------+---------  ...
  |      |           |               |
+----+ +----+      +----+          +----+
|  S1| |  S2|      |  Sn|          |  M1|
+----+ +----+      +----+          +----+
                                     |
                                     +--  ...  --+
                                     |           |
                                   +----+      +----+
                                   |  T1|      |  Tm|
                                   +----+      +----+
```

Why does lag happen ?

- In which condition can lag be experienced ?
  - Too many transactions for replication to keep up:
    capacity problem, fix by scaling (sharding, parallel replication, …)
  - Long transactions:
    self induced, to fix by a developer in the application
  - Too aggressive “batch” workload on the master:
    optimize the batches or slow them down

Lag consequences

- What are the consequences of lag ?
    - Stale reads on slaves (but this is not necessarily a problem)

- When do stale reads become a problem ?
    - A user changes his email address but still sees the old one
    - A hotel changes its inventory but still sees old availability
    - A user books a hotel but does not see it in his reservations

Bad solution to cope with lag

- Bad solution #1: falling back to reading from the master
  - If slaves are lagging, maybe we should read from the master
  - This looks like an attractive solution to avoid stale reads
  - But this does not scale (why are you reading from slaves…)
  - This will cause a sudden load on the master (in case of lag)
  - And it might cause an outage on the master (and this would be bad)

- It might be better to fail a read than to fallback to (and kill) the master

Bad solution to cope with lag (bis)

- Bad solution #2: retry on another slave
  - When reading from a slave: if lag, then retry on another slave
  - This scales better and is OK-ish (when few slaves are lagging)
  - But what happens if all slaves are lagging ?
  - Increased load (retries) can slowdown replication
  - This might overload the slaves and cause a good slave to start lagging
  - In the worst case, this might kill slaves and cause a domino effect

- Again: probably better to fail a read than to cause a bigger problem

Coping with lag @ Booking.com

- Booking.com solution: “waypoint”
  - Creating a waypoint is similar to creating a “read view”
  - Waiting for a waypoint is similar to waiting for a slave to catch-up

- Booking.com implementation:
  - Table: db_waypoint (a waypoint is a row in that table)
  - API function: commit_wait(timeout) → (err_code, waypoint)
    - INSERTs a waypoint and waits – until timeout – for its arrival on a slave
    - This is the same a creating a “read view” and “forcing” it on a slave
  - API function: waypoint_wait(waypoint, timeout) → err_code
    - Waits for a waypoint – until timeout – on a slave
    - This is the same as “waiting for a slave to catch-up”
  - Garbage collection: cleanup job that DELETEs old waypoints

Booking.com

Coping with lag @ Booking.com’

- Booking.com deployment:
  - Throttling batches:
    - use commit_wait with a high timeout
    - use “small” transactions (chunks of 100 to 1000 rows)
    - and sleep between chunks
  - Protect from stale reads after writing:
    - commit_wait with zero timeout
    - store the waypoint in web session
    - and waypoint_wait when reading

Improving B.com waypoints

- The waypoint design and implementation still suits us.
- Sometime, we have a “fast” slave problem:
  - Throttling batches on a fast slave is sub-optimal
  - But this does not happen often in practice though
  - And it would be easy to fix: “find the slowest slave (or a slow slave)”
- But starting from scratch, we might do things differently:
  - Inserting, deleting and purging waypoint could be simplified
  - And we could get rid of the waypoint table

Improving B.com waypoints'

- GTIDs as waypoint
    - Get the GTID of the last transaction:
        - last_gtid session variable in MariaDB Server

From https://mariadb.com/kb/en/mariadb/master_gtid_wait/:

MASTER_GTID_WAIT() can also be used in client applications together with the last_gtid session variable. This is useful in a read-scaleout replication setup, where the application writes to a single master but divides the reads out to a number of slaves to distribute the load. In such a setup, there is a risk that an application could first do an update on the master, and then a bit later do a read on a slave, and if the slave is not fast enough, the data read from the slave might not include the update just made, possibly confusing the application and/or the end-user. One way to avoid this is to request the value of last_gtid on the master just after the update. Then before doing the read on the slave, do a MASTER_GTID_WAIT() on the value obtained from the master; this will ensure that the read is not performed until the slave has replicated sufficiently far for the update to have become visible.

Improving B.com waypoints’

- GTIDs as waypoint:
  - Get the GTID of the last transaction :
    - last_gtid session variable in MariaDB Server
    - gtid_executed global variable in Oracle MySQL (get all executed GTIDs)
    - the last GTID can also be requested in the OK packet (only Oracle MySQL) (session_track_gtids variable and mysql_session_track_get_{first,next} API functions)
  - Waiting for GTID:
    - MASTER_GTID_WAIT in MariaDB Server
    - WAIT_FOR_EXECUTED_GTID_SET in Oracle MySQL
- But not portable (replicating from MySQL to MariaDB or vice-versa)

Improving B.com waypoints”

- Binary log file and position as waypoint:
  - MASTER_POS_WAIT
  - However this breaks using intermediate masters
  - But it is OK with Binlog Servers[1]
    (in a Binlog Server deployment, the binlog file and position is a GTID)
  - But currently no way of getting file and position after committing

[1]: https://blog.booking.com/abstracting_binlog_servers_and_mysql_master_promotion_wo_reconfiguring_slaves.html

Improving B.com waypoints’’’

- Feature requests:
  - Bug#84747: Expose last transaction GTID in a session variable.
  - Bug#84748: Request transaction GTID in OK packet on COMMIT (without needing a round-trip).
  - MDEV-11956: Get last_gtid in OK packet.
  - Bug#84779: Expose binlog file and position of last transaction.
  - MDEV-11970: Expose binlog file and position of last transaction.

Booking.com

Improving B.com waypoints’’’’

- Better solution for throttling:
  - Connecting to a (the right) slave is a hurdle
  - Having the information about slave state on the master would be great
  - A plugin exists for something close to that: semi-sync
  - Using this to track transaction execution on slaves would be great
  - This is the No-Slave-Left-Behind MariaDB Server Patch

No-Slave-Left-Behind

- No-Slave-Left-Behind MariaDB Server patch[1]:
  - the semi-sync reply also reports SQL-thread position
  - transactions are kept in the master plugin until executed by one slave
  - the slave lag can be estimated from above
  - client-threads wait before commit until lag is acceptable
  - (Thanks Jonas Oreland and Google)

- This could easily be modified to implement commit_wait

[1]: https://jira.mariadb.org/browse/MDEV-8112

Booking.com

Galera and Group Replication

- Group Replication allows to “read from the primary”
- Galera has a feature to protect against slate reads:
  - wsrep_sync_wait=1 (or other values for other type of consistency)
    the node blocks new queries while the database server catches up with all updates made in the cluster to the point where the check was begun
- wsrep_sync_wait=1 is very similar to checkpointing
- No-Slave-Left-Behind could also apply here:
  - No-Stale-Node-After-Write

Links

- Booking.com:
  - https://blog.booking.com/
- MariaDB Server last_gtid (thanks Kristian Nielsen for implementing this):
  - https://mariadb.com/kb/en/mariadb/master_gtid_wait/
- Binlog Server:
  - https://blog.booking.com/abstracting_binlog_servers_and_mysql_master_promotion_wo_reconfiguring_slaves.html
- No-Slave-Left-Behind MariaDB Server patch:
  - https://jira.mariadb.org/browse/MDEV-8112 (thanks Jonas Oreland and Google)

Links’

- Pull request to extent Perl-DBI for reading GTID in OK packet:
  - https://github.com/perl5-dbi/DBD-mysql/pull/77 (thanks Daniël van Eeden)
- Bug reports/Feature requests:
  - Bug#84747: Expose last transaction GTID in a session variable.
  - Bug#84748: Request transaction GTID in OK packet on COMMIT (without needing a round-trip).
  - Bug#84779: Expose binlog file and position of last transaction.
  - MDEV-11956: Get last_gtid in OK packet.
  - MDEV-11970: Expose binlog file and position of last transaction.

Thanks

Jean-François Gagné
jeanfrancois DOT gagne AT booking.com