How to Avoid Pitfalls in Schema Upgrade

with Galera

February 2, 2020

Sveta Smirnova
Sveta Smirnova

- MySQL Support engineer
- Author of
  - MySQL Troubleshooting
  - JSON UDF functions
  - FILTER clause for MySQL
- Speaker
  - Percona Live, OOW, Fosdem, DevConf, HighLoad...
Table of Contents

- TOI
- RSU
- pt-online-schema-change (pt-osc)
Introduction

- Galera Replication Library
  - Provides synchronous replication for MySQL
Introduction

- Galera Replication Library
  - Provides synchronous replication for MySQL
- Galera Clusters
  - MariaDB Galera Cluster
  - Percona XtraDB Cluster
  - Galera Cluster for MySQL
How Galera works

- Data modification happens on a node
- Optimistic locking control
How Galera works

- Data modification happens on a node
- Optimistic locking control
- At the COMMIT time
  - Broadcasts write set for the cluster
  - Waits confirmation of the successful update
    - From all other nodes

Yes  Commits transaction locally
No   Rollbacks transaction
Data Updates

• Committed on all nodes or nowhere
• Safe
Challenges of DDL

• Replicated independently from storage engine
Challenges of DDL

- Replicated independently from storage engine
- Changes may affect query results
  - Adding/removal of UNIQUE keys
  - Adding/removal columns
  - Changing column definition
Challenges of DDL

- Replicated independently from storage engine
- Changes may affect query results
- Modification can happen on any node
  - The schema must be upgraded before DML
  - There is no way to rollback schema upgrade
  - MDLs are set only on one node
    - Not across the cluster
    - Not possible to rely on them for all nodes
    - Additional control required
Total Order Isolation (TOI)

• DDL changes are replicated in the same order regarding other transactions
• All nodes are in the absolutely same state at any point of time
TOI: Illustration

• 3-nodes cluster
  • Node A
  • Node B
  • Node C
<table>
<thead>
<tr>
<th>Node A</th>
<th>Node B</th>
<th>Node C</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSERT(103)</td>
<td>UPDATE(101)</td>
<td>SELECT(100)</td>
</tr>
<tr>
<td>UPDATE(104)</td>
<td>INSERT(102)</td>
<td>INSERT(112)</td>
</tr>
<tr>
<td>ALTER(105)</td>
<td>DELETE(108)</td>
<td>SELECT(113)</td>
</tr>
<tr>
<td></td>
<td>UPDATE(109)</td>
<td>UPDATE(114)</td>
</tr>
</tbody>
</table>
## TOI: Illustration

- **Queries status**

<table>
<thead>
<tr>
<th>Node A</th>
<th>Node B</th>
<th>Node C</th>
</tr>
</thead>
<tbody>
<tr>
<td>➡️ INSERT(103)</td>
<td>➡️ UPDATE(101)</td>
<td>➡️ SELECT(100)</td>
</tr>
<tr>
<td>➡️ UPDATE(104)</td>
<td>➡️ INSERT(102)</td>
<td>🔄 INSERT(112)</td>
</tr>
<tr>
<td>🔥 ALTER(105)</td>
<td>🔥 DELETE(108)</td>
<td>🔥 SELECT(113)</td>
</tr>
<tr>
<td></td>
<td>🔥 UPDATE(109)</td>
<td>🔥 UPDATE(114)</td>
</tr>
</tbody>
</table>
TOI: Illustration

• ALTER in progress

Node A

► ALTER(105)

Node B

● DELETE(108)
● UPDATE(109)

Node C

● INSERT(112)
● SELECT(113)
● UPDATE(114)
TOI: Illustration

- ALTER finished

Node A
- DELETE(108)
- UPDATE(109)

Node B

Node C
- INSERT(112)
- SELECT(113)
- UPDATE(114)
### PROCESSLIST: DML before ALTER

DML node> select DB, COMMAND, TIME, STATE, INFO from information_schema.processlist WHERE DB='sbtest';

<table>
<thead>
<tr>
<th>DB</th>
<th>COMMAND</th>
<th>TIME</th>
<th>STATE</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>sbtest</td>
<td>Query</td>
<td>1</td>
<td>wsrep: initiating pre-commit for write set (2886)</td>
<td>COMMIT</td>
</tr>
<tr>
<td>sbtest</td>
<td>Query</td>
<td>1</td>
<td>wsrep: initiating pre-commit for write set (2888)</td>
<td>COMMIT</td>
</tr>
<tr>
<td>sbtest</td>
<td>Query</td>
<td>1</td>
<td>wsrep: initiating pre-commit for write set (2884)</td>
<td>COMMIT</td>
</tr>
<tr>
<td>sbtest</td>
<td>Query</td>
<td>1</td>
<td>updating</td>
<td>DELETE FROM sbtest1..</td>
</tr>
<tr>
<td>sbtest</td>
<td>Query</td>
<td>1</td>
<td>wsrep: initiating pre-commit for write set (2887)</td>
<td>COMMIT</td>
</tr>
<tr>
<td>sbtest</td>
<td>Query</td>
<td>0</td>
<td>wsrep: initiating pre-commit for write set (2889)</td>
<td>COMMIT</td>
</tr>
<tr>
<td>sbtest</td>
<td>Query</td>
<td>1</td>
<td>wsrep: initiating pre-commit for write set (2885)</td>
<td>COMMIT</td>
</tr>
<tr>
<td>sbtest</td>
<td>Query</td>
<td>1</td>
<td>wsrep: pre-commit/certification passed (2883)</td>
<td>COMMIT</td>
</tr>
</tbody>
</table>

8 rows in set (0.00 sec)
### PROCESSLIST: SELECT before ALTER

```sql
SELECT node> select DB, COMMAND, TIME, STATE, INFO from information_schema.processlist
   -> WHERE DB='sbtest';
```

<table>
<thead>
<tr>
<th>DB</th>
<th>COMMAND</th>
<th>TIME</th>
<th>STATE</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>sbtest</td>
<td>Query</td>
<td>0</td>
<td>statistics</td>
<td>SELECT pad FROM sбtest2 WHERE id=5009</td>
</tr>
<tr>
<td>sbtest</td>
<td>Query</td>
<td>0</td>
<td>starting</td>
<td>SELECT pad FROM sбtest3 WHERE id=4951</td>
</tr>
<tr>
<td>sbtest</td>
<td>Query</td>
<td>0</td>
<td>statistics</td>
<td>SELECT pad FROM sбtest4 WHERE id=4954</td>
</tr>
<tr>
<td>sbtest</td>
<td>Query</td>
<td>0</td>
<td>System lock</td>
<td>SELECT pad FROM sбtest2 WHERE id=5351</td>
</tr>
<tr>
<td>sbtest</td>
<td>Query</td>
<td>0</td>
<td>cleaning up</td>
<td>SELECT pad FROM sбtest2 WHERE id=4954</td>
</tr>
<tr>
<td>sbtest</td>
<td>Sleep</td>
<td>0</td>
<td></td>
<td>NULL</td>
</tr>
<tr>
<td>sbtest</td>
<td>Query</td>
<td>0</td>
<td>Sending to client</td>
<td>SELECT pad FROM sбtest1 WHERE id=4272</td>
</tr>
<tr>
<td>sbtest</td>
<td>Query</td>
<td>0</td>
<td>closing tables</td>
<td>SELECT pad FROM sбtest4 WHERE id=4722</td>
</tr>
</tbody>
</table>

8 rows in set (0.00 sec)
DDL node> use ddltest;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

DDL node> alter table sbtest1 add key k1(c, k, pad);
Query OK, 0 rows affected (3 min 53.73 sec)
Records: 0 Duplicates: 0 Warnings: 0
### PROCESSLIST: DML during ALTER

```sql
DML node> select DB, COMMAND, TIME, STATE, INFO from information_schema.processlist -> WHERE DB in ('sbtest','ddltest');
```

<table>
<thead>
<tr>
<th>DB</th>
<th>COMMAND</th>
<th>TIME</th>
<th>STATE</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>sbtest</td>
<td>Query</td>
<td>36</td>
<td>wsrep: initiating pre-commit for write set (7886)</td>
<td>COMMIT</td>
</tr>
<tr>
<td>sbtest</td>
<td>Query</td>
<td>37</td>
<td>wsrep: initiating pre-commit for write set (7882)</td>
<td>COMMIT</td>
</tr>
<tr>
<td>sbtest</td>
<td>Query</td>
<td>27</td>
<td>wsrep: initiating pre-commit for write set (7887)</td>
<td>COMMIT</td>
</tr>
<tr>
<td>sbtest</td>
<td>Query</td>
<td>27</td>
<td>wsrep: initiating pre-commit for write set (7888)</td>
<td>COMMIT</td>
</tr>
<tr>
<td>sbtest</td>
<td>Query</td>
<td>36</td>
<td>wsrep: initiating pre-commit for write set (7885)</td>
<td>COMMIT</td>
</tr>
<tr>
<td>sbtest</td>
<td>Query</td>
<td>37</td>
<td>wsrep: initiating pre-commit for write set (7883)</td>
<td>COMMIT</td>
</tr>
<tr>
<td>sbtest</td>
<td>Query</td>
<td>37</td>
<td>wsrep: initiating pre-commit for write set (7884)</td>
<td>COMMIT</td>
</tr>
<tr>
<td>sbtest</td>
<td>Query</td>
<td>10</td>
<td>wsrep: initiating pre-commit for write set (7889)</td>
<td>COMMIT</td>
</tr>
<tr>
<td>ddltest</td>
<td>Sleep</td>
<td>38</td>
<td>altering table</td>
<td>alter table sbtest1.</td>
</tr>
</tbody>
</table>

9 rows in set (0.00 sec)
### PROCESSLIST: SELECT during ALTER

```
SELECT node> select DB, COMMAND, TIME, STATE, INFO from information_schema.processlist
    -> WHERE DB in ('sbtest', 'ddltest');
```

<table>
<thead>
<tr>
<th>DB</th>
<th>COMMAND</th>
<th>TIME</th>
<th>STATE</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>sbtest</td>
<td>Sleep</td>
<td>0</td>
<td></td>
<td>NULL</td>
</tr>
<tr>
<td>sbtest</td>
<td>Sleep</td>
<td>0</td>
<td></td>
<td>NULL</td>
</tr>
<tr>
<td>sbtest</td>
<td>Query</td>
<td>0</td>
<td>Sending to client</td>
<td>SELECT pad FROM sbtest4 WHERE id=4989</td>
</tr>
<tr>
<td>sbtest</td>
<td>Sleep</td>
<td>0</td>
<td></td>
<td>NULL</td>
</tr>
<tr>
<td>sbtest</td>
<td>Query</td>
<td>0</td>
<td>query end</td>
<td>SELECT pad FROM sbtest2 WHERE id=4961</td>
</tr>
<tr>
<td>sbtest</td>
<td>Sleep</td>
<td>0</td>
<td></td>
<td>NULL</td>
</tr>
<tr>
<td>sbtest</td>
<td>Sleep</td>
<td>0</td>
<td></td>
<td>NULL</td>
</tr>
<tr>
<td>sbtest</td>
<td>Sleep</td>
<td>0</td>
<td></td>
<td>NULL</td>
</tr>
<tr>
<td>ddltest</td>
<td>Sleep</td>
<td>39</td>
<td>altering table</td>
<td>alter table sbtest1 add key k1(c, k, pad)</td>
</tr>
</tbody>
</table>

9 rows in set (0.14 sec)
TOI Advantages

- Data always consistent
- DDL applied to all nodes at the same time
- No failure due to schema inconsistency
TOI Disadvantages

• The whole cluster blocked
  • For the duration of the entire DDL operation
• Schema upgrades replicated as a statement
  • There is no guarantee that the ALTER succeed!
How to Perform Upgrade with TOI

- Schedule maintenance window
- Run DDL
- Cluster won’t be accessible until DDL finishes
  - SELECTs can continue
  - `wsrep_sync_wait != 1`
When to Use TOI

• Quick DDL operations
When to Use TOI

- Quick DDL operations
- Creating new database objects
  - CREATE DATABASE
  - CREATE TABLE
When to Use TOI

• Quick DDL operations
• Creating new database objects
• Online operations which modify metadata only
  • RENAME INDEX
  • RENAME TABLE
  • DROP INDEX
  • ALGORITHM=INSTANT

Full list
Rolling Schema Upgrade (RSU)

- Variable `wsrep_osu_method`
- Puts node into de-sync state
  - For the duration of DDL
- Pauses Galera provider
- Schema can get out of sync!
User Responsibility

• Run DDL on the each node of the cluster
• Block read-write access that depend on DDL
  • Until all nodes are in sync
• Make sure no write is performed to the table
  • Until upgrade finishes on all nodes
• Failure makes cluster unrecoverable!
RSU Workflow

- **User Action**
  - SET SESSION wsrep_OSU_method = 'RSU';
  - DDL
  - Any other statement

- **Node Operation**
  - Nothing
  - Is wsrep_OSU_method set to RSU?
    - **Yes** Performs DDL
    - **Nothing**
How Node Internally Executes DDL in RSU Mode?

Does node have transactions in COMMIT mode?
How Node Internally Executes DDL in RSU Mode?

▼ Does node have transactions in COMMIT mode?
Yes Wait for 5 milliseconds
How Node Internally Executes DDL in RSU Mode?

▼ Does node have transactions in COMMIT mode?

Yes  Wait for 5 milliseconds
▼ Still transactions in the COMMIT mode exist?
How Node Internally Executes DDL in RSU Mode?

▼ Does node have transactions in COMMIT mode?
Yes  Wait for 5 milliseconds

▼ Still transactions in the COMMIT mode exist?
Yes  Abort DDL
How Node Internally Executes DDL in RSU Mode?

▼ Does node have transactions in COMMIT mode?
No  Put node into de-sync state
How Node Internally Executes DDL in RSU Mode?

▼ Does node have transactions in COMMIT mode?

No  Put node into de-sync state
▼ Pause write-set application
How Node Internally Executes DDL in RSU Mode?

- Does node have transactions in COMMIT mode?
- No Put node into de-sync state
- Pause write-set application
- Execute DDL
How Node Internally Executes DDL in RSU Mode?

- Does node have transactions in COMMIT mode?
  - No: Put node into de-sync state
  - Yes: Pause write-set application
  - Execute DDL
  - Bring the node back to the cluster
How Node Internally Executes DDL in RSU Mode?

- Does node have transactions in COMMIT mode?
  - No  Put node into de-sync state
  - Pause write-set application
  - Execute DDL
  - Bring the node back to the cluster
  - Synchronize
RSU: Locking

- Not avoidable
- Updates to all objects on the node in RSU mode must finish before the operation
- Failure aborts DDL
RSU Advantages

- Cluster remains functional
- Schedule long-running ALTER
  - In the best time possible
RSU Disadvantages

• No checks for data and schema consistency
  • This is your responsibility!
RSU Disadvantages

- No checks for data and schema consistency
- All writes must be stopped on the affected node
  - Otherwise DDL fails with an error
RSU Disadvantages

- No checks for data and schema consistency
- **All** writes must be stopped on the affected node
- `gcache` should be big enough to hold changes
  - Made while DDL was running
  - Failure will cause SST when node re-joins cluster
  - **All** schema changes will be lost

Any error can make cluster dysfunctional

Affected table must be offline

Until the schema upgrade is done on all nodes

Unless this is schema-compatible change
RSU Disadvantages

• No checks for data and schema consistency
• All writes must be stopped on the affected node
• gcache should be big enough to hold changes
• Any error can make cluster dysfunctional
RSU Disadvantages

- No checks for data and schema consistency
- **All** writes must be stopped on the affected node
- gcache should be big enough to hold changes
- Any error can make cluster dysfunctional
- Affected table must be offline
  - Until the schema upgrade is done on all nodes
  - Unless this is schema-compatible change
How to Use RSU

• Make sure gcache is big enough
  • Must hold all updates while DDL is in progress
How to Use RSU

- Make sure gcache is big enough
  - Must hold all updates while DDL is in progress
- Block all writes to the table/schema
How to Use RSU

Choose an "upgrading node"
How to Use RSU

- Choose an "upgrading node"
- Block all write requests to this node
How to Use RSU

- Choose an "upgrading node"
- Block all write requests to this node
- SET SESSION wsrep_OSU_method = 'RSU';
How to Use RSU

- Choose an “upgrading node”
- Block all write requests to this node
- `SET SESSION wsrep_OSU_method = 'RSU';`
- Perform DDL in the same session
How to Use RSU

1. Choose an "upgrading node"
2. Block all write requests to this node
3. Perform DDL in the same session
4. Re-enable writes

Repeat for other nodes

SET SESSION wsrep_OSU_method = 'RSU';

SET SESSION wsrep_OSU_method = 'TOI';
How to Use RSU

- Choose an "upgrading node"
- Block all write requests to this node
- SET SESSION wsrep_OSU.method = 'RSU';
- Perform DDL in the same session
- SET SESSION wsrep_OSU.method = 'TOI';
- Re-enable writes
How to Use RSU

- Choose an "upgrading node"
- Block all write requests to this node
- SET SESSION wsrep_OSU_method = 'RSU';
- Perform DDL in the same session
- SET SESSION wsrep_OSU_method = 'TOI';
- Re-enable writes
- Repeat for other nodes
pt-online-schema-change (pt-osc)
pt-online-schema-change (pt-osc)

• A tool, performing non-blocking upgrades
  • With TOI
pt-online-schema-change (pt-osc)

• A tool, performing non-blocking upgrades
• Creates a copy of table with altered definition
pt-online-schema-change (pt-osc)

• A tool, performing non-blocking upgrades
• Creates a copy of table with altered definition
• Creates triggers which will copy modified rows
pt-online-schema-change (pt-osc)

- A tool, performing non-blocking upgrades
- Creates a copy of table with altered definition
- Creates triggers which will copy modified rows
- Starts copying data in chunks
  - Absolutely under control
  - Can be paused or stopped
  - --max-flow-ctl
pt-online-schema-change (pt-osc)

- A tool, performing non-blocking upgrades
- Creates a copy of table with altered definition
- Creates triggers which will copy modified rows
- Starts copying data in chunks
  - All rows already in the table are copied in chunks
  - Newly modified rows are copied using triggers
pt-online-schema-change (pt-osc)

- A tool, performing non-blocking upgrades
- Creates a copy of table with altered definition
- Creates triggers which will copy modified rows
- Starts copying data in chunks
- Once copy is complete, drops the table
pt-online-schema-change (pt-osc)

- A tool, performing non-blocking upgrades
- Creates a copy of table with altered definition
- Creates triggers which will copy modified rows
- Starts copying data in chunks
- Once copy is complete, drops the table
- Renames the copy into the original table name
pt-osc Advantages

• DDL is safe and non-blocking
pt-osc Disadvantages

• Works only with InnoDB tables
• Increases IO load even for inplace operations
• Conflicts with already existing triggers
  • Unless you use MariaDB \( \geq 10.2.3 \)
• Foreign keys updates are not effectively safe
How to Use pt-osc

• Study pt-osc options
  • \texttt{--max-flow-ctl}
• Set appropriate limits
• Make sure \texttt{wsrep\_OSU\_method} is TOI
• Run pt-osc
Which Method to Use?

▼ Will DDL be fast?

• CREATE DATABASE
• CREATE TABLE
• DROP INDEX
• Any ALTER on small tables
• Other
Which Method to Use?

▼ Will DDL be fast?
Yes Use TOI
Which Method to Use?

▼ Will DDL be fast?

Yes Use TOI

No Evaluate if you can use pt-osc

- Operation on the InnoDB table
- Table has no triggers or MariaDB \( \geq 10.2.3 \)
- Table is not referenced by a foreign key
- You can tolerate increased IO
Which Method to Use?

▶ Will DDL be fast?

Yes Use TOI
No Evaluate if you can use pt-osc
Yes Use pt-osc
Which Method to Use?

▼ Will DDL be fast?
Yes Use TOI
No Evaluate if you can use pt-osc

Yes Use pt-osc
No Use RSU

• Stop all write traffic on the node
• Stop all write traffic to the modified table
• Make sure to upgrade on all nodes
Conclusion

• Use TOI whenever possible
• Then use pt-osc
• RSU is a last resort
More information

Galera Cluster
MariaDB Galera Cluster
pt-online-schema-change
Thank you!

www.slideshare.net/SvetaSmirnova

twitter.com/svetsmirnova

github.com/svetasmirnova