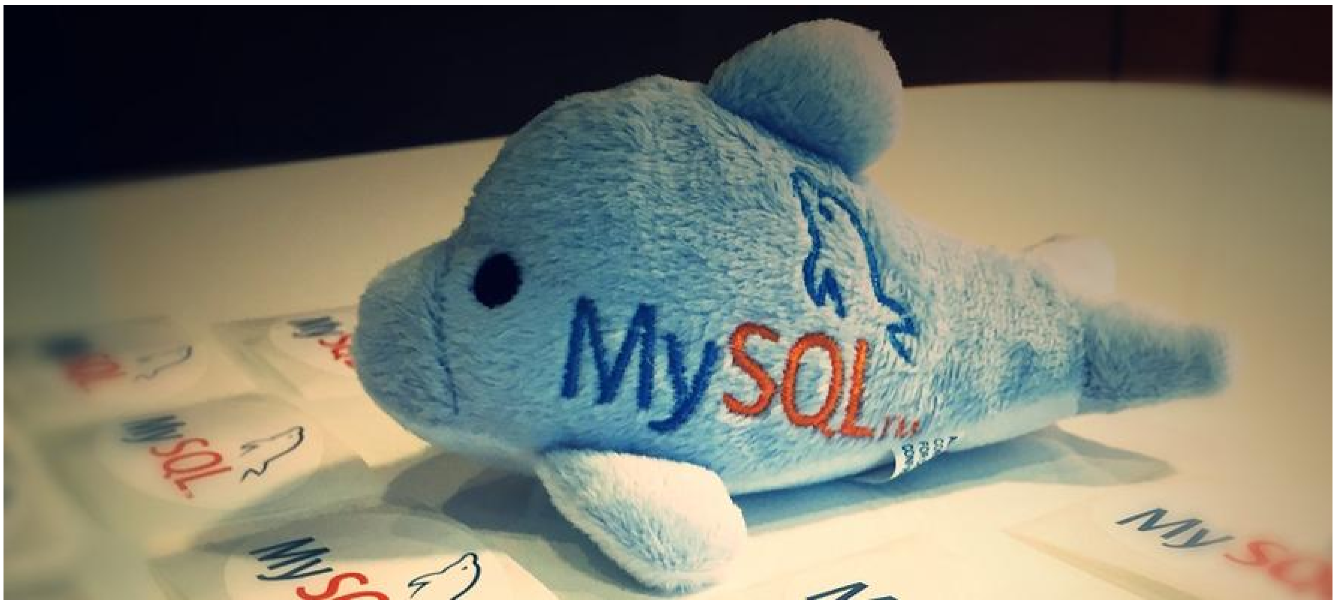
链滴

MySQL Group Replication, Single-Primary or Multi-Primary, how to make the right decision?

作者：zxniuniu
原文链接：https://ld246.com/article/1555303560220
来源网站：链滴
许可协议：署名-相同方式共享 4.0 国际 (CC BY-SA 4.0)

By default MySQL Group Replication runs in Single-Primary mode. And it’ s the best option nd the option you should use.

But sometimes it might happen that in very specific cases you would like to run you MGR Clus er in Multi-Primary mode: writing simultaneously on all the nodes member of the Group.



It’ s of course feasible but you need to make some extra verification as not all workload are ompatible with this behavior of the cluster.

Requirements

The requirements are the same as those for using MGR in Single-Primary mode:

- InnoDB Storage Engine
- Primary Keys
- IPv4 Network
- Binary Log Active
- Slave Updates Logged
- Binary Log Row Format
- Global Transaction Identifiers On
- Replication Information Repositories stored to tables
- Transaction Write Set Extraction set to XXHASH64

You can find more details in the online manual.

Limitations

These are the MySQL Group Replication Limitations as in the manual:

- Replication Event Checksums must be set to NONE
- Gap Locks, so better to use READ-COMMITTED as tx_isolation

原文链接：MySQL Group Replication, Single-Primary or Multi-Primary, how to make the right decision?

- Table Locks and Named Locks are not supported
- Savepoints are also not supported.
- SERIALIZABLE Isolation Level is not supported.
- Concurrent DDL vs DML/DDL Operations
- Foreign Keys with Cascading Constraints

So from the list above, the limitations that we will affect Multi-Primary mode are the Concurr nt DDLs/DML and the foreign keys.

Let’ s have a more detail look at them.

Concurrent DDL vs DML/DDL Operations

The manual says_Concurrent DDL vs DML/DDL operations on the same object, executing at di ferent servers, is not supported in multi-primary deployments. Conflicting data definition stat ments (DDL) executing on different servers are not detected. Concurrent data definition stat ments and data manipulation statements executing against the same object but on different ervers is not supported._

So this is clear. The only thing we can do, is then be sure we don’ t allow writes on the other odes when we need to run a DDL. This can be done in your router/proxy solution and/or set t e nodes in READ_ONLY.

This means that if your application performs DDL on it’ s own (and not handled by a DBA), I ould recommend you to not use Multi-Primary at all !

To verify if your application is running such statements, you can run the following query seve al times during the day and see how the values increase or not:

```
SELECT event_name, count_star, sum_errors
FROM events_statements_summary_global_by_event_name
WHERE event_name  REGEXP '.*sql/(create|drop|alter).*'
AND event_name NOT REGEXP '.*user';
```

Foreign Keys with Cascading Constraints

Again, let’ s see what the manual says about this limitation:Multi-primary mode groups do not ully support using foreign key constraints. Foreign key constraints that result in cascading operations executed by a multi-primary mode group have a risk of undetected conflicts. Therefore we recomm nd setting group_replication_enforce_update_everywhere_checks=ON on server instances used in ulti-primary mode groups. Disabling group_replication_enforce_update_everywhere_checks and usi g foreign keys with cascading constraints requires extra care. In single-primary mode this is not a pr blem.

So let’ s find if we have such design:

```
SELECT CONCAT(t1.table_name, '.', column_name) AS 'foreign key',
CONCAT(t1.referenced_table_name, '.', referenced_column_name) AS 'references',
t1.constraint_name AS 'constraint name', UPDATE_RULE, DELETE_RULE
FROM information_schema.key_column_usage as t1
JOIN information_schema.REFERENTIAL_CONSTRAINTS as t2
```

```
WHERE t2.CONSTRAINT_NAME = t1.constraint_name
AND t1.referenced_table_name IS NOT NULL
AND (DELETE_RULE = "CASCADE" OR UPDATE_RULE = "CASCADE");

+----------------------+--------------------+--------------------+------------+------------+
| foreign key          | references         | constraint name    | UPDATE_RULE | DELETE_RULE |
+----------------------+--------------------+--------------------+------------+------------+
| dept_emp.emp_no      | employees.emp_no   | dept_emp_ibfk_1    | RESTRICT   | CASCADE
|
| dept_emp.dept_no     | departments.dept_no | dept_emp_ibfk_2   | RESTRICT   | CASCADE
|
| dept_manager.emp_no  | employees.emp_no   | dept_manager_ibfk_1 | RESTRICT   | CASCA
E     |
| dept_manager.dept_no | departments.dept_no | dept_manager_ibfk_2 | RESTRICT   | CASCA
E     |
| salaries.emp_no      | employees.emp_no   | salaries_ibfk_1    | RESTRICT   | CASCADE    |
| titles.emp_no        | employees.emp_no   | titles_ibfk_1      | RESTRICT   | CASCADE    |
+----------------------+--------------------+--------------------+------------+------------+
```

So in our case above, we have a problem and it’ s not recommended to use multi-primary.

Let me show you what kind or error you may have.

Case 1: default settings + group_replication_single_primary_mode = o f

In that case, if we perform a DML on such table,… nothing happens ! No error as there is no c nflict on my test machine without concurrent workload.

But this is not safe as it might happen, remember, this is not fully supported !

Case 2: group_replication_single_primary_mode = off + group_replicat on_enforce_update_everywhere_checks = 1

Now if we run a DML on such table, we have an error:

```
mysql> update employees.salaries set salary = 60118 where emp_no=10002 and salary<6011
;
ERROR 3098 (HY000): The table does not comply with the requirements by an external plugin.
```

and in the error log we can read:

```
[ERROR] Plugin group_replication reported: 'Table salaries has a foreign key with 'CASCADE' c
ause. This is not compatible with Group Replication'
```

So be careful that by default you could get some issues as the check is disabled.

I also want to add that all the nodes in the Group must have that setting. I you try to start gro p replication on a node where you have a different value for

group_replication_enforce_update_everywhere_checks, then the node won’ t be able to join nd in the error log you will see:

```
[ERROR] Plugin group_replication reported: 'The member configuration is not compatible with
```

the group configuration. Variables such as single_primary_mode or enforce_update_everywhe e_checks must have the same value on every server in the group.

 (member configuration option: [group_replication_enforce_update_everywhere_checks], gro p configuration option: []).

Is this enough to be sure that our cluster will run smoothly in Multi-Primary mode ? In fact no t isn’ t !

Now we also try to reduce the risk of certification failure that might happen when writing on multiple nodes simultaneously.

Workload Check

Group Replication might be sensible when writing on multiple nodes (Multi-Primary mode) d the following workload:

- Large transactions (they have the risk to be in conflict with a short one that they have to rol back too frequently)
- Hotspots: rows that might be changed on all the nodes simultaneously

Large Transactions

With Performance_Schema, we have in MySQL everything we need to be able to identify large transactions. We will focus then on identifying :

- the transactions with most statements (and most writes in particular)
- the transactions with most rows affected
- the largest statements by row affected

Before being able to verify all this on your current system that you want to migrate to Group eplication, we need to activate some consumers and instruments in Performance_Schema:

```
mysql> update performance_schema.setup_consumers
set enabled = 'yes'
where name like 'events_statement%' or name like 'events_transaction%';

mysql> update performance_schema.setup_instruments
set enabled = 'yes', timed = 'yes'
where name = 'transaction';
```

Now we should let the system run for some time and verify when we have enough data collec ed.

In the future some of the data we are collecting in this article will be available viasys.

Transactions with most statements

```
select t.thread_id, t.event_id, count(*) statement_count,
sum(s.rows_affected) rows_affected,
length(replace(group_concat(
case when s.event_name = "statement/sql/update" then 1
when s.event_name = "statement/sql/insert" then 1
when s.event_name = "statement/sql/delete" then 1
else null end),',','')) as "# write statements"
from performance_schema.events_transactions_history_long t
join performance_schema.events_statements_history_long s
on t.thread_id = s.thread_id and t.event_id = s.nesting_event_id
group by t.thread_id, t.event_id order by rows_affected desc limit 10;
```

We can also see those statements has I illustrate it below:

```
mysql> set group_concat_max_len = 1000000;
mysql> select t.thread_id, t.event_id, count(*) statement_count,
    ->        sum(s.rows_affected) rows_affected,
    ->        group_concat(sql_text order by s.event_id separator '\n') statements
    -> from performance_schema.events_transactions_history_long t
    -> join performance_schema.events_statements_history_long s
    ->   on t.thread_id = s.thread_id and t.event_id = s.nesting_event_id
    -> group by t.thread_id, t.event_id order by statement_count desc limit 1\G
*************************** 1. row ***************************
      thread_id: 332
       event_id: 20079
statement_count: 19
  rows_affected: 4
     statements: SELECT c FROM sbtest1 WHERE id=5011
SELECT c FROM sbtest1 WHERE id=4994
SELECT c FROM sbtest1 WHERE id=5049
SELECT c FROM sbtest1 WHERE id=5048
SELECT c FROM sbtest1 WHERE id=4969
SELECT c FROM sbtest1 WHERE id=4207
SELECT c FROM sbtest1 WHERE id=4813
SELECT c FROM sbtest1 WHERE id=4980
SELECT c FROM sbtest1 WHERE id=4965
SELECT c FROM sbtest1 WHERE id=5160
SELECT c FROM sbtest1 WHERE id BETWEEN 4965 AND 4965+99
SELECT SUM(K) FROM sbtest1 WHERE id BETWEEN 3903 AND 3903+99
SELECT c FROM sbtest1 WHERE id BETWEEN 5026 AND 5026+99 ORDER BY c
SELECT DISTINCT c FROM sbtest1 WHERE id BETWEEN 5015 AND 5015+99 ORDER BY c
UPDATE sbtest1 SET k=k+1 WHERE id=5038
UPDATE sbtest1 SET c='09521266577-73910905313-02504464680-26379112033-242685503
4-82474773859-79238765464-79164299430-72120102543-79625697876' WHERE id=4979
DELETE FROM sbtest1 WHERE id=4964
INSERT INTO sbtest1 (id, k, c, pad) VALUES (4964, 5013, '92941108506-80809269412-934669
1769-85515755897-68489598719-07756610896-31666993640-93238959707-66480092830-9
721213568', '74640142294-85723339839-62552309335-30960818723-80741740383')
COMMIT
```

Of course there is not rules of thumb saying what is a transaction with too much statements, his is your role as DBA to analyze this and see how often such transaction could enter in confl ct on multiple nodes at the same time.

You can also see the amount of conflicts per host using the following statement:

```
mysql> select COUNT_CONFLICTS_DETECTED from performance_schema.replication_group_
ember_stats;
```

原文链接：MySQL Group Replication, Single-Primary or Multi-Primary, how to make the right decision?

```
+--------------------------+
| COUNT_CONFLICTS_DETECTED |
+--------------------------+
|                        4 |
+--------------------------+
```

Transactions with most rows affected

This is of course a more important value to get than the previous one and here we will directly now how many rows could enter in conflict:

```
select t.thread_id, t.event_id, count(*) statement_count,
       sum(s.rows_affected) rows_affected,
       length(replace(group_concat(
       case
         when s.event_name = "statement/sql/update" then 1
         when s.event_name = "statement/sql/insert" then 1
         when s.event_name = "statement/sql/delete" then 1
         else null end),',','')) as "# write statements"
from performance_schema.events_transactions_history_long t
join performance_schema.events_statements_history_long s
  on t.thread_id = s.thread_id and t.event_id = s.nesting_event_id
group by t.thread_id, t.event_id order by rows_affected desc limit 10;
```

If you find some with a large amount of rows, you can again see what were the statements in hat particular transaction. This is an example:

```
select t.thread_id, t.event_id, count(*) statement_count,
       sum(s.rows_affected) rows_affected,
       group_concat(sql_text order by s.event_id separator '\n') statements
from performance_schema.events_transactions_history_long t
join performance_schema.events_statements_history_long s
  on t.thread_id = s.thread_id and t.event_id = s.nesting_event_id
group by t.thread_id, t.event_id order by rows_affected desc limit 1\G
```

Just**don’t forget to verify the auto_commit ones as they are not returned**with the query bove.

Largest statements by row affected

This query can be used to find the specific statement that modifies the most rows:

```
SELECT query, db, rows_affected, rows_affected_avg
FROM sys.statement_analysis
ORDER BY rows_affected_avg DESC, rows_affected DESC LIMIT 10;
```

Hotspots

For hotspots, we will look for the queries updating most the same PK and therefore having to wait more.

```
SELECT *
FROM performance_schema.events_statements_history_long
```

```
WHERE rows_affected > 1 ORDER BY timer_wait DESC LIMIT 20
```

Conclusion

As you can see, the workload is also important when you decide to spread your writes to all n des or use only a dedicated one. The default is safer and requires less analysis.

Therefore, I recommend to use MySQL Group Replication in Multi-Primary Mode only to adv nced users

感谢原作者，博客转自：https://lefred.be/content/mysql-group-replication-single-primary-or-ulti-primary