MariaDB 10.5 New Features for Troubleshooting

Valerii Kravchuk, Principal Support Engineer, MariaDB

valerii.kravchuk@mariadb.com

Who am I and What Do I Do?

Valerii (aka **Valeriy**) **Kravchuk**:

- MySQL Support Engineer in MySQL AB, Sun and Oracle, 2005-2012
- Principal Support Engineer in Percona, 2012-2016
- Principal Support Engineer in MariaDB Corporation since March 2016
- **http://mysqlentomologist.blogspot.com** - my blog about MariaDB and MySQL (including some **HowTo**s, not only bugs marketing)
- **https://www.facebook.com/valerii.kravchuk** - my Facebook page
- **http://bugs.mysql.com** - my personal playground
- **@mysqlbugs #bugoftheday**
- **MySQL Community Contributor of the Year 2019**
- I speak about MySQL and MariaDB in public. Some slides from previous talks are here and there…
- "I solve problems", "I drink and I know things"

Disclaimers

- Since September, 2012 I act as an Independent Consultant providing services to different companies
- All views, ideas, conclusions, statements and approaches in my presentations and blog posts are mine and may not be shared by any of my previous, current and future employees, customers and partners
- All examples are either based on public information or are truly fictional and has nothing to do with any real persons or companies. Any similarities are pure coincidence :)
- The information presented is true to the best of my knowledge

What is this session about?

- MariaDB 10.5 new features that may help DBAs and application developers to find out what's going on when a problem occurs:
  - Performance Schema updates to match MySQL 5.7 instrumentation (and add some more)
  - New tables in the INFORMATION_SCHEMA to monitor the internals of a generic thread pool and few new server variables
  - Improvements of ANALYZE for statements
- Some related examples, blog posts and discussions

Performance Schema: 10.4 vs MySQL 5.7 vs 10.5

```
MySQL [information_schema]> select version(), count(*) from tables where
table_schema='performance_schema';
+-----------+----------+
| version() | count(*) |
+-----------+----------+
| 5.7.30    |       87 | -- was 52 in 10.4
+-----------+----------+
1 row in set (0,001 sec)

MySQL [information_schema]> select version(), count(*) from
performance_schema.global_variables where variable_name like 'performance%';
+-----------+----------+
| version() | count(*) |
+-----------+----------+
| 5.7.30    |       42 | -- was 32 in 10.4, 42 in 10.5
+-----------+----------+
1 row in set (0,002 sec)

MariaDB [(none)]> select version(), count(*) from information_schema.tables where
table_schema='performance_schema';
+----------------+----------+
| version()      | count(*) |
+----------------+----------+
| 10.5.6-MariaDB |       80 |
+----------------+----------+
1 row in set (0,060 sec)
```

What's new in Performance Schema?

- Memory (MDEV-16431)
- Metadata locking (MDL) (MDEV-16432)
- Prepared statements (MDEV-16433)
- [show] status instrumentation and tables (MDEV-16438)
- Stored procedures (MDEV-16434)
- SX-locks (MDEV-16436)
- Transactions (MDEV-16435)
- User variables (MDEV-16439)
- Replication-related tables
- Now some memory for P_S is allocated dynamically

P_S Memory Instrumentation: Instruments

- 270 additional instruments (not properly documented, see MDEV-23436 and this blog post):

```
MariaDB [performance_schema]> select name from
performance_schema.setup_instruments where name like 'memory%';
+-----------------------------------------------------------------+
| name                                                            |
+-----------------------------------------------------------------+
| memory/performance_schema/mutex_instances                       |
| memory/performance_schema/rwlock_instances                      |
| memory/performance_schema/cond_instances                        |
| memory/performance_schema/file_instances                        |
...
| memory/sql/udf_mem                                              |
+-----------------------------------------------------------------+
270 rows in set (0,001 sec)
```

P_S Memory Instrumentation: Summary Tables

- 5 summary tables
- KB does not help much with them, so I add some hints here:

```
MariaDB [performance_schema]> show tables like '%memory%';
+----------------------------------------+
| Tables_in_performance_schema (%memory%) |
+----------------------------------------+
| memory_summary_by_account_by_event_name | -- user, host
| memory_summary_by_host_by_event_name    | -- host char(60)
| memory_summary_by_thread_by_event_name  | -- threads.thread_id
| memory_summary_by_user_by_event_name    | -- user char(32)
| memory_summary_global_by_event_name     |
+----------------------------------------+
5 rows in set (0,019 sec)
```

P_S Memory Instrumentation: Tables Structure

- Common columns (see also MySQL 5.7 manual):

```
MariaDB [performance_schema]> desc
memory_summary_global_by_event_name;
...
| EVENT_NAME                  | varchar(128)         ...
| COUNT_ALLOC                 | bigint(20) unsigned ...
| COUNT_FREE                  | bigint(20) unsigned ...
| SUM_NUMBER_OF_BYTES_ALLOC   | bigint(20) unsigned ...
| SUM_NUMBER_OF_BYTES_FREE    | bigint(20) unsigned ...
| LOW_COUNT_USED              | bigint(20)          ...
| CURRENT_COUNT_USED          | bigint(20)          ...
| HIGH_COUNT_USED             | bigint(20)          ...
| LOW_NUMBER_OF_BYTES_USED    | bigint(20)          ...
| CURRENT_NUMBER_OF_BYTES_USED | bigint(20)         ...
| HIGH_NUMBER_OF_BYTES_USED   | bigint(20)          ...
```

P_S Memory Instrumentation: Example

- Let's see what memory was allocated most often for:

```
MariaDB [performance_schema]> select * from
memory_summary_global_by_event_name order by count_alloc desc
limit 1\G
*************************** 1. row ***************************
                 EVENT_NAME: memory/sql/QUICK_RANGE_SELECT::alloc
               COUNT_ALLOC: 147976
                COUNT_FREE: 147976
SUM_NUMBER_OF_BYTES_ALLOC: 600190656
 SUM_NUMBER_OF_BYTES_FREE: 600190656
           LOW_COUNT_USED: 0
       CURRENT_COUNT_USED: 0
          HIGH_COUNT_USED: 68
  LOW_NUMBER_OF_BYTES_USED: 0
CURRENT_NUMBER_OF_BYTES_USED: 0
HIGH_NUMBER_OF_BYTES_USED: 275808
1 row in set (0,069 sec)
```

P_S Memory Instrumentation in MariaDB 10.5

- The implementation is different vs MySQL (MDEV-22841)
- Memory for **performance_schema** may now be allocated dynamically after startup:

  "The Performance Schema dynamically allocates memory incrementally, scaling its memory use to actual server load, instead of allocating required memory during server startup. Once memory is allocated, it is not freed until the server is restarted."
- We can see it from **performance_schema** (demo):

```
openxs@ao756:~/dbs/maria10.5$ bin/mysql
--socket=/tmp/mariadb105.sock -e"select
sum(SUM_NUMBER_OF_BYTES_ALLOC) alloc,
sum(SUM_NUMBER_OF_BYTES_FREE) free,
sum(CURRENT_NUMBER_OF_BYTES_USED) used from
performance_schema.memory_summary_global_by_event_name where
event_name like 'memory/performance%'"
```

Performance Schema: MDL Instrumentation

- There are different ways to study metadata locks…
- In MariaDB 10.5 we can now use performance_schema:

```
MariaDB [performance_schema]> show tables like '%metadata%';
+-----------------------------------------+
| Tables_in_performance_schema (%metadata%) |
+-----------------------------------------+
| metadata_locks                          |
+-----------------------------------------+
1 row in set (0,001 sec)
MariaDB [performance_schema]> select * from setup_instruments where
name like 'wait/lock/metadata%';
+----------------------------+---------+-------+
| NAME                       | ENABLED | TIMED |
+----------------------------+---------+-------+
| wait/lock/metadata/sql/mdl | NO      | NO    |
+----------------------------+---------+-------+
1 row in set (0,001 sec)
```

MDL Instrumentation: Basic Usage

- Enable:

```
MariaDB [performance_schema]> update setup_instruments set
enabled='YES', timed='YES' where name like 'wait/lock/metadata%';
Query OK, 1 row affected (0,016 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

- Check:

```
MariaDB [performance_schema]> select * from metadata_locks\G
*************************** 1. row ***************************
          OBJECT_TYPE: TABLE
        OBJECT_SCHEMA: performance_schema
          OBJECT_NAME: metadata_locks
OBJECT_INSTANCE_BEGIN: 139893728670576
            LOCK_TYPE: SHARED_READ
        LOCK_DURATION: TRANSACTION
          LOCK_STATUS: GRANTED
               SOURCE:
     OWNER_THREAD_ID: 129 -- join to p_s.threads.thread_id
      OWNER_EVENT_ID: 1
```

Performance Schema: PS Instrumentation

- *Active* prepared statements are instrumented by default:

```
MariaDB [performance_schema]> show tables like '%prepare%';
+-----------------------------------------+
| Tables_in_performance_schema (%prepare%) |
+-----------------------------------------+
| prepared_statements_instances           |
+-----------------------------------------+
1 row in set (0,001 sec)
MariaDB [performance_schema]> select * from setup_instruments where name
like 'statement/%/prepare%' or name like 'statement/%/execute%';
+---------------------------------+---------+-------+
| NAME                            | ENABLED | TIMED |
+---------------------------------+---------+-------+
| statement/sql/prepare_sql       | YES     | YES   | -
mysql_stmt_prepare()
| statement/sql/execute_sql       | YES     | YES   | -
mysql_stmt_execute()
| statement/sql/execute_immediate | YES     | YES   |
| statement/com/Prepare           | YES     | YES   | - PREPARE
| statement/com/Execute           | YES     | YES   | - EXECUTE
+---------------------------------+---------+-------+
5 rows in set (0,001 sec)
```

Prepared Statements Instrumentation: Example

- Let's run some **sysbench** test and check (demo):

```
MariaDB [(none)]> select count(*) from prepared_statements_instances;
+-----------+
| count(*) |
+-----------+
|   204    |
+-----------+
1 row in set (0,001 sec)

MariaDB [(none)]> select * from
performance_schema.prepared_statements_instances limit 1\G
*************************** 1. row ***************************
    OBJECT_INSTANCE_BEGIN: 139894074271256
             STATEMENT_ID: 18
             STATEMENT_NAME: NULL
                 SQL_TEXT: COMMIT
         OWNER_THREAD_ID: 234
             OWNER_EVENT_ID: 3
         OWNER_OBJECT_TYPE: NULL
...
```

Performance Schema: Status Variables

- Status variables are instrumented more ar less like in MySQL 5.7. Let's quickly check a demo...
- But there are 3 more summary tables it seems:

```
MariaDB [performance_schema]> show tables like '%status%';
+-----------------------------------------+
| Tables_in_performance_schema (%status%) |
+-----------------------------------------+
| global_status                           |
...
| session_status                          |
| status_by_account                       | -- user, host
| status_by_host                          | -- host
| status_by_thread                        | -- threads.thread_id
| status_by_user                          | -- user
+-----------------------------------------+
8 rows in set (0,001 sec)
```

Performance Schema: Stored Procedures Instrumentation

- Along the lines of MySQL WL#5766
- New instrumentable object types added:

```
MariaDB [performance_schema]> select distinct object_type from
setup_objects;
+-------------+
| object_type |
+-------------+
| EVENT       |
| FUNCTION    |
| PROCEDURE   |
| TABLE       |
| TRIGGER     |
+-------------+
5 rows in set (0,023 sec)
```

- Enabled/timed by default in non-system databases

P_S Stored Procedures Instrumentation: Details

- 20 related instruments added:

```
MariaDB [performance_schema]> select * from setup_instruments where
name like 'statement/sp/%' or name like 'statement/scheduler%';
+--------------------------------+---------+-------+
| NAME                           | ENABLED | TIMED |
+--------------------------------+---------+-------+
| statement/sp/stmt              | YES     | YES   |
| statement/sp/set               | YES     | YES   |
| statement/sp/set_trigger_field | YES     | YES   |
| statement/sp/jump              | YES     | YES   |
| statement/sp/jump_if_not       | YES     | YES   |
| statement/sp/freturn           | YES     | YES   |
...
| statement/sp/set_case_expr     | YES     | YES   |
| statement/scheduler/event      | YES     | YES   |
+--------------------------------+---------+-------+
20 rows in set (0,002 sec)
```

P_S Stored Procedures Instrumentation: Details

- New **events_statements_summary_by_program** table added
- KB just lists columns without much details
- Some additional columns with statistics about nested statements invoked during stored program execution:

```
...
| COUNT_STATEMENTS          | bigint(20) unsigned ...
| SUM_STATEMENTS_WAIT       | bigint(20) unsigned ...
| MIN_STATEMENTS_WAIT       | bigint(20) unsigned ...
| AVG_STATEMENTS_WAIT       | bigint(20) unsigned ...
| MAX_STATEMENTS_WAIT       | bigint(20) unsigned …
...
```

- Let’s run a quick demo...

Performance Schema: SX-locks Instrumentation

- See MySQL WL#7445 - "**PERFORMANCE SCHEMA: instrument SX-lock for rw_lock**"
- performance schema instrumentation for read/write locks is enhanced to support the new SX-lock operation
- New **wait/synch/sxlock/%** instruments
- The list of operations supported by the performance schema read-write lock
- Instrumentation is extended to include the following operations:
  - SHARED LOCK
  - SHARED EXCLUSIVE LOCK
  - EXCLUSIVE LOCK
  - TRY SHARED LOCK
  - TRY SHARED EXCLUSIVE LOCK
  - TRY EXCLUSIVE LOCK
- Let's consider an example (demo) of MariaDB 10.4 vs 10.5 and new instruments

Performance Schema: Transactions Instrumentation

- Transactions are now instrumented similarly to MySQL 5.7 (see MySQL manual and compare to MariaDB KB).
- **event_transactions_%** tables (**current**, **history**, **history_long**)
- "**transaction**" instrument in **setup_instruments**
- P_S events hierarchy is extended:

  transactions → statements → stages → waits
- Let's consider an example (demo) of getting the sequence of statements executed in frames of current transaction per thread

Performance Schema: User Variables Instrumentation

- It was really hard (but possible with **gdb**) to find the value of user variables in specific thread in the past...
- User variables instrumentation in P_S helps to make it trivial:

```
MariaDB [performance_schema]> desc user_variables_by_thread;
+----------------+---------------------+------+-----+---------+-------+
| Field          | Type                | Null | Key | Default | Extra |
+----------------+---------------------+------+-----+---------+-------+
| THREAD_ID      | bigint(20) unsigned | NO   |     | NULL    |       |
| VARIABLE_NAME  | varchar(64)         | NO   |     | NULL    |       |
| VARIABLE_VALUE | longblob            | YES  |     | NULL    |       |
+----------------+---------------------+------+-----+---------+-------+
3 rows in set (0,002 sec)
```

P_S User Variables Instrumentation: Example

- Let's check how to find user variables in the current thread:

```
MariaDB [performance_schema]> set @a := 10;
Query OK, 0 rows affected (0,000 sec)

MariaDB [performance_schema]> select uv.* from
user_variables_by_thread uv join threads t where t.thread_id =
uv.thread_id and t.processlist_id=connection_id()\G
*************************** 1. row ***************************
    THREAD_ID: 239
 VARIABLE_NAME: a
VARIABLE_VALUE: 10
1 row in set (0,001 sec
```

Performance Schema: Replication Instrumentation

- Related tables, subset of those in MySQL 5.7:

```
MariaDB [performance_schema]> show tables like 'replication%';
+-----------------------------------------+
| Tables_in_performance_schema (replication%) |
+-----------------------------------------+
| replication_applier_configuration       |
| replication_applier_status              |
| replication_applier_status_by_coordinator |
| replication_connection_configuration    |
+-----------------------------------------+
4 rows in set (0,001 sec)
```

- Probably work in progress still, see my MDEV-23590
- Only partially documented in the KB...
- Let’s try to do a quick test...

What's new in Thread Pool?

- MariaDB Thread Pool (since 5.5!) is cool!
- Information Schema tables (4) were added in 10.5 for internals of generic thread pool (MDEV-19313)
- **thread_pool_dedicated_listener** - the queueing time in the THREAD_POOL_QUEUES and the actual queue size in the THREAD_POOL_GROUPS table will be more exact, since IO requests are immediately dequeued from pool, without delay
- **thread_pool_exact_stats** - better queueing time statistics by using a high precision timestamp, at a small performance cost, for the time when the connection was added to the queue. This timestamp helps calculate the queuing time shown in the THREAD_POOL_QUEUES table.
- KB still misses details about the tables, columns, output examples...
- This commit is a useful reading
- Let's just check what we can see in these tables (demo)

What's new in ANALYZE?

- Execute the statement, and then produce EXPLAIN output instead of the result set, annotated with execution stats
- ANALYZE FORMAT=JSON for statements is improved, now it also shows the time spent checking the WHERE clause and doing other auxiliary operations (MDEV-20854)
- We now count the "gap" time between table accesses and display it as **r_other_time_ms** in the "table" element
- Table access time is reported as **r_table_time_ms** (former **r_total_time_ms**)
- Let's consider the example (demo)
- Compare to MySQL 8.0.18+ EXPLAIN ANALYZE

Summary

- **MariaDB 10.5** added a lot of useful and interesting features and improvements that may help for troubleshooting
- Documentation for many of them (P_S improvements specifically) is not yet completed. We have to rely on MySQL manual, tests and source code review when in doubts
- So, there is still a lot of work to do for Engineering, Documentation team, users and bloggers (like me)

Thank you!

Questions and Answers?

Please, search and report bugs at:

https://jira.mariadb.org