



ORACLE[®]

Introduction to InnoDB monitoring System

Jimmy Yang
Oracle Corp

Agenda

- Overview
- Current Monitoring System in InnoDB
- Performance Schema in InnoDB
- Metrics Counter System in InnoDB
- Summary

Overview

- Monitor System is a generic term that refers to system that collects and displays the system state information
 - System Monitoring is important for server resource configuration as well as server performance tuning
 - Monitoring system could target different audience, developers, DBAs or common users
 - Ideally it should be light weight with minimum performance impact
 - The system should have easy interfaces for further data process and presentation

Current Monitoring System in InnoDB

- InnoDB provide monitor information through following interfaces:
 - SHOW ENGINE INNODB STATUS (or create table innodb_monitor)
 - create specific innodb monitor tables for periodically output
 - innodb_lock_monitor for extensive lock information
 - innodb_tablespace_monitor provides list of file segments in the shared tablespace (Space Allocation)
 - innodb_table_monitor prints metadata information about InnoDB internal data dictionary (Metadata)
 - Status Variables

What's coming for Monitoring System in InnoDB

- More Monitoring: Support a variety of server status/state Monitoring through additional Monitoring System features
- Centralize: Centralize the monitor information interfaces, provide one-stop shop for server monitoring information. Move away from “create monitor table” interfaces.
- Extensible: Additional monitors can be added into different modules over the time, or on demand.
- Easy Query: Monitoring Data displayed through “fake” tables/views, and be able to query with SQL
- Both resource monitoring and performance monitoring are important to support.
- InnoDB Metrics Table(s) can achieve this
- For event monitoring, performance schema is a good choice

What is Performance Schema

- New feature in MySQL 5.5 (Peter Gultzan, Marc Alff)
- Monitors low-level server events (mutex, rwlock and IO etc.)
- Instrumentations give you both information about “Elapsed time” and “counter” for the events
- Expose as SQL tables for easy query, can be aggregated, averaged, ordered etc.

Performance Schema in InnoDB

- Performance Schema in InnoDB 1.1
 - 42 mutexes
 - 10 rwlocks
 - 7 types of threads
 - 3 types of I/O (data, log, tmpfile)
- What do you get from Performance Schema
 - Mutexes / rwlock usage statistics and current status
 - IO statistics
 - Active running threads

Performance Schema Instance table

- Find out instrumented mutexes / rwlocks through instance table:

```
mysql> SELECT DISTINCT(NAME)
-> FROM MUTEX_INSTANCES
-> WHERE NAME LIKE "%innodb%";
```

name
wait/synch/mutex/innodb/analyze_mutex
wait/synch/mutex/innodb/mutex_list_mutex
wait/synch/mutex/innodb/ibuf_mutex
.....
wait/synch/mutex/innodb/rseg_mutex
wait/synch/mutex/innodb/srv_monitor_file_mutex
wait/synch/mutex/innodb/buf_pool_mutex
wait/synch/mutex/innodb/mem_pool_mutex

34 rows in set (0.02 sec)

Performance Schema Instance Table

- There could be multiple instances of a mutex in the server

```
mysql> SELECT COUNT(*)  
      -> FROM MUTEX_INSTANCES  
      -> WHERE NAME LIKE "%rseg_mutex%";
```

```
+-----+  
| COUNT(*) |  
+-----+  
|   128   |  
+-----+
```

1 row in set (0.92 sec)

- Buffer block mutex/rwlock instrumentations are disabled by default.
 - One mutex/rwlock per 16k buffer block, server with large buffer pool configuration could easily create thousands of instances
 - Exceed the default value of max mutex/rwlock instances (1000) allowed
- If the mutex is not yet created, it will not be listed in the instance table, so you might see fewer events/instances than you might expected.

Performance Schema PROCESSLIST Table

- Check out running InnoDB Threads with PROCESSLIST table

```
mysql> SELECT * FROM PROCESSLIST WHERE name LIKE "%innodb%";
```

THREAD_ID	ID	NAME
1	0	thread/innodb/io_handler_thread
2	0	thread/innodb/io_handler_thread
3	0	thread/innodb/io_handler_thread
4	0	thread/innodb/io_handler_thread
5	0	thread/innodb/io_handler_thread
6	0	thread/innodb/io_handler_thread
8	0	thread/innodb/srv_lock_timeout_thread
9	0	thread/innodb/srv_error_monitor_thread
10	0	thread/innodb/srv_monitor_thread
11	0	thread/innodb/srv_master_thread

```
10 rows in set (0.02 sec)
```

Performance Schema WAITS Table

- Find out what is the last event happened to a thread with `EVENTS_WAITS_CURRENT` table

```
mysql> SELECT THREAD_ID, EVENT_NAME, SOURCE  
-> FROM EVENTS_WAITS_CURRENT  
-> WHERE EVENT_NAME LIKE "%innodb%";
```

THREAD_ID	EVENT_NAME	SOURCE
2	wait/synch/mutex/innodb/ios_mutex	srv0start.c:495
8	wait/synch/mutex/innodb/kernel_mutex	srv0srv.c:2182
9	wait/synch/mutex/innodb/log_sys_mutex	log0log.ic:405
11	wait/synch/mutex/innodb/thr_local_mutex	thr0loc.c:127

```
4 rows in set (0.00 sec)
```

Performance Schema HISTORY Table

- Check “Limited history” with HISTORY tables
 - Two “HISTORY” tables that record each instrumented events. The EVENTS_WAITS_HISTORY table contains the most recent 10 events per thread. And EVENTS_WAITS_HISTORY_LONG contains the most recent 10,000 events by default
 - For example, following query gives you exact mutex instances that has been on the top list as shown in the history table:

```
mysql> SELECT EVENT_NAME, SUM(TIMER_WAIT), COUNT(*), SOURCE  
-> FROM EVENTS_WAITS_HISTORY_LONG  
-> WHERE EVENT_NAME LIKE "%innodb%"  
-> GROUP BY SOURCE  
-> ORDER BY SUM(TIMER_WAIT) DESC;
```

- But “History” is short, even if we extend EVENTS_WAITS_HISTORY_LONG to 1 million events, you might record just a few seconds data on a busy system

Performance Schema HISTORY Table

```
mysql> SELECT EVENT_NAME, SUM(TIMER_WAIT), COUNT(*), SOURCE  
-> FROM EVENTS_WAITS_HISTORY_LONG  
-> WHERE EVENT_NAME LIKE "%innodb%"  
-> GROUP BY SOURCE  
-> ORDER BY SUM(TIMER_WAIT) DESC;
```

EVENT_NAME	SUM(TIMER_WAIT)	count(*)	source
buffer_block_mutex	12760956054	157504	buf0buf.ic:1036
buf_pool_mutex	6888003804	70691	buf0buf.c:2072
buffer_block_mutex	6401817972	70691	buf0buf.c:2314
buffer_block_mutex	5821074270	72217	buf0buf.c:2417
log_sys_mutex	3820390002	4911	log0log.ic:404
kernel_mutex	3574332630	2452	srv0srv.c:2160
buffer_block_mutex	1581747948	14624	buf0buf.c:2528
buf_pool_mutex	1547999046	17431	buf0buf.ic:1031
kernel_mutex	785501838	1311	lock0lock.c:3185

Performance Schema SUMMARY Table

- Find out aggregated information from SUMMARY Tables

```
mysql> SELECT EVENT_NAME, COUNT_STAR, SUM_TIMER_WAIT, AVG_TIMER_WAIT  
-> FROM EVENTS_WAITS_SUMMARY_BY_EVENT_NAME  
-> WHERE EVENT_NAME like "%innodb%"  
-> order by COUNT_STAR DESC;
```

EVENT_NAME	COUNT_STAR	SUM_TIMER_WAIT	AVG_TIMER_WAIT
buf_pool_mutex	1925253	264662026992	137468
buffer_block_mutex	720640	80696897622	111979
kernel_mutex	243870	44872951662	184003
purge_sys_mutex	162085	12238011720	75503
trx_undo_mutex	120000	11437183494	95309
rseg_mutex	102167	14382126000	140770
fil_system_mutex	97826	15281074710	156206
log_sys_mutex	80034	35446553406	442893
dict_sys_mutex	80003	6249472020	78115

Performance Schema in InnoDB – a few notes

- Individual Mutexes / rwlocks can be excluded from instrumentation in InnoDB using SETUP tables, although by default all mutexes / rwlock are enabled.
- The number of mutex / rwlock instance could be very large. Need to adjust new system variables to accommodate:
performance_schema_max_mutex_instances
 - Remember the block mutex/rwlock example, for a server with 2G buffer pool size configured, you would have 125,000 mutexes and rwlocks respectively.
- Additional defines in InnoDB so that each of 4 instrumented modules can be disabled from instrumentation
- Now the study on mutex/rwlock can be quantified!

Performance Impact from Performance Schema

- DBT2 test
 - 32 connections
 - 50 warehouses
 - 2G innodb_buffer_pool_size
 - 500 MB innodb_log_file_size

Server Configuration	NOTPM	% change
Without PS	9505	0
PS built-in Not enabled	9488	-0.1%
PS Enabled (InnoDB Instrument Disabled)	8585	-8.02%
PS Enabled	8574	-8.12%

Performance Impact from Performance Schema

- Performance Impact from Performance Schema(Data from Peter Gulutzan)

PERFSCHEMA-COMPILED-OUT	1	1	1	1	1	1
PERFSCHEMA-DISABLED	0.991871	0.99438	0.982554	0.978089	0.958384	0.973615
PERFSCHEMA-ENABLED-STANDBY	0.990765	0.986137	0.983541	0.977715	0.986779	0.98358
PERFSCHEMA-ENABLED-CURRENT	0.982305	0.972789	0.966903	0.958982	0.974484	0.980018
PERFSCHEMA-ENABLED-CURRENT-CYCLE	0.979761	0.966326	0.955422	0.947115	0.951505	0.977591
PERFSCHEMA-ENABLED-HISTORY-CYCLE	0.922473	0.971384	0.953987	0.949171	0.951993	0.971446
PERFSCHEMA-ENABLED-HISTORY_LONG-CYCLE	0.97025	0.958599	0.947933	0.936931	0.951456	0.97408
PERFSCHEMA-ENABLED-BIGBANG-CYCLE	0.966434	0.956772	0.936721	0.939547	0.955408	0.975784

Performance Schema in InnoDB

- Performance Schema helps us study and quantify the low level server events
- Good tool for event monitoring and development tuning, especially for mutex optimization
- It comes a cost, but working to minimize the cost

Introducing InnoDB Metrics Table

- What is InnoDB Metrics Table
 - Infrastructure for Monitor counter based monitoring system
 - Display through information schema tables
 - Light weight counters, rely on caller for synchronization protection
 - Control system to turn on/off and reset the monitors. Counter start to counting only if the counter is turned on
 - Used for resource usage (capacity) as performance counters
 - Important for “knobs” – configurable options
 - Easy to add additional counters
 - No measurable performance impact

Column Defines for Monitor Table

<i>column name</i>	<i>data type</i>	<i>description</i>
<i>metric_name</i>	<i>string</i>	<i>contains the name of some metric as a string</i>
<i>subsystem</i>	<i>string</i>	<i>the name or the feature the metric pertains to</i>
<i>value_since_start</i>	<i>int</i>	<i>value since start of counter</i>
<i>max_since_start</i>	<i>int (nullable)</i>	<i>max value since the start</i>
<i>min_since_start</i>	<i>int (nullable)</i>	<i>min value since the start</i>
<i>avg_since_start</i>	<i>int (nullable)</i>	<i>average since the start</i>
<i>value_since_reset</i>	<i>int</i>	<i>value since last reset</i>
<i>max_since_reset</i>	<i>int (nullable)</i>	<i>max value since last reset</i>
<i>min_since_reset</i>	<i>int (nullable)</i>	<i>min value since last reset</i>
<i>avg_since_reset</i>	<i>float (nullable)</i>	<i>avg value since last reset</i>
<i>start_time</i>	<i>timestamp (nullable)</i>	<i>Timestamp of last start</i>
<i>stop_time</i>	<i>timestamp (nullable)</i>	<i>Timestamp of last stop</i>
<i>status</i>	<i>string</i>	<i>whether the counter is still running or stopped</i>
<i>type</i>	<i>string</i>	<i>whether the counter is incremental counter or resource related</i>
<i>description</i>	<i>string</i>	<i>Description of the counter</i>

InnoDB Metrics Monitor Table

- What does the metrics table look like:

```
mysql> SELECT * FROM information_schema.innodb_metrics  
-> WHERE name="server_table_open" \G
```

```
***** 1. row *****
```

```
name: server_table_open  
subsystem: Server Metadata  
value_since_start: 12  
max_since_start: 12  
min_since_start: NULL  
avg_since_start: NULL  
value_since_reset: 5  
max_since_reset: 5  
min_since_reset: NULL  
avg_since_reset: NULL  
start_time: NULL  
stop_time: NULL  
status: stopped  
type: counter_value  
description: Number of table handler opened  
1 row in set (0.00 sec)
```

Control System for Monitor Counters

- Control System:
 - 1) Turn on the monitor:
 - set global innodb_monitor_counter_on
 - Ex. set global innodb_monitor_counter_on = server_table_open;
 - 2) After the sampling period, stop the monitor counting:
 - set global innodb_monitor_counter_off
 - 3) Reset the counter:
 - set global innodb_monitor_counter_reset
 - 4) Reset all of the counter
 - set global innodb_monitor_counter_reset_all

Monitor Counters Grouped into Modules

- Counters are grouped into modules:

	module name
1	module_server
2	module_lock
3	module_buffer
4	module_trx
5	module_log
6	module_page
7	module_index
8	module_dml

- Counters in the module can be turned on/off/reset together by applying the module name to the control system.

Monitor Counter Examples

- An example: obtain DML stats in a fixed period of time

```
mysql> SET GLOBAL innodb_monitor_counter_on = module_dml;  
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> SELECT name, max_since_start, value_since_start, status  
-> FROM information_schema.innodb_metrics  
-> WHERE name LIKE "dml%";
```

name	max_since_start	value_since_start	status
dml_num_reads	0	0	started
dml_num_inserts	24295254	24295254	started
dml_num_deletes	0	0	started
dml_updates	0	0	started

4 rows in set (0.00 sec)

consumer	1500000
district	500
history	1500000
item	100000
new_order	450000
order_line	14244704
orders	1500000
stock	5000000
warehouse	50
Total	24295254

Monitor Counter Examples

- Getting more information:
 - SET global innodb_monitor_counter_off = module_dml;
 - SET global innodb_monitor_counter_reset_all = all;
 - SET global innodb_monitor_counter_on = all;
 - SET global innodb_monitor_counter_off = all;

<i>Counter</i>	<i>Value</i>
<i>server_table_open</i>	<i>141</i>
<i>server_table_close</i>	<i>0</i>
<i>dml_num_reads</i>	<i>5571754</i>
<i>dml_num_inserts</i>	<i>87584</i>
<i>dml_num_deletes</i>	<i>5180</i>
<i>dml_updates</i>	<i>155683</i>
<i>log_num_checkpoint</i>	<i>413</i>
<i>log_lsn_last_flush</i>	<i>5978767236</i>
<i>trx_num_commit</i>	<i>14244</i>
<i>trx_num_abort</i>	<i>63</i>
<i>index_num_split</i>	<i>2538</i>
<i>buffer_read_ahead_evict</i>	<i>42</i>

Why Metrics Monitor Table

- Easy to query and aggregates with table format
- Minimum performance impact.
- Easy to extend. With this infrastructure, add monitor counter is easy
- Generic. Can be used for a variety of monitoring purpose as well as statistics collection
- One stop provider for all monitor info. 31 Existing system status variables are now supported through Metrics Counter (but will still support status variables)
- All status variable will have entry in the metrics table

Performance Schema vs. Metrics Counters

- Performance Schema vs. Metrics Counters
 - They are complimentary. Performance schema gives more information on events, mutex / rwlock. Metrics counter is a more generic infrastructure for resource and performance counting.
 - Both tried to avoid involving mutexes. And counter value could be approximate.
 - For most metrics counters, upper level synchronization protection gives it a relatively reliable value.
 - No measurable performance impact from metrics counter, designed to be generic, simple and cheap.

Summary

- Metrics System/Tables are likely to be our main interfaces for obtaining performance statistics and resource usage information. More performance and resource counters can be added with ease. And it will be one-stop shop for most performance/resource information
- Performance Schema in InnoDB gives us powerful tools to study events such as mutex, rwlock and I/O related performance bottlenecks. Its value will soon be proved.

References and More Information

- Session: “Performance Schema”
 - Peter Gulutzan (MySQL AB)
 - 11:55am Tuesday, 04/13/2010
- Session: “Mastering InnoDB Diagnostics”
 - Harrison Fisk (Oracle Corporation)
 - 3:05pm Wednesday, 04/14/2010
- Performance Documentation
 - <http://dev.mysql.com/doc/refman/5.5/en/performance-schema.html>
- Blog on Performance Schema in InnoDB
 - <http://blogs.innodb.com/wp/2010/04/innodb-performance-schema>



ORACLE®

- The preceding is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.