Tracing and Profiling MySQL

Mostly with Linux tools: from strace and gdb to ftrace, bpftrace, perf and dynamic probes

Valerii Kravchuk, Principal Support Engineer, MariaDB
vkravchuk@gmail.com
Who am I?

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](http://mysqlentomologist.blogspot.com) - my blog about MySQL (a lot about MySQL bugs, but some **HowTos** as well)
- [https://www.facebook.com/valerii.kravchuk](https://www.facebook.com/valerii.kravchuk) - my Facebook page, a lot about MySQL (mostly bugs…)
- [http://bugs.mysql.com](http://bugs.mysql.com) - my personal playground
- [@mysqlbugs #bugoftheday](https://twitter.com/mysqlbugs/status/1234567890)
- **Community Contributor of the Year 2019**
- I like [FOSDEM](https://fosdem.org) and used to participate at [Percona Live](https://www.percona.com) conferences…
Sources of tracing and profiling info for MySQL

- Trace files from `-debug` binaries, optimizer trace files
- *(Extended)* slow query log (thanks Percona!)
- `show [global] status;`
- `show engine innodb status\G`
- `show engine innodb mutex;`
- InnoDB-related tables in the INFORMATION_SCHEMA
- `userstat` (Percona Server and other builds)
- `show profiles;`
- PERFORMANCE_SCHEMA
- Profilers (even simple like `pt-pmp` or real like `perf`)
- OS-level tracing and profiling tools
- `tcpdump analysis`
What is this session about?

- It’s about **tracing** and **profiling** MySQL, and mostly some tools MySQL DBA can use for tracing and profiling in **production** on Linux:
  - **perf** (I think it’s the best and easiest to use now)
  - few words on **PMP** (**pt-pmp**)
  - **ftrace**
  - eBPF and related tools (like **bpftrace**)
  - … and maybe more...

- Why not about **gprof**, **Callgrind**, **Massif**, **dtrace**, **SystemTap**?
- Why not entirely about **Performance Schema**?
- **Performance impact** of tracing and profiling
Why not about Performance Schema?

- It may be NOT compiled in (see MySQL from Facebook)
- It may be NOT enabled when server was started (see MariaDB)
- Specific instruments may not be enabled at startup and then it’s too late (see Bug #68097)
- Sizing instruments properly (memory used and performance impact vs details collected) may be problematic (depends on version also)
- Part of the code or 3rd party plugin may not be instrumented at all or in enough details (see Bug #83912)
- It does not give you a system-wide profiling, just for selected parts of MySQL server code
- Other people (including myself) talk and write a lot about it
Not Enough Details in Performance Schema

Samples: 52K of event 'cpu-clock', Event count (approx.): 13037500000

<table>
<thead>
<tr>
<th>Overhead</th>
<th>Command</th>
<th>Shared Object</th>
<th>Symbol</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>43.75%</td>
<td>mysqld</td>
<td>mysqld</td>
<td>[.]</td>
<td>[.]</td>
</tr>
<tr>
<td>Item_func_mul::int_op</td>
<td>mysqld</td>
<td>mysqld</td>
<td>[.]</td>
<td>[.]</td>
</tr>
<tr>
<td>16.97%</td>
<td>mysqld</td>
<td>mysqld</td>
<td>[.]</td>
<td>[.]</td>
</tr>
<tr>
<td>Item_func_benchmark::val_int</td>
<td>mysqld</td>
<td>mysqld</td>
<td>[.]</td>
<td>[.]</td>
</tr>
<tr>
<td>14.10%</td>
<td>mysqld</td>
<td>mysqld</td>
<td>[.]</td>
<td>[.]</td>
</tr>
<tr>
<td>Item_int::val_int</td>
<td>mysqld</td>
<td>mysqld</td>
<td>[.]</td>
<td>[.]</td>
</tr>
<tr>
<td>13.50%</td>
<td>mysqld</td>
<td>mysqld</td>
<td>[.]</td>
<td>[.]</td>
</tr>
<tr>
<td>Item_func_numhybrid::val_int</td>
<td>mysqld</td>
<td>mysqld</td>
<td>[.]</td>
<td>[.]</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.58%</td>
<td>mysqld</td>
<td>mysqld</td>
<td>[.]</td>
<td>[.]</td>
</tr>
<tr>
<td>Item_func_numhybrid::result_type</td>
<td>mysqld</td>
<td>mysqld</td>
<td>[.]</td>
<td>[.]</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

30 SELECT `benchmark` ( ?, ... * ? )
(13055172.39?)
30 stage/sql/init (51.56?)
30 stage/sql/checking permissions
(2.27?)
30 stage/sql/Opening tables (1.00?)
30 stage/sql/After opening tables
(0.62?)
30 stage/sql/init (9.32?)
30 stage/sql/optimizing (7.41?)
30 stage/sql/executing (13055061.32?)
30 stage/sql/end (3.98?)
30 stage/sql/query end (2.34?)
30 stage/sql/closing tables (1.73?)
30 stage/sql/freeing items (4.22?)
30 stage/sql/cleaning up (1.13?)

- Yes, this is for primitive `select benchmark(500000000,2*2)` from Bug #39630
- Performance Schema query is like 20 lines long to make it readable
Typical “profiling” query to Performance Schema

- This is how it may look like:

```
SELECT thread_id, event_id, nesting_event_id, CONCAT(CASE WHEN event_name LIKE 'stage%' THEN CONCAT(' ', event_name) WHEN event_name LIKE 'wait%' AND nesting_event_id IS NOT NULL THEN CONCAT(' ', event_name) ELSE IF(digest_text IS NOT NULL, SUBSTR(digest_text, 1, 64), event_name) END, ' (',ROUND(timer_wait/1000000000, 2),'ms) ') event
FROM (  
(SELECT thread_id, event_id, event_name, timer_wait, timer_start, nesting_event_id, digest_text FROM events_statements_history_long)
UNION
(SELECT thread_id, event_id, event_name, timer_wait, timer_start, nesting_event_id, NULL FROM events_stages_history_long)
UNION
(SELECT thread_id, event_id, event_name, timer_wait, timer_start, nesting_event_id, NULL FROM events_waits_history_long)
) events
ORDER BY thread_id, event_id;
```
Not Enough Details in Performance Schema

- Now, where the time is spent on “statistics” stage in case presented in Bug #83912?

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>26</td>
<td>379</td>
<td>NULL</td>
<td>SELECT * FROM <code>t0</code> WHERE ID = ? (13072.50ms)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>380</td>
<td>379</td>
<td>stage/sql/init (0.05ms)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>383</td>
<td>379</td>
<td>stage/sql/checking permissions (0.00ms)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>384</td>
<td>379</td>
<td>stage/sql/Opening tables (0.02ms)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>386</td>
<td>379</td>
<td>stage/sql/After opening tables (0.00ms)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>387</td>
<td>379</td>
<td>stage/sql/System lock (0.00ms)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>389</td>
<td>379</td>
<td>stage/sql/Table lock (0.00ms)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>391</td>
<td>379</td>
<td>stage/sql/init (0.02ms)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>392</td>
<td>379</td>
<td>stage/sql/optimizing (0.01ms)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>393</td>
<td>379</td>
<td>stage/sql/statistics (13072.32ms)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>396</td>
<td>379</td>
<td>stage/sql/preparing (0.00ms)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>397</td>
<td>379</td>
<td>stage/sql/Unlocking tables (0.02ms)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>398</td>
<td>379</td>
<td>stage/sql/executing (0.00ms)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>399</td>
<td>379</td>
<td>stage/sql/Sending data (0.01ms)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>400</td>
<td>379</td>
<td>stage/sql/end (0.00ms)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>401</td>
<td>379</td>
<td>stage/sql/query end (0.00ms)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
So, what do I suggest?

- Use Linux tracing tools!
- Yes, all that “…strace, and ltrace, kprobes, and tracepoints, and uprobes, and ftrace, and perf, and eBPF”
- **Julia Evans** explains and illustrates them all [here](http://www.percona.com)
- **Brendan D. Gregg** explains them all with a lot of details and numerous examples:
Few words on strace

• **strace** may help MySQL DBA to find out:
  • what files are accessed by the **mysqld** process or utilities, and in what order
  • why some MySQL-related command (silently) fails or hangs
  • why some commands end up with permission denied or other errors
  • what signals MySQL server and tools get
  • what system calls could took a lot of time when something works slow
  • when files are opened and closed, and how much data are read
  • where the error log and other logs are really located (we can look for system calls related to writing to stderr, for example)
  • how MySQL really works with files, ports and sockets
• See [my blog post](#) for more details
• **Use in production as a last resort** (**2 interrupts per system call**, even not those we care about, may leave traced process hanged)
• **strace** surely **slows server down**
Few words on DTrace

- DTrace
- If you use Oracle Linux, go for it! Making it work on Fedora 29 took me too much time to complete for the talk
Few words on SystemTap (stap)

- SystemTap:
Few words on pt-pmp (Poor Man’s Profiler)


  ```
  ```

- It is based on original idea by Domas, [http://poormansprofiler.org/](http://poormansprofiler.org/). I use the awk code from the above to analyse backtraces of all threads.

- When **mysqld** hangs or is slow, you can get some insight quickly - use **pt-pmp** to find out why (or what threads mostly do at the moment). For example, see [Bug #92108](http://bugs.launchpad.net/percona-toolkit/+bug/92108) (fixed in 5.7.25+, binlog access vs P_S query),

- Yet another example of how it is used: [Bug #78277](http://bugs.launchpad.net/percona-toolkit/+bug/78277) - InnoDB deadlock, thread stuck on kernel calls from transparent page compression, “Open”

- **Use in production as a last resort** (may hang **mysqld**, --SIGCONT)

- **pt-pmp** surely **slows server down** :) Hints:
  - [https://bugs.launchpad.net/percona-toolkit/+bug/1320168](https://bugs.launchpad.net/percona-toolkit/+bug/1320168) - partial workaround
  - Use **quickstack** instead of **gdb** (check this discussion)
pt-pmp Applied to “statistics” Case of Bug #83912

MariaDB [test]> select * from t0 where id = 15;
+----+------+--------------------+
<table>
<thead>
<tr>
<th>id</th>
<th>c1</th>
<th>c2</th>
</tr>
</thead>
</table>
+----+------+--------------------+
| 15 | 290  | 0.7441205286831786 |
+----+------+--------------------+
1 row in set (52.27 sec)

select(libc.so.6), os_thread_sleep(os0thread.cc:303), srv_conc_enter_innodb_with_atomics(srv0conc.cc:298), srv_conc_enter_innodb(srv0conc.cc:298), innobase_srv_conc_enter_innodb(ha_innodb.cc:1906), ha_innodb::index_read(ha_innodb.cc:1906), handler::index_read_idx_map(handler.cc:5441), handler::ha_index_read_idx_map(handler.cc:2646), join_read_(handler.cc:2646), join_read__table(handler.cc:2646), make_join_statistics(sql_select.cc:3935), JOIN::optimize_inner(sql_select.cc:1366), JOIN::optimize(sql_select.cc:1045), mysql_select(sql_select.cc:3430), handle_select(sql_select.cc:372), execute_sqlcom_select(sql_parse.cc:5896), mysql_execute_command(sql_parse.cc:2971), mysql_parse(sql_parse.cc:7319), dispatch_command(sql_parse.cc:1488), do_command(sql_parse.cc:1109), do_handle_one_connection(sql_connect.cc:1349), handle_one_connection(sql_connect.cc:1261), pfs_spawn_thread(pfs.cc:1860), start_thread(libpthread.so.0), close(libc.so.6)

...
A lot about tracing sources

•

•
A lot about ftrace

- **ftrace**
- The way you fundamentally interact with **ftrace** is:
  - Write to files in `/sys/kernel/debug/tracing/`
  - Read output from files in `/sys/kernel/debug/tracing/`
A lot about eBPF

- **eBPF** is a tiny language for a VM that can be executed inside Linux Kernel. **eBPF** instructions can be JIT-compiled into a native code. **eBPF** was originally conceived to power tools like *tcpdump* and implement programmable network packet dispatch and tracing. Since Linux 4.1, **eBPF** programs can be attached to *kprobes* and later - *uprobes*, enabling efficient programmable tracing.

- Brendan Gregg explained it [here](#).

---

**Diagram:**

User Program:
1. generate
   - BPF bytecode
2. load
3. perf_output
   - per-event data
   - statistics

Kernel:
- verifier
- BPF
- maps
- kprobes
- uprobes
- tracepoints
- perf_events

3. async read
Few words about bpftrace

●
●
A lot about perf

- If you are interested in details presented nicely...
Adding uprobes to MySQL dynamically with perf

- The idea was to add dynamic probe to capture SQL queries
- This was done on Ubuntu 16.04 with recent Percona Server 5.7
- First I had to find out with `gdb` where is the query (hint: `dispatch_command` has `com_data` parameter):
  ```
  (gdb) p com_data->com_query.query
  $4 = 0x7fb0dba8d021 "select 2"
  ```
- Then it’s just as easy as follows:

  ```
  openxs@ao756:~$ sudo perf probe -x /usr/sbin/mysqld 'dispatch_command.com_data->com_query.query:string'
  openxs@ao756:~$ sudo perf record -e 'probe_mysqld:dispatch_command*' -aR
  ^C[ perf record: Woken up 1 times to write data ]
  [ perf record: Captured and wrote 0.676 MB perf.data (3 samples) ]
  openxs@ao756:~$ sudo perf script >/tmp/queries.txt
  openxs@ao756:~$ sudo perf probe --del dispatch_command
  ```
Adding uprobes to MySQL dynamically with perf

- We have queries captured with probe added on previous slide:

```bash
openxs@ao756:~$ cat /tmp/queries.txt

mysqld 31340 [001] 3888.849079: probe_mysqld:dispatch_command:
  (be9250) query="select 100"

mysqld 31340 [001] 3891.648739: probe_mysqld:dispatch_command:
  (be9250) query="select user, host from mysql.user"

mysqld 31340 [001] 3895.890141: probe_mysqld:dispatch_command:
  (be9250) query="select 2"
```

- We can control output format, but basically we see binary, PID, CPU where uprobe was executed on, timestamp (milliseconds since start of record), probe and variables with format we specified.
perf - Success Stories

- **perf** (sometimes called **perf_events** or **perf tools**, originally **Performance Counters for Linux, PCL**) is a new performance analyzing tool for Linux, available from kernel version 2.6.31 (supported by RHEL6 since 2010)
- It is easier to use and more popular recently for MySQL than older profilers
- Here is the list of some MySQL bugs by **Mark Callaghan** confirmed by **perf**:
  - **Bug #69236** - "performance regressions for single-threaded workloads, part 2" - MySQL 5.6 is spending a lot more time in `rec_get_offsets_func`, `trx_undo_report_row_operation`, `btr_cur_optimistic_insert`. Same in 5.7.8, “Verified”
  - **Bug #74325** - “updates to indexed column much slower in 5.7.5” - nice **perf** outputs there. It’s about `innodb_fill_factor=100` (that leaves 1/16 free space since 5.7.8).
  - **Bug #74280** - “covering secondary scans worse than PK scan at high concurrency” - the mutex contention that isn't visible in P_S output because the block rw-lock isn't instrumented. Verified regression since 5.7.5 vs 5.6.x. See also: **Bug #74283** - “Non-covering secondary index scans degrade more in 5.7 than 5.6”
  - [http://smalldatum.blogspot.com/2014/10/details-on-range-scan-performance.html](http://smalldatum.blogspot.com/2014/10/details-on-range-scan-performance.html) - on two bugs above, **perf** allows to see the difference
perf - Basic Usage

● Check my post, “perf Basics for MySQL Profiling”, for details and references, but basic minimal steps are:
  ○ Make sure perf-related packages are installed (perf with RPMs) for your kernel:
    sudo apt-get install linux-tools-generic
  ○ Make sure debug symbols are installed and software is built with -fno-omit-frame-pointer
  ○ Start data collection for some time using perf record:
    sudo perf record -a [-g] [-F99] [-p `pidof mysqld`] sleep 30
    Run your problematic load against MySQL server
  ○ Samples are collected in `pwd`/perf.data by default
  ○ Process samples and display the profile using perf report:
    sudo perf report [-n] [-g] --stdio

● Alternatively, run in foreground and interrupt any time with Ctrl-C:
  [root@centos ~]# perf record -ag
  ^C

● Or run in background and send -SIGINT when done:
  [root@centos ~]# perf record -ag &
  [1] 2353
  [root@centos ~]# kill -sigint 2353

● Let’s see how it works alive… (demo). We’ll see perf top, perf record -g etc
perf - Call Graphs

Use `-g` option of `perf record` to get call graphs/backtraces with `perf`, then:

```
openxs@ao756:~/dbs/maria10.1$ sudo perf report --stdio
...
  31.02%   mysqld   mysqld
  | --- Item_func_mul::int_op()
  |   |-- Item_func_mul::int_op()
  |   |   |-- Item_func_hybrid_field_type::val_int()
  |   |   |   |   |   Item_func_benchmark::val_int()
  |   |   |   |   |   |   Item::send(Protocol*, String*)
  |   |   |   |   |   |   Protocol::send_result_set_row(List<Item>*)
  |   |   |   |   |   |   select_send::send_data(List<Item>*)
  |   |   |   |   |   |   JOIN::exec_inner()
  |   |   |   |   |   |   JOIN::exec()
  |   |   |   |   |   |   mysql_select(THD*, Item***, TABLE_LIST*, ...
  |   |   |   |   |   |   |   handle_select(THD*, LEX*, select_result*, unsigned long)
  |   |   |   |   |   |   |   execute_sqlcom_select(THD*, TABLE_LIST*)
  |   |   |   |   |   |   |   mysql_execute_command(THD*)
  |   |   |   |   |   |   |   mysql_parse(THD*, char*, unsigned int, Parser_state*)
  |   |   |   |   |   |   |   dispatch_command(enum_server_command, THD*, char*, ...
  |   |   |   |   |   |   |   do_command(THD*)
  ...
```
Studying Hanging in “statistics” Case(s)

- See [my blog post](#) for details and full outputs:

```
| | --71.70%-- srv_conc_enter_innodb(trx_t*)
| |    ha_innобase::index_read(...)  
| |    handler::index_read_idx_map(...)  
| |    handler::ha_index_read_idx_map(...)  
| |    join_read_const(st_join_table*)  
| |    join_read_const_table(THD*, ...)  
| |    make_join_statistics(JOIN*, ...)  
| |    JOIN::optimize_inner()  
| |    JOIN::optimize()  
| |    mysql_select(THD*, ...)  
```

- We can see that time to do [SELECT](#) is mostly spent waiting to enter InnoDB queue while reading data via index (dive) to get statistics for the optimizer.
- We can see where the time is spent by kernel and other processes (`-a`)
Am I crazy doing these?

- Quite possible, maybe I just have too much free time :)
- Or maybe I do not know how to use Performance Schema properly
- But I am not alone...
- MariaDB developers are interested in adding dynamic probes with `perf` while working on RocksDB storage engine performance problems
- People use `perf` probes for tracing Oracle RDBMS! There is enough instrumentation there for almost everything, but still...
- Dynamic tracers are proven tools for **instrumenting OS calls** (probes for measuring I/O latency at microsecond precision, for example)
- Another topic, more complex but also more exciting, is dynamic tracing of RDBMS **userspace**. This topic is of growing interest with modern servers hosting large amounts of RAM and workloads that are often CPU-bound. There are no “waits”, but time is still spent somewhere!
- For open source RDBMS like MySQL there is no reason NOT to use dynamic probes while UDST are not on every other line of the code :)
- **eBPF** and **bpftrace** make it even easier and safer to do these in production
Thank you!

Questions and Answers?

Please, report bugs at:
https://bugs.mysql.com
https://jira.mariadb.org
https://jira.percona.com