MariaDB Performance Analysis with Linux BPF

How I pinpointed a MariaDB scalability issue with a BPF script.

Max Kellermann (max.kellermann@ionos.com)

September 17, 2024

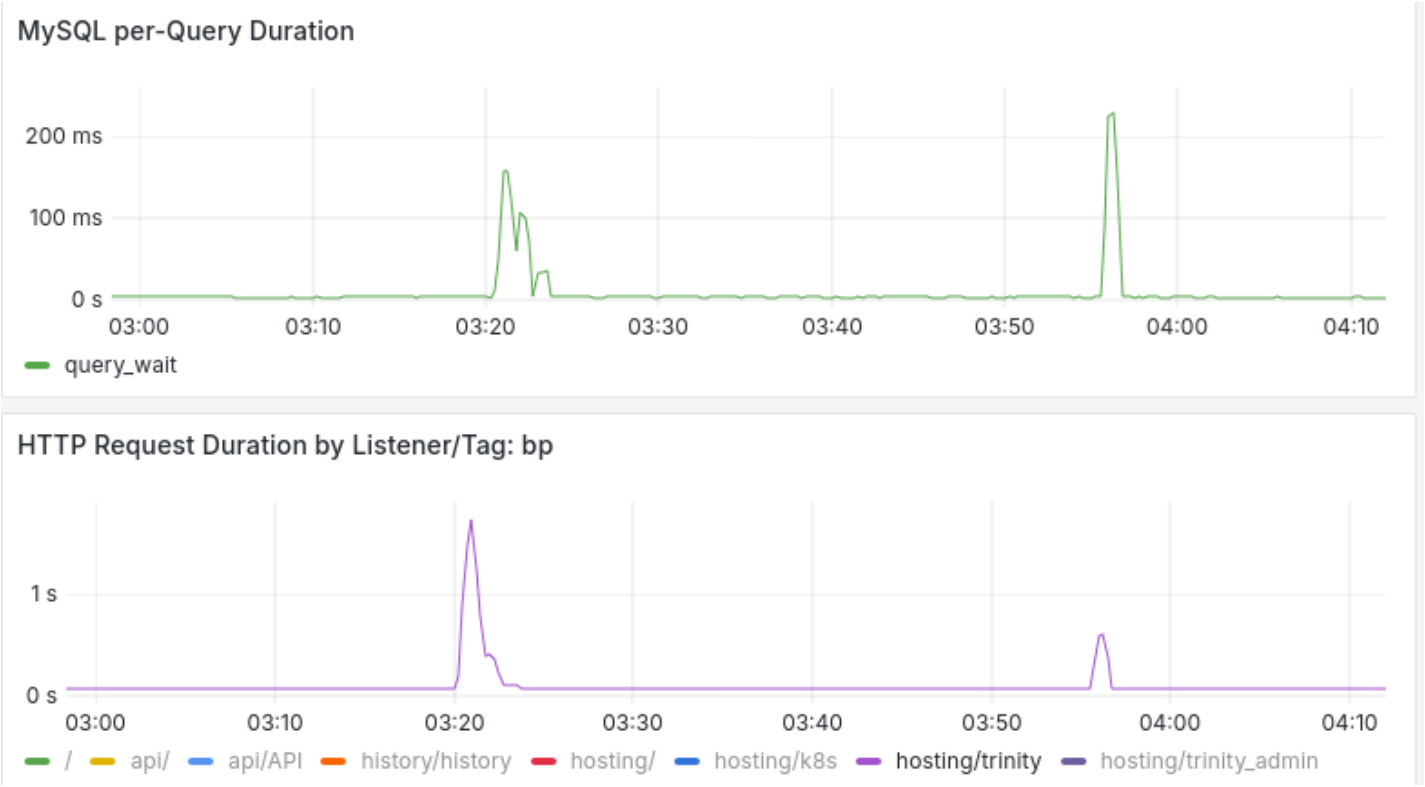
About me

Max Kellermann

- Linux hacker and Open Source enthusiast
- working for 20+ years at CM4all (now part of IONOS)
- building shared webhosting infrastructure with C++
- visit github.com/CM4all - much of my code is Open Source!

Shared webhosting = runtime for PHP (WordPress).
Heavy load on MariaDB servers, thousands of clients.

Problem: WordPress Latency Spikes



WordPress spikes from 50ms to several seconds.

Correlation with slow database queries.

Many MariaDB Writes?

No obvious correlation with database writes.

MariaDB Slow Query Log

```
# Thread_id: 160714161  Schema: HOSSSSSXBXMA  QC_hit: No
# Query_time: 5.358472  Lock_time: 0.000592  Rows_sent: 4  Rows_examined: 4
# Rows_affected: 0  Bytes_sent: 1065
SHOW FULL COLUMNS FROM wp_options;
# Thread_id: 162118047  Schema: HOSSSSSXOSAH  QC_hit: No
# Query_time: 4.288169  Lock_time: 0.000743  Rows_sent: 4  Rows_examined: 4
# Rows_affected: 0  Bytes_sent: 1065
SHOW FULL COLUMNS FROM wp_options;
# Thread_id: 162117201  Schema: HOSSSSSXYAVC  QC_hit: No
# Query_time: 23.411452  Lock_time: 0.003371  Rows_sent: 0  Rows_examined: 0
# Rows_affected: 0  Bytes_sent: 50
ALTER TABLE wp_yoast_migrations CONVERT TO CHARACTER SET utf8mb4 COLLATE [...]
# Thread_id: 162118038  Schema: HOSSSSSXPENK  QC_hit: No
# Query_time: 4.456552  Lock_time: 0.000669  Rows_sent: 4  Rows_examined: 4
# Rows_affected: 0  Bytes_sent: 1065
SHOW FULL COLUMNS FROM wp_options;
# Thread_id: 162079502  Schema: HOSSSSSXFPAR  QC_hit: No
# Query_time: 6.407237  Lock_time: 0.000606  Rows_sent: 4  Rows_examined: 4
# Rows_affected: 0  Bytes_sent: 1065
SHOW FULL COLUMNS FROM wp_options;
```

`ALTER TABLE` takes 23 seconds.

Why is `SHOW FULL COLUMNS` slow?

MariaDB Slow Query Log

```
# Thread_id: 160714161  Schema: HOSSSSSXBXMA  QC_hit: No
# Query_time: 5.358472  Lock_time: 0.000592  Rows_sent: 4  Rows_examined: 4
# Rows_affected: 0  Bytes_sent: 1065
SHOW FULL COLUMNS FROM wp_options;
# Thread_id: 162118047  Schema: HOSSSSSXOSAH  QC_hit: No
# Query_time: 4.288169  Lock_time: 0.000743  Rows_sent: 4  Rows_examined: 4
# Rows_affected: 0  Bytes_sent: 1065
SHOW FULL COLUMNS FROM wp_options;
# Thread_id: 162117201  Schema: HOSSSSSXYAVC  QC_hit: No
# Query_time: 23.411452  Lock_time: 0.003371  Rows_sent: 0  Rows_examined: 0
# Rows_affected: 0  Bytes_sent: 50
ALTER TABLE wp_yoast_migrations CONVERT TO CHARACTER SET utf8mb4 COLLATE [...]
# Thread_id: 162118038  Schema: HOSSSSSXPENK  QC_hit: No
# Query_time: 4.456552  Lock_time: 0.000669  Rows_sent: 4  Rows_examined: 4
# Rows_affected: 0  Bytes_sent: 1065
SHOW FULL COLUMNS FROM wp_options;
# Thread_id: 162079502  Schema: HOSSSSSXFPAR  QC_hit: No
# Query_time: 6.407237  Lock_time: 0.000606  Rows_sent: 4  Rows_examined: 4
# Rows_affected: 0  Bytes_sent: 1065
SHOW FULL COLUMNS FROM wp_options;
```

`ALTER TABLE` takes 23 seconds.

Why is `SHOW FULL COLUMNS` slow?

```
MySQL [HOSSSSSXYAVC]> ALTER TABLE wp_yoast_migrations CONVERT TO [...]
Query OK, 0 rows affected (0.009 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Does not reproduce.

Counting Slow Queries

Counting slow queries:

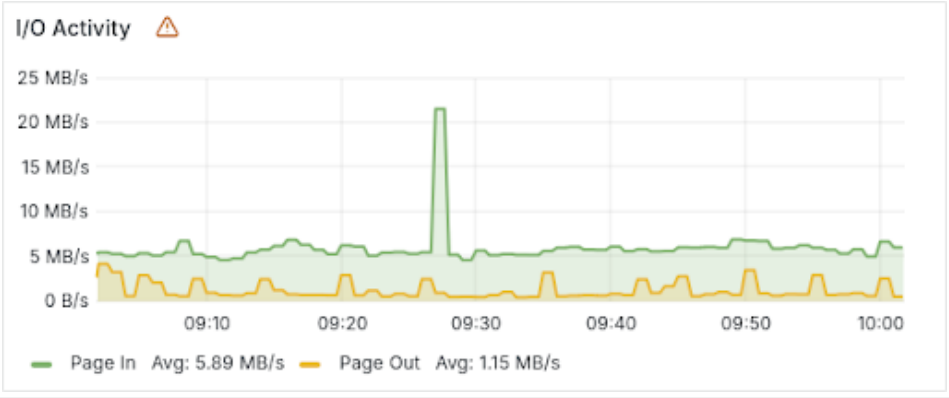
```
808 SHOW FULL COLUMNS FROM wp_options;
  1 ALTER TABLE wp_yoast_migrations CONVERT TO CHARACTER SET utf8mb4 COLLATE [...]
  1 ALTER TABLE wp_yoast_primary_term CONVERT TO CHARACTER SET utf8mb4 [...]
  1 DROP INDEX permalink_hash ON wp_yoast_indexable;
```

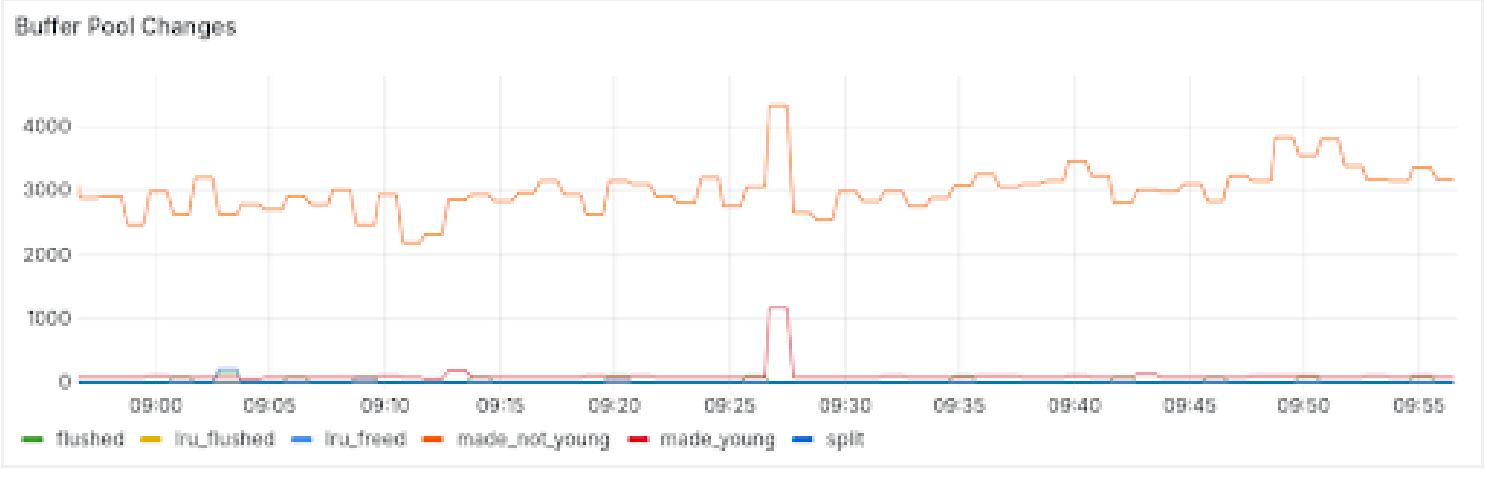
The modified tables/indexes are (almost) empty.

Cannot reproduce, only happens every few hours. No correlation with HTTP request load.

Made Young?

Desperately looking for correlations in our metrics.





Read spike, made_young spike.
I don't know what made_young is, but let's look at disk I/O!

Looking for Analysis Tools

How do you analze I/O bottlenecks in a daemon with tens of thousands of threads?

Looking for Analysis Tools

How do you analze I/O bottlenecks in a daemon with tens of thousands of threads?

Let's try `ext4slower` (from BCC):

```
Tracing ext4 operations slower than 10 ms
TIME     COMM       PID    T BYTES   OFF_KB   LAT(ms) FILENAME
12:44:01 mariadbd   1540   W 512     52116      29.33 ib_logfile0
12:44:01 mariadbd   1540   R 16384   6188608    10.04 ibdata1
12:46:10 mariadbd   1540   R 16384   1463536    42.46 ibdata1
12:46:10 mariadbd   1540   W 512     95784     113.37 ib_logfile0
12:46:10 mariadbd   1540   S 0       0          29.86 ib_logfile0
12:46:10 mariadbd   1540   R 16384   1704896    29.83 ibdata1
12:46:10 mariadbd   1540   R 16384   1701408    20.54 ibdata1
12:46:10 mariadbd   1540   W 1024    95784      20.59 ib_logfile0
12:47:48 mariadbd   1540   R 16384   48        116.52 wp_options.ibd
12:47:48 mariadbd   1540   R 16384   12898080   29.84 innodb_index_stats.ibd
12:47:48 mariadbd   1540   W 1024    34262      25.09 ib_logfile0
12:50:11 mariadbd   1540   R 16384   336        51.16 wp_posts.ibd
```

Some I/O is slow, but that alone does not explain the huge spikes.

$\Rightarrow$ Interesting tool, but can I have metrics more specific to my problem?

BPF to the rescue!

BPF is a Linux kernel feature to run scripts inside the kernel.

BPF to the rescue!

BPF is a Linux kernel feature to run scripts inside the kernel.

Example using `bpftrace` (a BPF scripting language):

```
root@pxc23:~# bpftrace -e 't:syscalls:sys_enter_openat {
  printf("%s(%d) opening %s\n", comm, tid, str(args->filename)
}'
mariadbd(478480) opening ./HOSSSSSXFBDI/wp_postmeta.frm
mariadbd(478480) opening ./HOSSSSSXFBDI/wp_postmeta.ibd
mariadbd(478480) opening ./HOSSSSSXFBDI/wp_usermeta.frm
mariadbd(486188) opening ./HOSSSSSXFIPW/wp_posts.frm
mariadbd(478480) opening ./HOSSSSSXFBDI/wp_users.frm
mariadbd(486188) opening ./HOSSSSSXFIPW/wp_postmeta.frm
mariadbd(482873) opening /tmp/#sql-temptable-604-b2a571-2c.MAI
mariadbd(482873) opening /tmp/#sql-temptable-604-b2a571-2c.MAD
mariadbd(482873) opening /tmp
mariadbd(482873) opening #sql-temptable-604-b2a571-2c.MAI
mariadbd(482873) opening /tmp/#sql-temptable-604-b2a571-2c.MAD
mariadbd(485078) opening ./HOSSSSSXOKZN/wp_posts.frm
[...]
```

Attached to the `openat` system call.

Userspace BPF (uprobes)

Attach BPF to arbitrary userspace functions:

```
# bool alloc_query(THD *thd, const char *packet, size_t packet_length);
root@pxc23:~# bpftrace -e 'u:/usr/sbin/mariadbd:_Z11alloc_queryP3THDPKcm {
  printf("%d %s\n", tid, str(arg1))
}'
482052 SHOW FULL COLUMNS FROM wp_options
475464 SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
475464 BEGIN;
482052 UPDATE wp_options SET option_value = '0' WHERE option_name
479981 SELECT ID, post_name, post_parent, post_type FROM wp_posts
478216 SET NAMES utf8mb4
475085 SET NAMES utf8mb4
478216 SET NAMES 'utf8mb4' COLLATE 'utf8mb4_unicode_520_ci'
478216 SELECT @@SESSION.sql_mode
475085 SET NAMES 'utf8mb4' COLLATE 'utf8mb4_unicode_520_ci'
486733 COMMIT;
475085 SELECT @@SESSION.sql_mode
475085 SET SESSION sql_mode='ERROR_FOR_DIVISION_BY_ZERO,NO_AUTO_CREATE
475085 SELECT option_name, option_value FROM wp_options WHERE autoload
[...]
```

Logging all MariaDB queries with a BPF one-liner is easy!

Complex BPF scripts

A more complex BPF script (this time with BCC, a framework for BPF scripts in C):

- uprobe on `alloc_query`: obtain SQL statement
- uprobe on `do_command`: remember start timestamp
- return uprobe on `do_command`: calculate query duration

(no code here, doesn't fit on a slide - but it's public on GitHub)

Result:

```
duration query
     6ms 'SELECT post_id, meta_key, meta_value FROM wp_postmeta WHERE post_id IN [...]
     5ms 'SELECT post_id, meta_key, meta_value FROM wp_postmeta WHERE post_id IN [...]
     5ms 'SELECT post_id, meta_key, meta_value FROM wp_postmeta WHERE post_id IN [...]
     5ms "SELECT option_name, option_value FROM wp_options WHERE autoload = 'yes'"
  1478ms 'COMMIT;'
```

Emulating MariaDB's slow query log with BPF.

BPF with Disk Read

Adding more features to the BPF script:

- tracepoints on `raw_syscalls:sys_enter` and `raw_syscalls:sys_exit` to record time spent in system calls for each query
- tracepoints on `syscalls:sys_enter_read` and `syscalls:sys_exit_read` (and other read-related syscalls)

```
time         pid duration sys_t n_sys read_t read_b  query
08:29:08   69019    194ms 193ms     6   0ms      0  'COMMIT;'
08:29:09  123927     10ms   4ms    31   0ms   7218  "SELECT wp_posts.* FROM [...]
08:29:09  123927     10ms   4ms    36   0ms   3035  'SELECT post_id, [...]
08:29:09  128999     10ms   5ms    55   0ms   2859  "SELECT option_name, [...]
08:29:10  130388     13ms   4ms   121   0ms   3035  'SELECT wp_posts.* FROM [...]
08:29:10  124597     15ms  10ms   178   0ms   3035  'SELECT wp_posts.* FROM [...]
08:29:11  111227     10ms   5ms    40   0ms   2859  "SELECT option_name, [...]
```

A lot of time spent in system calls, but it's not `read`.

Which one, though?

BPF with Disk Write and Futex

Now with `write` and `futex`:

```
duration  sys_t n_sys futex_t  w_t  w_b query
   191ms  190ms     6   190ms  0ms    0 "UPDATE wp_posts SET guid=[...]
   177ms  176ms     5   176ms  0ms  246 "UPDATE wp_options SET option_value='0' W[...]
   144ms  143ms     6   143ms  0ms  246 "UPDATE wp_options SET option_value='0' W[...]
   142ms  141ms     6   141ms  0ms    0 "UPDATE wp_options SET option_value='0' W[...]
   186ms  183ms    14   183ms  0ms    0 "SELECT ID, post_name, post_parent, [...]
   122ms  121ms     8   121ms  0ms    0 "UPDATE wp_options SET option_value='0' W[...]
   144ms  142ms    13   142ms  0ms  992 "UPDATE wp_options SET option_value='0' W[...]
   118ms  117ms    11   117ms  0ms  496 "UPDATE wp_options SET option_value='0' W[...]
    61ms   60ms     9    60ms  0ms    0 "UPDATE wp_options SET option_value='0' W[...]
   134ms  132ms     6   132ms  0ms    0 "UPDATE wp_options SET option_value='0' W[...]
   185ms  184ms    10   184ms  0ms  246 "UPDATE wp_options SET option_value='0' W[...]
   104ms  103ms     8   103ms  0ms    0 "UPDATE wp_options SET option_value='0' W[...]
    30ms   29ms     9    28ms  0ms  246 "UPDATE wp_options SET option_value='0' W[...]
   148ms  146ms    15   146ms  0ms  496 "UPDATE wp_options SET option_value='0' W[...]
   171ms  170ms    11   170ms  0ms    0 "UPDATE wp_options SET option_value='0' W[...]
   194ms  192ms    14   192ms  0ms  496 "UPDATE wp_options SET option_value='0' W[...]
```

Writes are free, but futexes are what we're looking for!

Chasing Futexes

Caught some “real” slow queries after a few hours:

```
db           duration  sys_t futex_t sync_t query
HOSSSSSXFLPH   1417ms 1416ms  1416ms    0ms "UPDATE wp_options SET option_[...]
HOSSSSSXFMBY   1416ms 1415ms  1415ms    0ms "UPDATE wp_options SET option_[...]
HOSSSSSXFPJO   1422ms 1420ms  1420ms    0ms "UPDATE wp_options SET option_[...]
HOSSSSSXFCWV   1401ms 1401ms  1401ms    0ms "UPDATE wp_options SET option_[...]
HOSSSSSXFFII   1410ms 1408ms  1408ms    0ms "UPDATE wp_options SET option_[...]
HOSSSSSXWZTL   1406ms 1405ms  1405ms    0ms "UPDATE wp_options SET option_[...]
HOSSSSSXWYMM   1401ms 1400ms  1400ms    0ms "UPDATE wp_options SET option_[...]
HOSSSSSXJPPL   1358ms 1357ms  1357ms    0ms "UPDATE wp_options SET option_[...]
HOSSSSSXPFIQ   1344ms 1343ms  1343ms    0ms "UPDATE wp_options SET option_[...]
HOSSSSSXWKOW   1370ms 1368ms  1368ms    0ms "UPDATE wp_options SET option_[...]
HOSSSSSXWHBO   1346ms 1344ms  1344ms    0ms "UPDATE wp_options SET option_[...]
HOSSSSSXBRAR   1301ms 1300ms  1300ms    0ms "UPDATE wp_options SET option_[...]
HOSSSSSXBPCK   1436ms    1ms     0ms    8ms 'DROP INDEX [...] ON wp_yoast[...]
HOSSSSSXWWCY   1280ms 1278ms  1278ms    0ms "UPDATE wp_options SET option_[...]
```

All queries go to a different database.
All UPDATEs wait for a lock which apparently the DROP is holding.
Must be a global lock → MariaDB scalability issue?
Which lock is that?

Chasing one Futex

Now with futex address (BPF script prints the futex address with the longest wait for each query):

```
duration  sys_t futex_t    futex_addr query
  1270ms 1269ms  1269ms  564b6f4f4388 "UPDATE wp_options SET option_value='0' W[...]
  1270ms 1269ms  1269ms  564b6f4f4388 "UPDATE wp_options SET option_value='0' W[...]
  1280ms 1279ms  1279ms  564b6f4f4388 "UPDATE wp_options SET option_value='0' W[...]
  1260ms 1259ms  1259ms  564b6f4f4388 "UPDATE wp_options SET option_value='0' W[...]
  1426ms   35ms    27ms  564b6f4f4388 'DROP INDEX permalink_hash ON wp_yoast_[...]
  1242ms 1240ms  1240ms  564b6f4f4388 'SELECT 1 FROM wp_bookingpress_settings'
  1110ms 1108ms  1108ms  564b6f4f4388 "UPDATE wp_options SET option_value='0' W[...]
  1144ms 1143ms  1143ms  564b6f4f4388 "UPDATE wp_options SET option_value='0' W[...]
```

What is at address `0x564b6f4f4388`?

Catching the Futex

What is at address 0x564b6f4f4388?

```
perf trace -p $(pidof mariadbd) -e syscalls:sys_enter_futex --call-graph fp
[...]
mariadbd/174439 syscalls:sys_enter_futex(uaddr: 0x564b6f4f4388, op: WAIT|[...]
  syscall (/usr/lib/x86_64-linux-gnu/libc.so.6)
  ssux_lock_impl<false>::rd_wait() (/usr/sbin/mariadbd)
  ha_innobase::referenced_by_foreign_key() (/usr/sbin/mariadbd)
  DML_prelocking_strategy::handle_table(THD*, Query_tables_list*, TABLE_LIST*, [...]
  extend_table_list(THD*, TABLE_LIST*, Prelocking_strategy*, bool)
  open_tables(THD*, DDL_options_st const&, TABLE_LIST**, unsigned int*, [...]
  mysql_update(THD*, TABLE_LIST*, List<Item>&, List<Item>&, Item*, unsigned int, [...]
  mysql_execute_command(THD*, bool) (/usr/sbin/mariadbd)
  mysql_parse(THD*, char*, unsigned int, Parser_state*) (/usr/sbin/mariadbd)
  dispatch_command(enum_server_command, THD*, char*, unsigned int, bool)
  do_command(THD*, bool) (/usr/sbin/mariadbd)
  do_handle_one_connection(CONNECT*, bool) (/usr/sbin/mariadbd)
```

https://github.com/MariaDB/server/blob/main/storage/innobase/handler/ha_innodb.cc#L15645

```
uint ha_innobase::referenced_by_foreign_key()
{
  dict_sys.freeze(SRW_LOCK_CALL);
  const bool empty= m_prebuilt->table->referenced_set.empty();
  dict_sys.unfreeze();
  return !empty;
}
```

That's dict_sys.latch!

dict_sys!

`dict_sys.latch`?

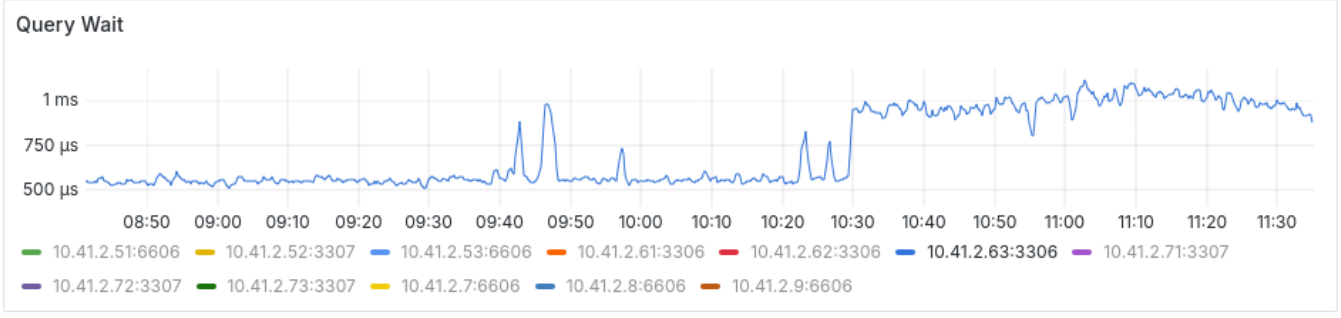
Candidate: `https://jira.mariadb.org/browse/MDEV-32899`
"InnoDB is holding shared dict_sys.latch while waiting for FOREIGN KEY child table lock on DDL"

Fixed in MariaDB 10.11.8. Debian Bookworm has 10.11.6. Duh!

Summary

- BPF is a superpower! Trace/analyze kernel & processes on production servers at any time (without recompiling)
- Frameworks like `bpftrace` and BCC allow quick script development.
- BPF is not easy. Requires some experience with C and it helps if you know the kernel.
- Tracing C++ is harder (symbol mangling, vtables).

Non-negligible overhead (adds 400µs to queries):



PR with my BPF script: github.com/iovisor/bcc/pull/5108

That's all for today. Questions?