More on bpftrace for MariaDB DBAs and Developers

Valerii Kravchuk, Principal Support Engineer, MariaDB

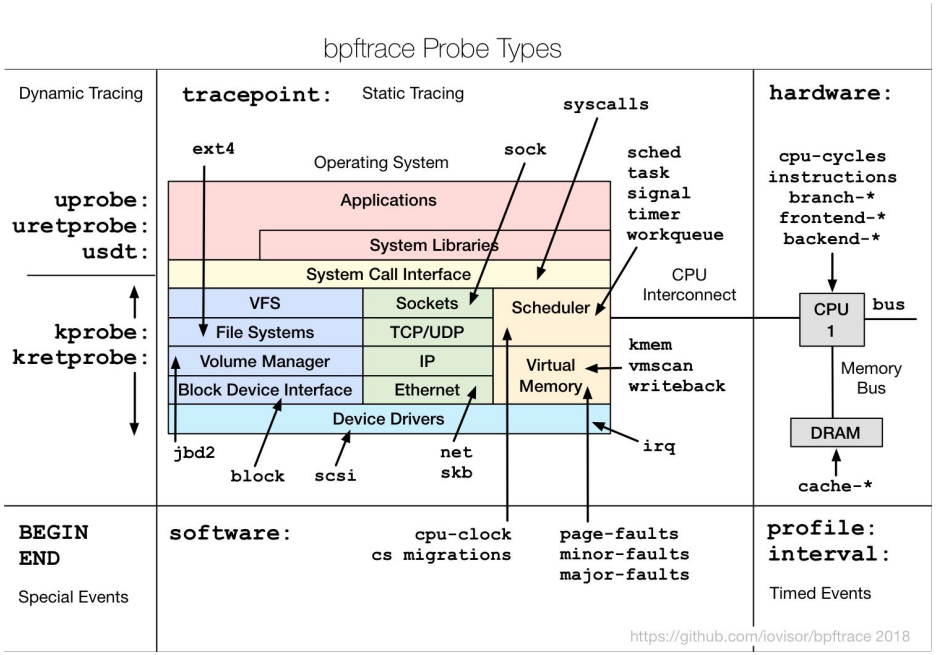
valerii.kravchuk@mariadb.com

What is this session about?

- Some links to other talks and resources on eBPF-based **bpftrace** tool
- Building recent **bpftrace** from source
- **bpftrace** as a MariaDB server code coverage testing tool (pushed to the limits)
- Advanced usage examples applied to MariaDB server:
  - Adding dynamic probe to some line inside the function with **bpftrace** (possible with probe on offset within function
  - Access to complex structures typical for MariaDB server (**bpftrace** needs headers)

bpftrace as a frontend for eBPF

- **bpftrace** (frontend with pattern/action based programming language resembling **awk** and **dtrace**) allows to define actions for probes presented below in easy and flexible way
- To use **bpftrace** you need recent enough kernel 5.x.y. Install the package or build it from GitHub source and then you can do many cool things…



Versions of bpftrace (on my Ubuntu 20.04)

```
openxs@ao756:~$ dpkg -l | grep bpftrace
ii  bpftrace                                    0.9.4-1
    amd64        high-level tracing language for Linux eBPF
openxs@ao756:~$ bpftrace --version
bpftrace v0.13.0-120-gc671
openxs@ao756:~$ cd git/bpftrace/
openxs@ao756:~/git/bpftrace$ git log -1
commit 422847b71dc38d31d9b352b0058196bbe5a9e278 (HEAD -> master,
origin/master, origin/HEAD)
Author: Viktor Malik <viktor.malik@gmail.com>
Date:   Mon Jan 10 16:41:05 2022 +0100

    Fix LLVM 13 warnings

    Since LLVM 13, CreateGEP and CreateLoad require explicit types.
openxs@ao756:~/git/bpftrace$ build/src/bpftrace --version
bpftrace v0.14.0-50-g4228
```

- You may have to build from source (or use Docker images)

Building bpftrace from current GitHub source

- GitHub instructions are not complete and may not work “as is”
- I’ve already described steps in case of Fedora here and there
- You may have to (and should) start with building current **bcc** tools from GitHub
- Make sure to update submodules for both **bcc** and **bpftrace**:
  `git submodule update --init --recursive`
- Build out of source and run tests before installing
- You may have to patch the source code depending on LLVM and CLang versions used, to fix new or known open issues
- Eventually I managed to get it built and working, doable
- Detailed blog post based on recent experience is not yet ready

Building bpftrace on Ubuntu 20.04 - LLVM Hell

- While trying to build recent versions I ended up with LLVM hell:

```
openxs@ao756:~/git/bpftrace/build$ dpkg -l | grep llvm
ii  libllvm10:amd64                          1:10.0.0-4ubuntu1                amd64
Modular compiler and toolchain technologies, runtime library
…
ii  llvm                                     1:10.0-50~exp1                   amd64
Low-Level Virtual Machine (LLVM)
ii  llvm-10                                  1:10.0.0-4ubuntu1                amd64
Modular compiler and toolchain technologies
…
ii  llvm-7                                   1:7.0.1-12                       amd64
Modular compiler and toolchain technologies
ii  llvm-7-dev                               1:7.0.1-12                       amd64
Modular compiler and toolchain technologies, libraries and headers
ii  llvm-7-runtime                           1:7.0.1-12                       amd64
Modular compiler and toolchain technologies, IR interpreter
ii  llvm-dev                                 1:10.0-50~exp1                   amd64
Low-Level Virtual Machine (LLVM), libraries and headers
…
```

- Adding **-DLLVM_REQUESTED_VERSION=10** to **cmake** command line helped
- Dirty hacks on top (**vi ./src/CMakeFiles/bpftrace.dir/link.txt**)

bpftrace as a code coverage testing tool (lame…)

- What if we try to add **uprobe** for every function in MariaDB server?

```
openxs@ao756:~$ sudo bpftrace -e
'uprobe:/home/openxs/dbs/maria10.6/bin/mariadbd:* { printf("%s\n", func);
}'
ERROR: Can't attach to 34765 probes because it exceeds the current limit
of 512 probes.
You can increase the limit through the BPFTRACE_MAX_PROBES environment
variable, but BE CAREFUL since a high number of probes attached can cause
your system to crash.
openxs@ao756:~$ sudo BPFTRACE_MAX_PROBES=40000 bpftrace -e
'uprobe:/home/openxs/dbs/maria10.6/bin/mariadbd:* { printf("%s\n", func);
}'
Attaching 34765 probes...
bpf: Failed to load program: Too many open files
processed 17 insns (limit 1000000) max_states_per_insn 0 total_states 1
peak_states 1 mark_read 0
...
ERROR: Error loading program:
uprobe:/home/openxs/dbs/maria10.6/bin/mariadbd:thd_client_ip (try -v)
Segmentation fault


...
```

bpftrace as a code coverage testing tool (PoC)

- What if we try try to trace less functions in MariaDB server? It works (blog post)

```
openxs@ao756:~$ sudo BPFTRACE_MAX_PROBES=40000 bpftrace -e
'uprobe:/home/openxs/dbs/maria10.6/bin/mariadbd:*do* { printf("%s\n", func);
}'
Attaching 1076 probes...
ERROR: Offset outside the function bounds ('__do_global_dtors_aux' size is
0)
openxs@ao756:~$ sudo bpftrace -e
'uprobe:/home/openxs/dbs/maria10.6/bin/mariadbd:*command* { @cnt[func] += 1;
}'
Attaching 70 probes...
^C

@cnt[is_log_table_write_query(enum_sql_command)]: 2
@cnt[mysql_execute_command(THD*, bool)]: 4
...
@cnt[general_log_print(THD*, enum_server_command, char const*, ...)]: 6
@cnt[Opt_trace_start::Opt_trace_start(THD*, TABLE_LIST*, enum_sql_command,
List<set_var_base>*, char const*, unsigned long, charset_info_st const*)]: 8
@cnt[dispatch_command(enum_server_command, THD*, char*, unsigned int,
bool)]: 9
```

Developers (and some DBAs) use gdb to work with MariaDB server code

- Because it's really easy to get all the details. Consider this breakpoint on the InnoDB function to set index record lock:

```
Thread 11 "mariadbd" hit Breakpoint 2,  lock_rec_lock (impl=false, mode=2,
    block=0x7f12480325a0, heap_no=2, index=0x7f1218066f90,
thr=0x7f121806cc10)
    at /home/openxs/git/server/storage/innobase/include/que0que.ic:37
37              return(thr->graph->trx);
(gdb) p index
$3 = (dict_index_t *) 0x7f1218066f90
(gdb) p index->name
$4 = {m_name = 0x7f1218067120 "PRIMARY"}
(gdb) p index->table->name
$5 = {m_name = 0x7f1218020908 "test/tt", static part_suffix = "#P#"}
(gdb) p index->table
$6 = (dict_table_t *) 0x7f1218065b20
```

- Can we access structured arguments in **bpftrace** programs?

Accessing complex structures in MariaDB server code with bpftrace - problem

- We need a way to define structures in **bpftrace** programs
- Including MariaDB headers is NOT an option in general:

```
openxs@ao756:~/git/bpftrace/build$ cat /tmp/test.bt
#include "dict0mem.h"
END {
  printf("\nIncluded headers...\n");
}
openxs@ao756:~/git/bpftrace/build$ sudo src/bpftrace -I
/home/openxs/git/server/storage/innobase/include /tmp/test.bt
/home/openxs/git/server/storage/innobase/include/univ.i:68:10: fatal
error: 'my_global.h' file not found
openxs@ao756:~/git/bpftrace/build$ cat /tmp/struct.bt
struct table_name_t
{
    char*   m_name;
    static const char part_suffix[4];
}
END { printf("Success\n"); }
openxs@ao756:~/git/bpftrace/build$ sudo src/bpftrace /tmp/struct.bt
definitions.h:5:9: error: type name does not allow storage class to be
specified
```

Accessing complex structures in MariaDB server code with bpftrace - solution

- Define only parts needed to dereference fields you need:

```
struct table_name_t {
    char*   m_name;
    char part_suffix[4]; }
struct dict_table_t {
    long long             id;
    struct dict_table_t*  id_hash;
    struct table_name_t   name; }
struct dict_index_t {
    long long         id;
    long long         heap;
    char*             name;
    struct dict_table_t*  table; }
…
uprobe:/home/openxs/dbs/maria10.6/bin/mariadbd:lock_rec_lock
  { printf("lock_rec_lock: impl (%d) mode %d index %s rec of %s,
thread: %p\n", arg0,  arg1,
    str(((struct dict_index_t *)arg4)->name),
    str(((struct dict_index_t *)arg4)->table->name.m_name),
    arg5); }
```

- See my recent blog post for more details

Tracing of class member functions with bpftrace

- Straightforward approach does not work:

```
openxs@ao756:~/git/bpftrace/build$ sudo src/bpftrace -l
'uprobe:/home/openxs/dbs/maria10.6/bin/mariadbd:ha_heap::records_in_range'
stdin:1:1-73: ERROR: uprobe probe type requires 2 arguments
```

- Use mangled names

```
openxs@ao756:~/git/bpftrace/build$ objdump -tT /home/openxs/dbs/maria10.6/bin/mariadbd
| grep 'records_in_range' | grep ha_heap
0000000000cbb960 l  F .text  000000000000008b
_ZN7ha_heap16records_in_rangeEjPK12st_key_rangeS2_P13st_page_range
openxs@ao756:~/git/bpftrace/build$ sudo src/bpftrace -e
'uretprobe:/home/openxs/dbs/maria10.6/bin/mariadbd:_ZN7ha_heap16records_in_rangeEjPK12
st_key_rangeS2_P13st_page_range { printf("Records in range: %d\n", retval); }'
Attaching 1 probe...
Records in range: 1
^C
```

- See how ti was done with **perf** here. Probe may be added by address:

```
openxs@ao756:~/git$ bpftrace -e
'uretprobe:/home/openxs/dbs/maria10.6/bin/mariadbd:0xcbb961 { printf("Records in
range: %d\n", retval); }'
Attaching 1 probe...
ERROR: uretprobes cannot be attached at function offset. (address resolved to:
_ZN7ha_heap16records_in_rangeEjPK12st_key_rangeS2_P13st_page_range+1)
```

Adding probe to some line inside the function, 1

- It's possible, just add probe on offset within function. How to get the offset?
- Read disassembled code and find the offset of the command:

```
(gdb) disassemble do_command
Dump of assembler code for function do_command(THD*, bool):
   0x000055ef5c31d2c0 <+0>:   endbr64
…
   0x000055ef5c31d35f <+159>:   mov    0x2b0(%rbx),%r10
   0x000055ef5c31d366 <+166>:   test   %rax,%rax
   0x000055ef5c31d369 <+169>:   je     0x55ef5c31d430 <do_command(THD*,
bool)+368>
   0x000055ef5c31d36f <+175>:   movb   $0x0,(%r10,%r12,1)
```

- Convert offset (**175**) to hex (**0xaf**) and add to start address to get **0x79036f**:

```
openxs@ao756:~/git/bpftrace/build$ objdump -tT
/home/openxs/dbs/maria10.6/bin/mariadbd | grep do_command
0000000007902c0 g    F .text  00000000000007e9
_Z10do_commandP3THDb
0000000007902c0 g    DF .text  00000000000007e9  Base
_Z10do_commandP3THDb
```

Adding probe to some line inside the function, 2

- Cheat with **perf –line** like this:

```
openxs@ao756:~/dbs/maria10.6$ sudo perf probe -x
/home/openxs/dbs/maria10.6/bin/mysqld --line do_command
<do_command@/home/openxs/git/server/sql/sql_parse.cc:0>
     0    dispatch_command_return return_value;
…
     120  command= fetch_command(thd, packet)
```

- We know that we can add probe at line **120** of **do_command()**. This way:

```
openxs@ao756:~/git/bpftrace/build$ sudo perf probe -x
/home/openxs/dbs/maria10.6/bin/mariadbd 'do_command:120 packet:string'
```

- Now let's check wihat was added:

```
openxs@ao756:~/git/bpftrace/build$ sudo cat
/sys/kernel/tracing/uprobe_events
p:probe_mariadbd/do_command_L120
/home/openxs/dbs/maria10.6/bin/mariadbd:0x0000000000 79036f
packet_string=+0(%r10):string
```

- **0x79036f** - **0x7902c0 = 0xaf = 175** (decimal)

Accessing local variables with bpftrace

- Time to put it all together, **bpftrace** probe on the line inside the function that prints the value of the local variable
- We need to know where is the value, in out case it's in the **r10** register:

```
openxs@ao756:~/git/bpftrace/build$ sudo src/bpftrace -e
'uprobe:/home/openxs/dbs/maria10.6/bin/mariadbd:0x0000000000790
36f { printf("Function: %s packet: %s\n", func,
str(reg("r10"))); }'
Attaching 1 probe...
Function: do_command(THD*, bool) packet: select 1+1
Function: do_command(THD*, bool) packet: select count(*) from
user
Function: do_command(THD*, bool) packet: select count(*) from
mysql.user
^C
```

- Yes, you can access registers in **bpftrace**, with **reg()** function
- Maybe I am crazy to suggest this to DBAs, but developers should be happy!

Thank you!

Thanks to **MariaDB Foundation** for this Devroom@FOSDEM

More blog posts on **bpftrace** are coming…

Please, search and report bugs at:

- https://jira.mariadb.org
- https://github.com/iovisor/bpftrace/issues

Questions and Answers?