ClickHouse

Hotpatching ClickHouse in production with XRay

Pablo Marcos Oltra

FOSDEM 2026

$ whoami

- Pablo Marcos Oltra
- Software Engineer @ ClickHouse's Core Team
- Systems Developer
- Passionate about videogame development
- Interested in compilers, build systems and new programming languages

ClickHouse

Who knows what hotpatching/hot reloading is?

ClickHouse

Who knows what LLVM's XRay is?

ClickHouse

Who's had to debug something in production and wished they had added an extra log to know what's happening?

ClickHouse

01 What is hotpatching

Hotpatching: The basics

- Change the code while the process is running
- Allows faster iteration
- Often done in videogame development
  - Hot reloading rather than hotpatching
  - Client-server architecture where the lightweight client loads a new version of the game (in a shared library)
  - Examples
    - http://runtimecompiledcplusplus.blogspot.com.es
    - https://fungos.github.io/cr-simple-c-hot-reload
    - https://liveplusplus.tech

Hotpatching: Adding instrumentation

- -finstrument-functions : Generates instrumentation code for every function entry/exit

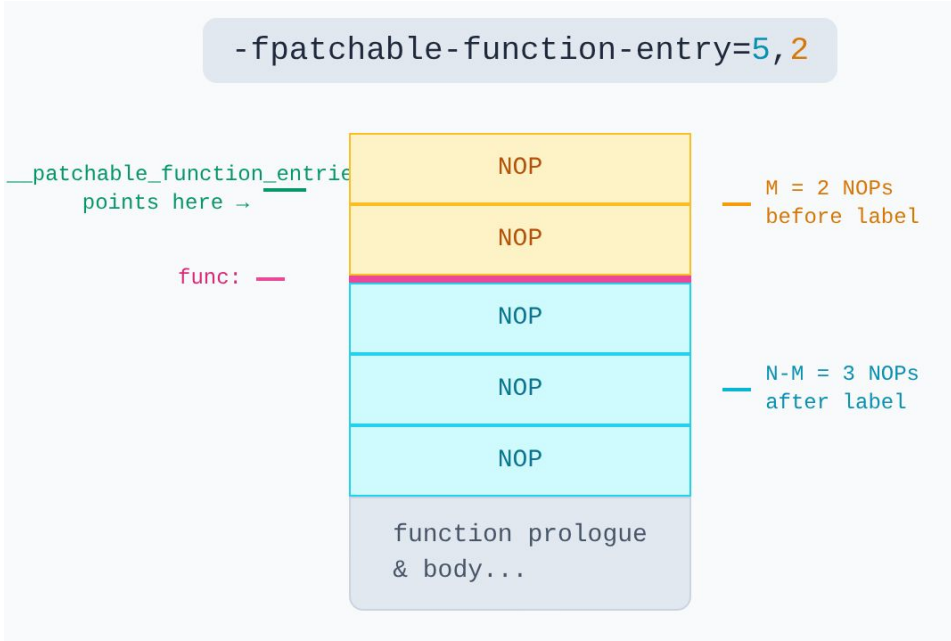
```
void __cyg_profile_func_enter(void *this_fn, void *call_site);
void __cyg_profile_func_exit(void *this_fn, void *call_site);
```

https://gcc.gnu.org/onlinedocs/gcc/Instrumentation-Options.html#index-finstrument-functions

Hotpatching: Adding instrumentation

- -fpatchable-function-entry=N[,M]:

  Generate M NOPs before function entry and N-M NOPs after function entry



https://gcc.gnu.org/onlinedocs/gcc/Instrumentation-Options.html#index-fpatchable-function-entry

ClickHouse

02 XRay

XRay

XRay consists of three main parts:

1. Compiler-inserted instrumentation points.
2. A runtime library for enabling/disabling tracing at runtime.
3. A suite of tools for analysing the traces.

As of July 25, 2018, XRay is only available for the following architectures running Linux:

x86_64, arm7 (no thumb), aarch64, powerpc64le, mips, mipsel, mips64, mips64el, NetBSD: x86_64, FreeBSD: x86_64 and OpenBSD: x86_64.

https://llvm.org/docs/XRay.html

XRay: Example

```
15  int main()
16  {
17      __xray_init();
18      __xray_patch();
19      __xray_set_handler(my_handler);
20
21      foo();
22
23      __xray_remove_handler();
24      __xray_unpatch();
25
26      return 0;
27  }
```

XRay: Example

```
#include <stdint.h>
#include <stdio.h>
#include <xray/xray_interface.h>

[[clang::xray_never_instrument]] void my_handler(int32_t fid, XRayEntryType type)
{
    printf("FuncID: %d, Type: %d\n", fid, type);
}

void foo()
{
    printf("Foo!\n");
}
```

XRay: Example

```
$ clang++ -o test_xray test_xray.cpp -fxray-instrument
-fxray-instruction-threshold=1


$ ./test_xray
FuncID: 1, Type: 0
Foo!
FuncID: 1, Type: 1
```

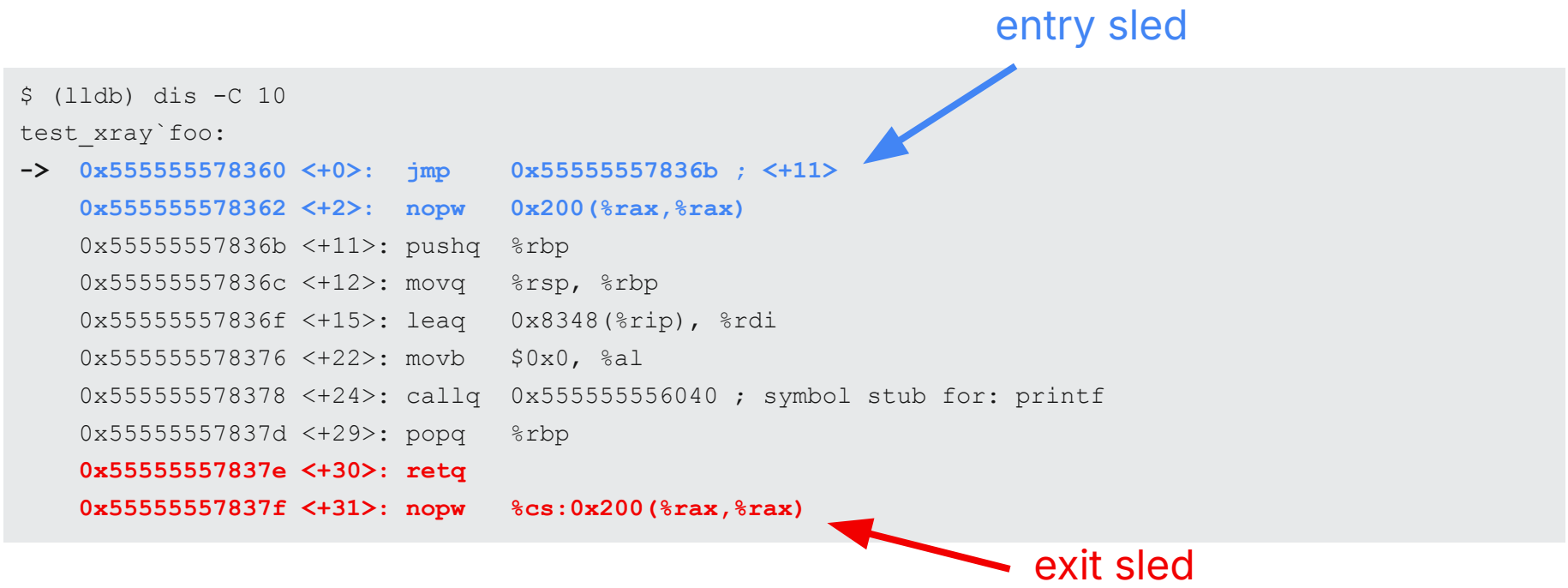
XRay: xray_instr_map

```
$ objdump -h -j xray_instr_map test_xray

test_xray:     file format elf64-x86-64

Sections:
Idx Name          Size      VMA               LMA               File off  Algn
 17 xray_instr_map 00000080  000000000002c6c4  000000000002c6c4  0002c6c4
2**0
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
```

XRay: Before patching foo



ClickHouse

XRay: After patching foo

```
$ (lldb) dis -C 10
test_xray`foo:
->  0x555555578370 <+0>:  movl   $0x1, %r10d
    0x555555578376 <+6>:  callq  0x555555574340 ; __xray_FunctionEntry
    0x55555557837b <+11>: pushq  %rbp
    0x55555557837c <+12>: movq   %rsp, %rbp
    0x55555557837f <+15>: leaq   0x8338(%rip), %rdi
    0x555555578386 <+22>: movb   $0x0, %al
    0x555555578388 <+24>: callq  0x555555556040 ; symbol stub for: printf
    0x55555557838d <+29>: popq   %rbp
    0x55555557838e <+30>: movl   $0x1, %r10d
    0x555555578394 <+36>: jmp    0x555555574480 ; __xray_FunctionExit
```

ClickHouse

03 Integration into ClickHouse

What’s ClickHouse

- Fast OLAP DBMS
- Started in 2009 at Yandex.Metrica by Alexey Milovidov
- Open sourced in 2016 under Apache 2.0 license
- Developed in C++
- Strong focus in performance
- Lots of dogfooding
    - CI results, observability, pastila, c.house, etc

https://clickhouse.com

https://benchmark.clickhouse.com

Integrating XRay into ClickHouse

- Use XRay's instrumentation to have negligible overhead in production when disabled
- Use XRay's runtime library to enable in production only the symbols we want to instrument
- Use SQL statements to enable/disable instrumentation points at runtime
- Kudos to our intern Alina Badakhova for doing the POC to prove it was feasible

https://github.com/ClickHouse/ClickHouse/issues/74249

Integrating XRay into ClickHouse

- Landed in ClickHouse 25.12 (December 25)
- Minor improvements since then, all backported

https://github.com/ClickHouse/ClickHouse/pull/89173

Hotpatching using SQL: Cheatsheet

```
-- Check symbols to add instrumentation points
SELECT * FROM system.symbols WHERE function_id IS NOT NULL

-- Add or remove instrumentation points
SYSTEM INSTRUMENT [ADD|REMOVE] 'SYMBOL' HANDLER [ENTRY|EXIT] [PARAMS]

-- Check the enabled instrumentation points
SELECT * FROM system.instrumentation

-- Check the instrumented points hit
SELECT * FROM system.trace_log WHERE trace_type = 'Instrumentation'
```

Hotpatching using SQL: Inspecting symbols

```
SET allow_introspection_functions=1

-- Check symbols to add instrumentation points
SELECT
    function_id, symbol_demangled
FROM
    system.symbols
WHERE
    symbol_demangled ILIKE '%QueryMetricLog::start%'
```

Hotpatching using SQL: Inspecting symbols

```
Row 1:
──────

function_id:      202915

symbol_demangled: DB::QueryMetricLog::startQuery(std::__1::basic_string<char,
std::__1::char_traits<char>, std::__1::allocator<char>> const&,
std::__1::chrono::time_point<std::__1::chrono::system_clock,
std::__1::chrono::duration<long long, std::__1::ratio<1l, 10000001>>>, unsigned long)


Row 2:
──────

function_id:      202933

symbol_demangled: void std::__1::__function::__policy_func<void
()>::__call_func[abi:ne210105]<DB::QueryMetricLog::startQuery(std::__1::basic_string<char,
std::__1::char_traits<char>, std::__1::allocator<char>> const&,
std::__1::chrono::time_point<std::__1::chrono::system_clock, std::__1::chrono::duration<long
long, std::__1::ratio<1l, 10000001>>>, unsigned
long)::$_0>(std::__1::__function::__policy_storage const*)
```

Hotpatching using SQL: Adding instrumentation

```
-- Add a log with stacktrace at function entry
SYSTEM INSTRUMENT ADD 'QueryMetricLog::startQuery'  LOG ENTRY 'FOSDEM 26';

-- Add a random sleep [0, 0.5] secs at function exit
SYSTEM INSTRUMENT ADD 'QueryMetricLog::startQuery'  SLEEP EXIT 0 0.5;

-- Profile deterministically
SYSTEM INSTRUMENT ADD 'QueryMetricLog::startQuery'  PROFILE;
```

Hotpatching using SQL: Adding instrumentation

```
SELECT * FROM system.instrumentation


Row 1:
──────
id:             0
function_id:    202915
function_name:  QueryMetricLog::startQuery
handler:        log
entry_type:     Entry
symbol:         DB::QueryMetricLog::startQuery(std::__1::basic_string<char,
std::__1::char_traits<char>, std::__1::allocator<char>> const&,
std::__1::chrono::time_point<std::__1::chrono::system_clock,
std::__1::chrono::duration<long long, std::__1::ratio<1l, 1000000l>>>,
unsigned long)
parameters:     ['FOSDEM 26']
```


ClickHouse

Hotpatching using SQL: Adding instrumentation

```
Row 2:
──────
id:              1
function_id:     202915
function_name:   QueryMetricLog::startQuery
handler:         sleep
entry_type:      Exit
symbol:
DB::QueryMetricLog::startQuery(std::__1::basic_string<char,
std::__1::char_traits<char>, std::__1::allocator<char>> const&,
std::__1::chrono::time_point<std::__1::chrono::system_clock,
std::__1::chrono::duration<long long, std::__1::ratio<1l,
1000000l>>>, unsigned long)
parameters:      [0,0.5]
```

Hotpatching using SQL: Adding instrumentation

```
Row 3:
──────
id:             2
function_id:    202915
function_name:  QueryMetricLog::startQuery
handler:        profile
entry_type:     EntryAndExit
symbol:
DB::QueryMetricLog::startQuery(std::__1::basic_string<char,
std::__1::char_traits<char>, std::__1::allocator<char>> const&,
std::__1::chrono::time_point<std::__1::chrono::system_clock,
std::__1::chrono::duration<long long, std::__1::ratio<1l,
1000000l>>>, unsigned long)
parameters:     []
```

Hotpatching using SQL: Removing instrumentation

```
-- Remove all instrumented points
SYSTEM INSTRUMENT REMOVE ALL;

-- Remove a specific instrumented point with id 2
SYSTEM INSTRUMENT REMOVE 2;

-- Remove all entry points for function 'QueryMetricLog::startQuery'
SYSTEM INSTRUMENT REMOVE 'QueryMetricLog::startQuery';
```

Hotpatching using SQL: Checking trace_log

```
SELECT
    event_time_microseconds,
    function_id,
    function_name,
    handler,
    entry_type,
    duration_nanoseconds
FROM system.trace_log
WHERE trace_type = 'Instrumentation'
```

Hotpatching using SQL: Checking trace_log

```
Row 1:
──────
event_time_microseconds: 2026-01-29 11:55:20.553718
function_id:             202915
function_name:
DB::QueryMetricLog::startQuery(std::__1::basic_string<char,
std::__1::char_traits<char>, std::__1::allocator<char>> const&,
std::__1::chrono::time_point<std::__1::chrono::system_clock,
std::__1::chrono::duration<long long, std::__1::ratio<1l,
1000000l>>>, unsigned long)
handler:                 profile
entry_type:              Entry
duration_nanoseconds:    □□□□
```

Hotpatching using SQL: Checking trace_log

```
Row 2:
──────
event_time_microseconds: 2026-01-29 11:55:20.553758
function_id:             202915
function_name:
DB::QueryMetricLog::startQuery(std::__1::basic_string<char,
std::__1::char_traits<char>, std::__1::allocator<char>> const&,
std::__1::chrono::time_point<std::__1::chrono::system_clock,
std::__1::chrono::duration<long long, std::__1::ratio<1l,
1000000l>>>, unsigned long)
handler:                 profile
entry_type:              Exit
duration_nanoseconds:    40059
```

Hotpatching using SQL: Visualizing the profile

- Use Chrome's Trace Event Format with different visualizers

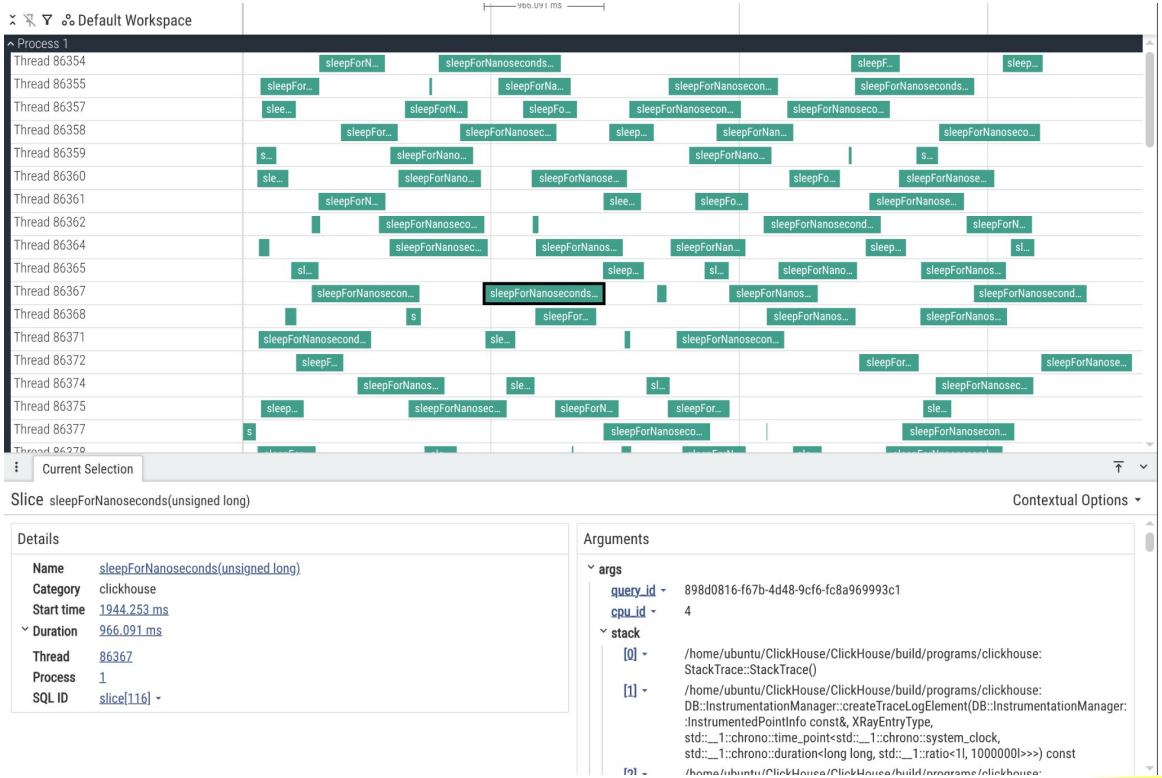
```
[
  {"name": "Asub", "cat": "PERF", "ph": "B", "pid": 22630,
"tid": 22630, "ts": 829},
  {"name": "Asub", "cat": "PERF", "ph": "E", "pid": 22630,
"tid": 22630, "ts": 833}
]
```

https://docs.google.com/document/d/1CvAClvFfyA5R-PhYUmn5OOQtYMH4h6I0nSsKchNAySU/preview?tab=t.0

Hotpatching using SQL: Converting the data

```
WITH traces AS (
    SELECT * FROM system.trace_log
    WHERE event_date >= today() AND trace_type = 'Instrumentation' AND handler = 'profile'
    ORDER BY event_time, entry_type
)
SELECT
    format(
        '{{"traceEvents": [{}\n]}}',
        arrayStringConcat(
            groupArray(
                format(
                    '\n{{"name": "{}", "cat": "clickhouse", "ph": "{}", "ts": {}, "pid": 1, "tid": {}, "args": {{"query_id": "{}", "cpu_id": {},
"stack": [{}]}}}},',
                    function_name, if(entry_type = 0, 'B', 'E'), timestamp_ns/1000,
                    toString(thread_id), query_id, cpu_id,
                    arrayStringConcat(arrayMap((x, y) -> concat('"', x, ': ', y, '"'), lines, symbols))
                )
            )
        )
    )
FROM traces;
```

Hotpatching using SQL: Visualizing in Perfetto



Future work

- Add more handlers as we see the need for them
- Add some VM to allow scripting? Wren, Lua
- Educate others that this exists so that they use it. Collect their feedback

Caveats

- Can’t mix XRay and sanitized builds due to symbols clashing in libclang_rt

```
ld.lld-19: error: duplicate symbol: __sanitizer::internal_allocator()
>>> defined at
sanitizer_allocator.cpp.o:(__sanitizer::internal_allocator()) in
archive
/usr/lib/llvm-19/lib/clang/19/lib/linux/libclang_rt.ubsan_standalone-
x86_64.a
>>> defined at
sanitizer_allocator.cpp.o:(.text._ZN11__sanitizer18internal_allocator
Ev+0x0) in archive
/usr/lib/llvm-19/lib/clang/19/lib/linux/libclang_rt.xray-x86_64.a
```

- If you mess something with this, you do it big time cause it’s in production
- Profiling a function doesn’t profile all functions underneath. That’s very difficult

ClickHouse

Connect with ClickHouse

slack

ClickHouse
Community Dinner

Try ClickHouse for your use case

- ClickHouse Cloud
- Download open source

Learn

- Academy / certifications
- Blogs / YouTube

Engage with our community

- Community Slack
- Monthly Community calls
- Meetups / events

We are Hiring. Come Work with Us!

ClickHouse

Q&A

All links from the slides

https://c.house/links