又拍云 OpenResty/Nginx 服务优化实践

OpenResty Con 2018

张超 (Alex Zhang)

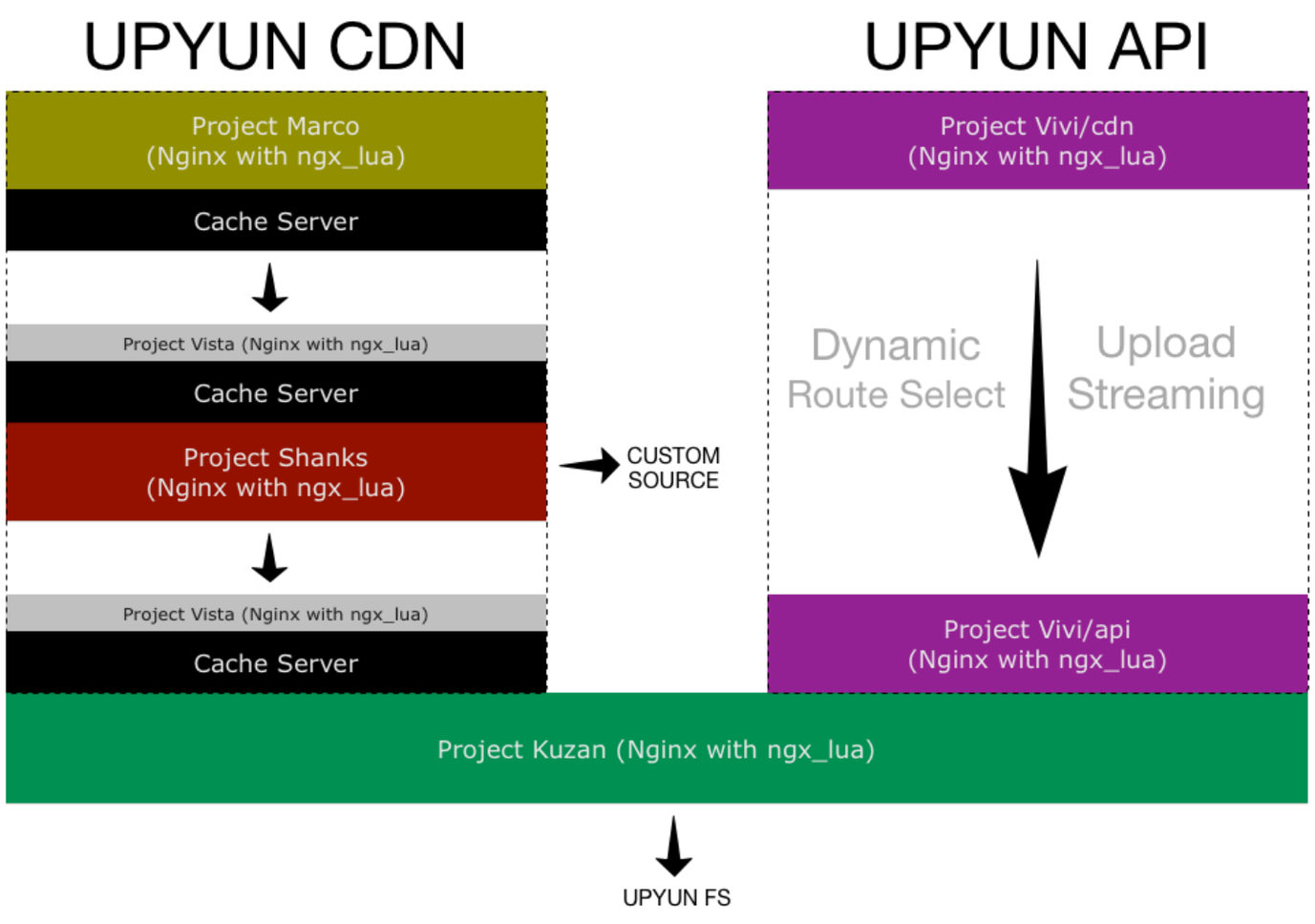
System Development Engineer at UPYUN Inc.

☆ Email: zchao1995@gmail.com

☆ Github: https://github.com/tokers

OpenResty in UPYUN

- Use OpenResty since 2013.
- All CDN services (except the Cache Server) are based on OpenResty.
- The gateway of UPYUN Storage System is based on OpenResty.
- All API services use Kong as the gateway.



https://github.com/upyun/upyun-resty

- Many lua-resty libraries
- Many Nginx C modules
- Reports and fixes some bugs about OpenResty and Nginx.

So we have entered a stage where we need to do the deep optimization for our services.

Performance Analysis & Tips

Name Resolution Management

SSL Acceleration in Practice

Resources analysis

- top
- pidstat
- vmstat
- iostat
- sar
- ……

Resource analysis focuses on resources’ utilization (U), saturation (S) and errors (E).

Workload analysis

- perf
- systemtap
- bcc/ebpf
- flamegraph
- ……

Workload analysis focuses on application’s own performance, like latency, bad status (5xx) ratio.

But sometimes tool chain is incomplete/absent in the production environment.

So build a docker image with:

- compilation/link tools
- perf
- systemtap
- flamegraph tools
- gdb
- mozilla rr
- valgrind
- …

💡 Performance diagnosis in a privileged docker container.

```
docker run --it --rm --privileged \
-v /lib/modules:/lib/modules:ro \
-v /etc/localtime:/etc/localtime:ro \
-v /usr/src:/usr/src:ro \
--net=host \
--pid=host upyun_stap:latest
```

ngx.ctx & ngx.var

Both ngx.ctx and ngx.var can be used to store request-specific data.

ngx.ctx is a Lua table while ngx.var is an interface to get/set Nginx's variables.

ngx.ctx is better

- Lua table is fast
- result type of ngx.var.* can only be string
- hash calculations, memory allocations inside ngx.var.* calls

💡 Cache ngx.ctx to avoid the expensive meta-method calls.

```
local function f()
    local ctx = ngx.ctx
    ctx.foo = “bar”
    ctx.bar = “foo”
    ctx.done = true
end
```

The life cycle of ngx.ctx is limited to a Nginx location.

Data will be lost when Nginx internal redirect happens.

💡 Use lua-resty-ctxdump to bypass this limitation.

```
location /t1 {
    set $ctx_ref "";
    content_by_lua_block {
        local ctxdump = require "resty.ctxdump"
        ngx.ctx = {
            Date = "Wed May  3 15:18:04 CST 2017",
        }
        ngx.var.ctx_ref = ctxdump.stash_ngx_ctx()
        ngx.exec("/t2")
    }
}
location /t2 {
    internal;
    content_by_lua_block {
        local ctxdump = require "resty.ctxdump"
        ngx.ctx = ctxdump.apply_ngx_ctx(ngx.var.ctx_ref)
    }
}
```

HTTP headers

```
local user_agent = ngx.req.get_headers()[“User-Agent”]
local cookie = ngx.req.get_headers()[“Cookie”]
local x_forwarded_for = ngx.req.get_headers()[“X-Fordwarded-For”]
```

ngx.req.get_headers() will return all the request headers. Keys (names) are all lowercase.

💡 Use lowercase name to index header value.

```
local req_headers = ngx.req.get_headers()
setmetatable(req_headers, nil)
local user_agent = req_headers[“user-agent”]
local cookie = req_headers[“cookie”]
local x_forwarded_for = req_headers[“x-forwarded-for”]
```

access log

```
http {
    access_log logs/error.log main buffer=4096;
}
```

- influenced by disk
  - too many `write()` system calls (if buffer too small)

💡 flush logs to external services with `lua-resty-logger-socket`!

Good Programming Habits

- 💡 avoid overusing global variables
- 💡 avoid inefficient string concatenations
- 💡 avoid too much table resize
- 💡 use lua-resty-core
- 💡 use JIT-compiled functions
- 💡 use ffi to call your C functions

http://wiki.luajit.org/NYI
https://blog.codingnow.com/cloud/LuaTips

Name Resolution

- misunderstanding about Nginx DNS resolution
- Nginx runtime DNS resolver's drawbacks
- Good solution based on ngx_lua

```
upstream backend {
    server upyun.com weight=10 max_fails=30 fail_timeout=30s;
}

server {
    listen *:8080;
    location / {
        proxy_pass http://backend;
    }
}
```

upyun.com will be resolved only once (while parsing the server directive).

```
server {
    listen *:8080;
    location / {
        # without an explicit upstream group config.
        proxy_pass http://upyun.com;
    }
}
```

Still upyun.com will be resolved only once.

```
server {
    listen *:8080;
    resolver 1.1.1.1:53 2.2.2.2:53 3.3.3.3:53 ipv6=off;
    location / {
        set $backend upyun.com;
        proxy_pass http://$backend;
    }
}
```

Force ngx_proxy module to parse the “complex value”.

Now upyun.com will be resolved by Nginx’s runtime DNS resolver every time the ngx_proxy module runs.

Nginx runtime DNS resolver

√ load balancing (RR)
√ cache
× backup
× failover
× use stale data

lua-resty-domain

- Based on lua-resty-dns.
- Combined with lua-resty-shcache, lua-resty-checkups.

√ load balancing
√ heartbeat and failover
√ backend
√ cache
√ use stale data

```
_M.dns = {
    enable = true,

    cluster = {
        {
            servers = {
                { host = “1.1.1.1”, port = 53 },
                { host = “2.2.2.2”, port = 53 },
            },
        },
        { -- backup
            servers = {
                { host = “3.3.3.3”, port = 53 },
            },
        },
    },

    retrans = 2,
    timeout = 1000,

    -- shcache ttl parameters
    positive_ttl = 15,
    negative_ttl = 5,
}
```

Patches for Cosocket

```
_G.ngx.socket.tcp = function(...)
    local sock = tcp_sock(...)
    sock.connect = function(sock, host, port, opts)
        local addr = domain.toip(host)
        return raw_connect(sock, addr, port, opts)
    end
    return sock
end
```

Force Cosocket use our lua-resty-domain rather than the Nginx runtime DNS resolver, borrowed from Kong. :)

SSL Acceleration

Offload the RSA/ECDHE/SHA/… tasks to hardware (Intel QAT card), to reduce application's load.

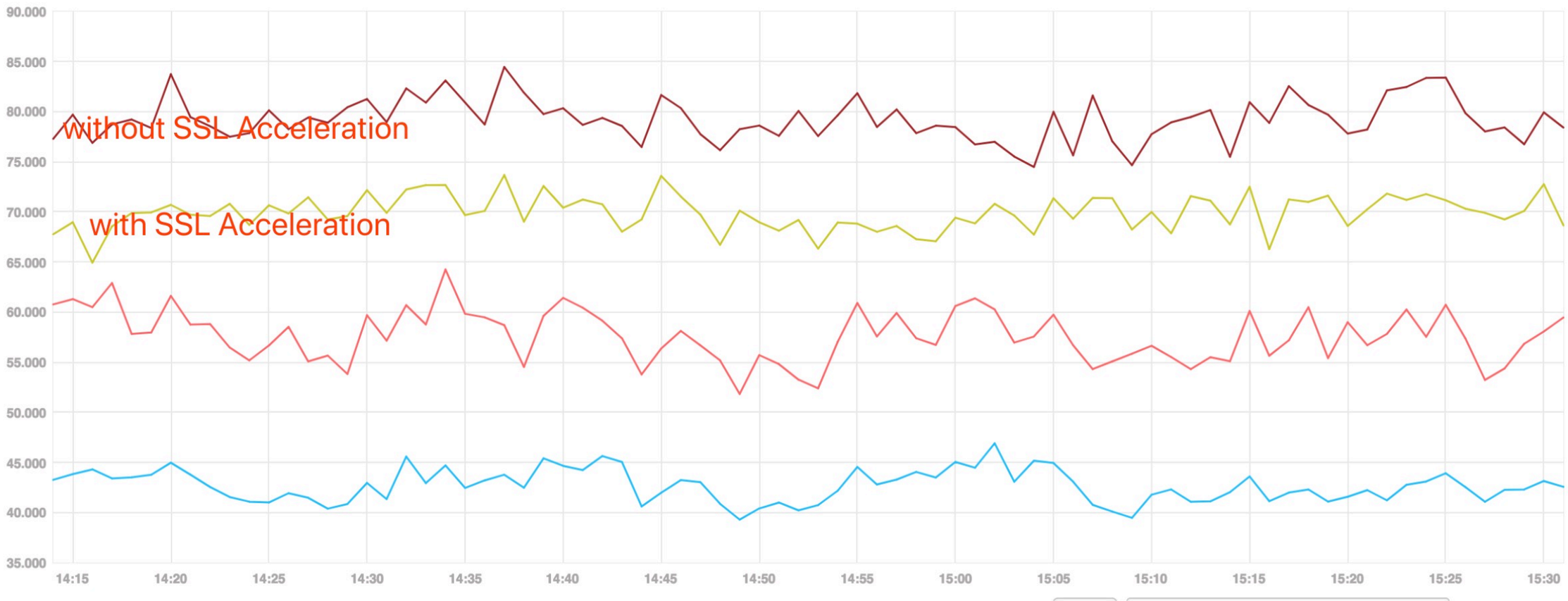
Acceleration is not really "speed up", but aims at increasing the application's throughput.

OpenResty + Asynchronous OpenSSL + Intel QAT

```
#openssl speed -elapsed rsa2048
You have chosen to measure elapsed time instead of user CPU time.
Doing 2048 bits private rsa's for 10s: 16640 2048 bits private RSA's in 10.00s
Doing 2048 bits public rsa's for 10s: 332804 2048 bits public RSA's in 10.00s

                  sign    verify    sign/s verify/s
rsa 2048 bits 0.000601s 0.000030s   1664.0  33280.4
```

```
#openssl speed -elapsed -engine qat -async_jobs 72 rsa2048
engine "qat" set.
You have chosen to measure elapsed time instead of user CPU time.
Doing 2048 bits private rsa's for 10s: 179738 2048 bits private RSA's in 10.01s
Doing 2048 bits public rsa's for 10s: 2213458 2048 bits public RSA's in 10.00s
                  sign    verify    sign/s verify/s
rsa 2048 bits 0.000056s 0.000005s  17955.8 221345.8
```




```
without-ssl-acceleration# ss -tn sport = :443 | wc -l
32863
with-ssl-acceleration# ss -tn sport = :443 | wc -l
34689
```

CPU util ↓ ~10%。

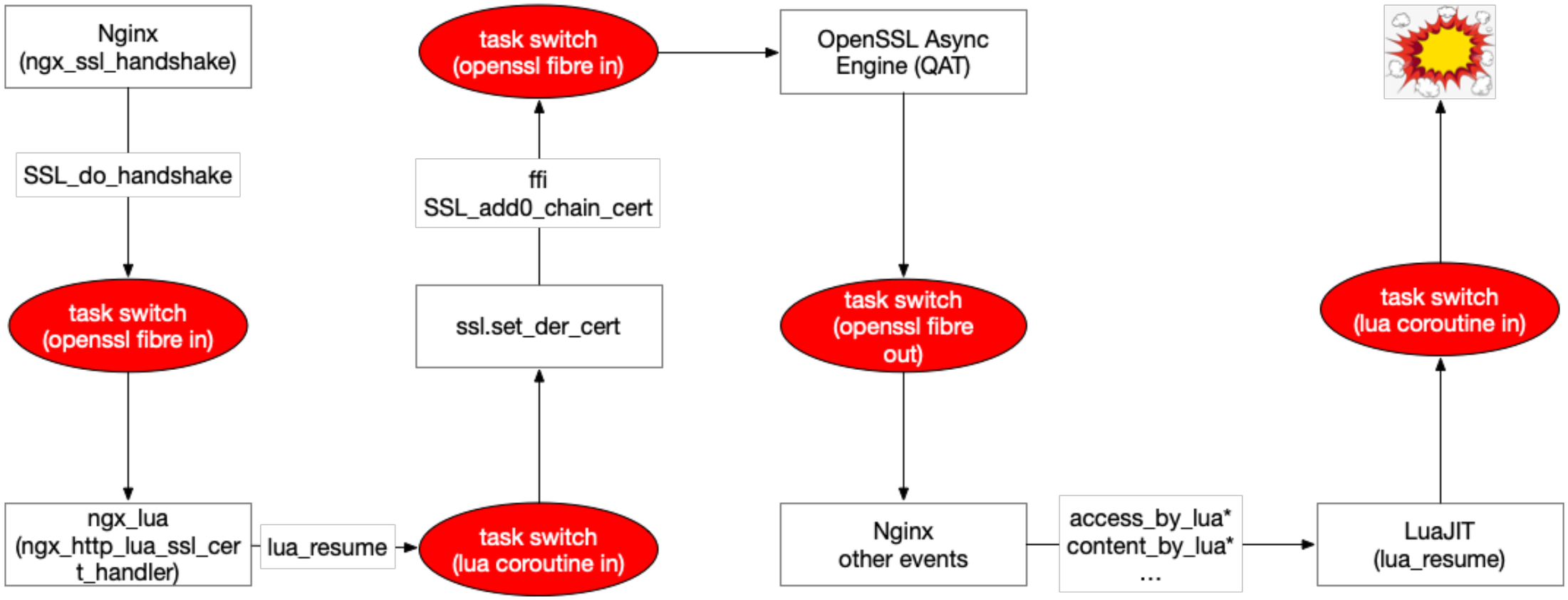
Conflict with LuaJIT

```
(gdb) bt
#0  0x0000000000000000 in ?? ()
#1  0x000000000047f8f1 in ngx_http_request_handler (ev=0x421ee98) at src/http/ngx_http_request.c:2259
#2  0x00000000004515d0 in ngx_event_process_posted (cycle=0x2c1c050, posted=0x9baa40 <ngx_posted_events>) at
src/event/ngx_event_posted.c:33
#3  0x000000000044f2ee in ngx_process_events_and_timers (cycle=0x2c1c050) at src/event/ngx_event.c:265
#4  0x000000000045c42b in ngx_worker_process_cycle (cycle=0x2c1c050, data=0x5) at src/os/unix/
ngx_process_cycle.c:792
#5  0x0000000000458eda in ngx_spawn_process (cycle=0x2c1c050, proc=0x45c37d <ngx_worker_process_cycle>,
data=0x5, name=0x6afd70 "worker process", respawn=-4)
    at src/os/unix/ngx_process.c:202
#6  0x000000000045b570 in ngx_start_worker_processes (cycle=0x2c1c050, n=16, type=-4) at src/os/unix/
ngx_process_cycle.c:373
#7  0x000000000045b0cd in ngx_master_process_cycle (cycle=0x2c1c050) at src/os/unix/ngx_process_cycle.c:256
#8  0x000000000041cef0 in main (argc=1, argv=0x7fff66d95838) at src/core/nginx.c:393
```

```
(gdb) bt
#0  0x00007f0c1a6aade2 in lj_vm_growstack_f () from /usr/local/marco/luajit/lib/libluajit-5.1.so.2
#1  0x0000000000550c13 in ngx_http_lua_run_thread (L=0x41a00378, r=0x1767f80, ctx=0x13793f0, nrets=0)
    at /disk/ssd2/alex_workflow/marco/deps/lua-nginx-module-0.10.11h/src/ngx_http_lua_util.c:1013
#2  0x000000000057825c in ngx_http_lua_ssl_cert_by_chunk (L=0x41a00378, r=0x1767f80)
    at /disk/ssd2/alex_workflow/marco/deps/lua-nginx-module-0.10.11h/src/ngx_http_lua_ssl_certby.c:527
#3  0x0000000000577457 in ngx_http_lua_ssl_cert_handler_file (r=0x1767f80, lscf=0x12c2138, L=0x41a00378)
    at /disk/ssd2/alex_workflow/marco/deps/lua-nginx-module-0.10.11h/src/ngx_http_lua_ssl_certby.c:57
#4  0x0000000000577c2c in ngx_http_lua_ssl_cert_handler (ssl_conn=0x1765790, data=0x0)
    at /disk/ssd2/alex_workflow/marco/deps/lua-nginx-module-0.10.11h/src/ngx_http_lua_ssl_certby.c:315
#5  0x00007f0c1a1e544a in tls_post_process_client_hello (s=0x1765790, wst=WORK_MORE_B) at ssl/statem/sta
#6  0x00007f0c1a1e2d2f in ossl_statem_server_post_process_message (s=0x1765790, wst=WORK_MORE_A) at ssl/
#7  0x00007f0c1a1cfe52 in read_state_machine (s=0x1765790) at ssl/statem/statem.c:660
#8  0x00007f0c1a1cf7a9 in state_machine (s=0x1765790, server=1) at ssl/statem/statem.c:428
#9  0x00007f0c1a1cf33b in ossl_statem_accept (s=0x1765790) at ssl/statem/statem.c:251
#10 0x00007f0c1a1b642d in ssl_do_handshake_intern (vargs=0x12c6730) at ssl/ssl_lib.c:3467
#11 0x00007f0c19d0770f in async_start_func () at crypto/async/async.c:154
#12 0x00007f0c199108f0 in __malloc_info (fp=0x7ffea8f29be0, options=<optimized out>) at malloc.c:5196
#13 0x0000000001f6b870 in ?? ()
#14 0x0000000000000000 in ?? ()
```

The devil is

- JIT compiler?
- Asynchronous OpenSSL mode?
- Lua task switch (Light Thread) in ssl_certificate_by_lua*?
- Nginx self?
- blah blah blah …



System Stack -> OpenSSL Stack -> Lua Stack -> OpenSSL Stack -> System Stack -> Lua Stack

Reentry LuaJIT without calling `lua_yield()`!

Fixup

Prohibit OpenSSL fibres' task switch in SSL-relevant APIs.

- ssl.set_der_cert: ngx_http_lua_ffi_ssl_set_der_certificate
- ssl.set_der_priv_key: ngx_http_lua_ffi_ssl_set_der_private_key
- ssl.validate_ocsp_response: ngx_http_lua_ffi_ssl_validate_ocsp_response

```
#if OPENSSL_VERSION_NUMBER >= 0x10100000L
    ASYNC_block_pause();
#endif
……

#if OPENSSL_VERSION_NUMBER >= 0x10100000L
    ASYNC_unblock_pause();
#endif
```

Thanks!