Layered Caching in OpenResty

OpenResty Con 2018

Thibault Charbonnier

Principal Engineer @ Kong Inc.
OpenResty contributor

November 18th, 2018

"OpenResty could use a better caching abstraction"

What to cache?

- Client sessions
- Configuration
- Injected logic
- Any state fetched from I/O

Challenges

- Forked nginx workers
- LuaJIT VM
- lua_shared_dict serialization


Kong

Caching in Kong

```
-- Retrieve a value from the cache, or fetch it
-- if it's a miss
function _M.cache_get_and_set(key, cb)
    local val = _M.cache_get(key)
    if not val then
        val = cb()
        if val then
            local succ, err = _M.cache_set(key, val)
            if not succ and ngx then
                ngx.log(ngx.ERR, err)
            end
        end
    end
    return val
end
```

Existing caching libraries

- mtourne/ngx.shcache
- lloydzhou/lua-resty-cache
- hamishforbes/lua-resty-tlc

Caching primitives in OpenResty

OpenResty offers a few off-the-shelf options for caching data for the Lua-land:

- The Lua VM itself
- lua-resty-lrucache
- lua_shared_dict

Caching primitives in OpenResty

OpenResty offers a few off-the-shelf options for caching data for the Lua-land:

- The Lua VM itself
- lua-resty-lrucache
- lua_shared_dict

Caching primitives in OpenResty

OpenResty offers a few off-the-shelf options for caching data for the Lua-land:

- The Lua VM itself
- lua-resty-lrucache
- lua_shared_dict

lua-resty-mlcache

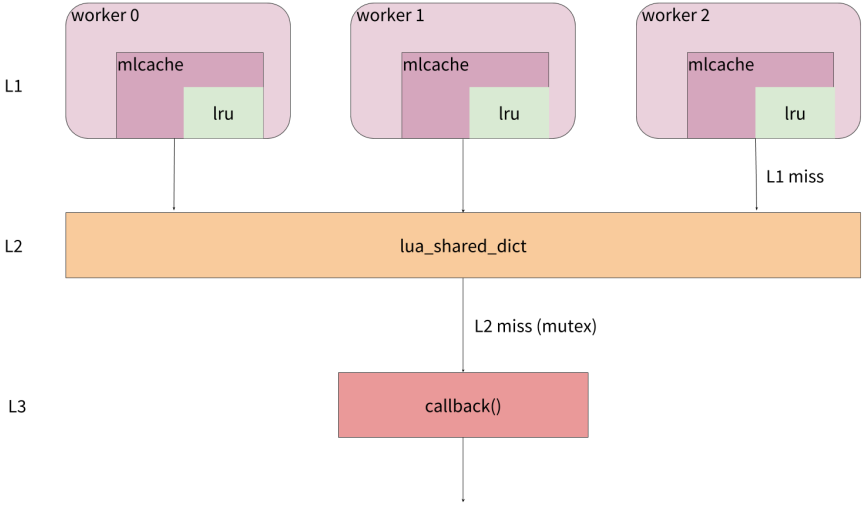
lua-resty-mlcache

The Swiss Army Knife of OpenResty caching

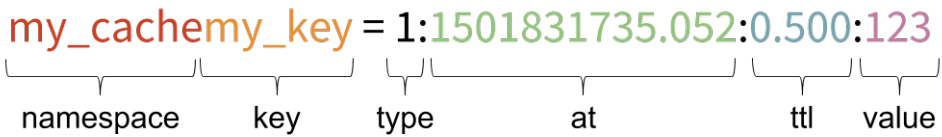
Methods

```
mlcache:get()
mlcache:set()
mlcache:delete()
mlcache:peek()
mlcache:purge()
mlcache:update()
```

Layered Architecture



lua_shared_dict serialization



Usage

```
http {
    lua_shared_dict cache_shm 128m;

    server {
        ...
    }
}
```

Usage

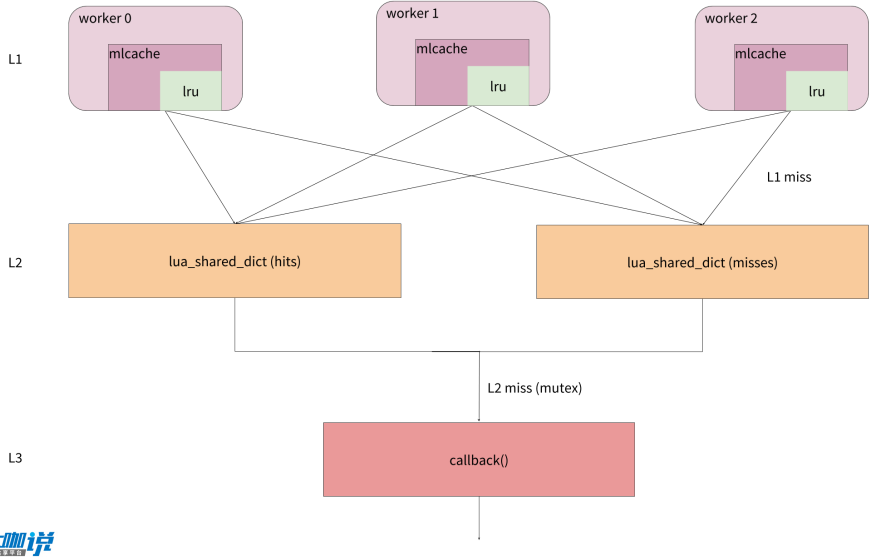
```lua
local mlcache = require "resty.mlcache"

local cache, err = mlcache.new("cache_name", "cache_shm", {
    lru_size = 1000, -- hold up to 1000 items in the L1 cache (Lua VM)
    ttl      = 3600, -- cache scalar types and tables for 1h
    neg_ttl  = 60    -- cache nil values for 60s
})
if not cache then
    error("failed to create mlcache: " .. err)
end
```

Practical Examples

Database caching

```lua
local function fetch_user(id)
    return db:query_user(id) -- row or nil
end

local id = 123

local user, err = cache:get(id, nil, fetch_user, id)
if err then
    ngx.log(ngx.ERR, "failed to fetch user: ", err)
    return
end

if user then
    print(user.id) -- 123
else
    -- miss is cached
end
```

Separate hit/miss caches

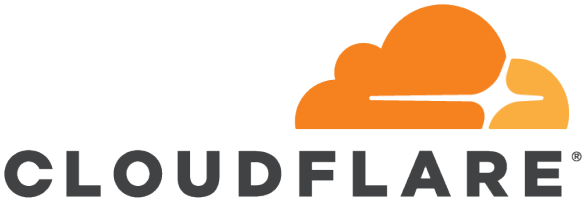


DNS caching

```
local resolver = require "resty.dns.resolver"
local r = resolver.new({ nameservers = { "1.1.1.1" } })

local function resolve(name)
    local answers = r:query(name)

    return answers[1], nil, answers[1].ttl -- override TTL
end

local host = "openresty.org"

local answers, err = cache:get(host, nil, resolve, host)
if err then
    -- ...
end
```

Injected logic

```lua
local function compile_code(row)
    row.f = loadstring(row.code) -- once
    return row
end

local user, err = cache:get(user_id, {
                        l1_serializer = compile_code
                    }, fetch_code)
if err then
    -- ...
end

user.f()
```


CLOUDFLARE®

Cache invalidation

worker 0: invalidate item

```
cache:delete("user:123")
```

worker 1: poll + re-fetch item

```
cache:update()

cache:get("user:123", nil,
    fetch_user)
```

A complete solution

- Negative caching
- Built-in mutex
- Caching Lua tables
- Invalidation events
- Flexible IPC support
  - Built-in
  - lua-resty-worker-events
  - slact/ngx_lua_ipc
- Hit levels tracking

In the wild

- Used in Kong for over a year
- Contributions from Cloudflare
- Well tested
  - OpenResty 1.11.2.2 to 1.13.6.2 (current)

OpenResty contributions & Improvements

- New shm API: `shm:ttl()` & `shm:expire()`
- User flags support - openresty/lua-resty-lrucache#35
- TODO: A Lua-land R/W lock
- TODO: A native IPC solution for OpenResty!

Q & A

Can you replace your caching logic with mlcache?

https://github.com/thibaultcha/lua-resty-mlcache

Contributions are much welcome!