mod_wombat
Apache + Lua + Worker For The Win!

Brian McCallister
Ning
require 'apache2'
require 'math'

s = "%/www/ac_us_07/htdocs/%s.html"

function rewrite_index(r)
  if r.uri == "/index.html" then
    r.filename = s:format(math.random(5))
    r:debug("sending " .. r.filename)
    return apache2.OK
  end
  return apache2.DECLINED
end

I got really tired of cases where apache comes >> close to exactly what I want.
mod_rewrite not enough, mod_perl too... heavy and perlish, mod_ruby... no threads, mod_python, GiL
and HUGE memory footprint. Need a really small, actual language, which you can do little, fast,
efficient modules in.
Why Lua?

The obvious question from there is why did we use Lua?
The first thing is its thread behavior. It doesn’t do threads, but it also encapsulates all interpreter state in a struct, so you can have as many as you want being accessed from different threads. Actually, it does to incredibly efficient cooperative multithreading, but not POSIX threads.
# Overhead

<table>
<thead>
<tr>
<th>Server</th>
<th>Reqs / Second</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache HTTPD mod_hello_world</td>
<td>18,823.58</td>
<td>minimal module which just prints hello world</td>
</tr>
<tr>
<td>Apache HTTPD mod_wombat</td>
<td>17,856.76</td>
<td>server-scoped mod_wombat handler</td>
</tr>
<tr>
<td>Apache Tomcat 5.5.20</td>
<td>17,644.40</td>
<td>JSP</td>
</tr>
<tr>
<td>Jetty 6.1.1</td>
<td>12,449.36</td>
<td>JSP</td>
</tr>
<tr>
<td>Mongrel</td>
<td>2,378.05</td>
<td>HttpHandler, not Rails</td>
</tr>
</tbody>
</table>

This invalid and pointless benchmark attempted to measure the cost of using a particular system. It did nothing but respond with “hello world” It is designed to see what the relative overhead to a baseline (mod_hello_world) is, mod_wombat did pretty nicely! In truth, the benchmark is even less valid. I had to run it twice before mod_hello_world was faster than mod_wombat. 15k or 16k RSS overhead per lua_State.
Third, Lua itself is actually damned fast. Not C fast, but it thrashes most other scripting languages in most circumstances.
Finally, it has to actually embed and play easily with C. Lua is 100% designed for... embedding in C/C++
Why Not $LANG$?

<table>
<thead>
<tr>
<th>Guile</th>
<th>Python</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gauche</td>
<td>SpiderMonkey</td>
</tr>
<tr>
<td>Chicken</td>
<td>Perl</td>
</tr>
<tr>
<td>MZScheme</td>
<td>Ruby</td>
</tr>
<tr>
<td>Elk</td>
<td>TCL</td>
</tr>
<tr>
<td>Scheme48</td>
<td>PHP</td>
</tr>
</tbody>
</table>

Make note that Lua uses the MIT license, matters to people here :-)

* Because Vadim asked
Global Interpreter
Global Namespace
Thread Behavior
Size of Runtime
Line Noise :-)

Line Noise :-(
Putting Wombat to Work
function bar(self, val)
    print(self.one .. " " .. val)
end

local foo = {
    one = 1,
    baz = bar
}

print(foo.one)
print(foo["one"]) 

foo:baz("hello")
foo.baz(foo, "hello")
bar(foo, "hello")

Brief tour of Lua -- SMALL -- SIMPLE --
Tables Tables TABLES
dynamic typing, first class functions, tables, “OO” sugar on tables
local mu = require "moonunit"
local test = mu.TestCase:new{}

module("moonunit", package.seeall)

TestCase = {}
function TestCase:new(it)
    it = it or {}
    setmetatable(it, self)
    self.__index = self
    return it
end
ServerRoot "/opt/httpd-2.2.6"
DocumentRoot "\\www\ac_us_07\htdocs"
Listen 80

LoadModule apreq_module modules/mod_apreq2.so
LoadModule wombat_module modules/mod_wombat.so

LuaRoot "\\www\ac_us_07\"

DirectoryIndex index.lua

AddHandler lua-script .lua

LuaHookTranslateName lib/hooks.lua \ rewrite_index

Point out the Lua bits, and the (current) dependency on apreq2
ServerRoot "/opt/httpd-2.2.6"
DocumentRoot "'/www/ac_us_07/htdocs"
Listen 80

LoadModule apr eq _module modules/mod_apreq2.so
LoadModule wombat_module modules/mod_wombat.so

LuaRoot "'/www/ac_us_07/"

DirectoryIndex index.lua

AddHandler lua-script .lua

LuaHookTranslateName lib/hooks.lua \ rewrite_index

The file is LuaRoot .. first arg to the directive
The function is second arg to the directive
require 'string'
require 'apache2'
\htdocs = '/www/ac_us_07/htdocs'

function rewrite_index(r)
  if r.uri == '/index.html' then
    r.filename = htdocs .. '/index.lua'
    return apache2.OK
  end

  return apache2.DECLINED
end
-- /www/ac_us_07/lib/hooks.lua
require 'string'
require 'apache2'

htdocs = '/www/ac_us_07/htdocs

function rewrite_index(r)
    if r.uri == '/index.html' then
        r.filename = htdocs .. '/index.lua'
        return apache2.OK
    end
    return apache2.DECLINED
end
-- /www/ac_us_07/lib/hooks.lua
require 'string'
require 'apache2'

htdocs = "'/www/ac_us_07/htdocs"

function rewrite_index(r)
    if r.uri == "'/index.html" then
        r.filename = htdocs .. "'/index.lua"
        return apache2.OK
    end
    return apache2.DECLINED
end

Look at function itself. The argument is the request_rec (sorta).
Access the standard fields on it just as lua table entries, both read and write (uses scary metatable hackery)
apache2.OK and apache2.DECLINED are the same as the C constants
(Should map in the DocumentRoot, LuaRoot, etc somewhere)
Just to point out, this is just setting and reading from the request_rec
require 'string'

function handle(r)
    local args = r:parseargs()
    if args['name'] then
        name = args.name
    else
        name = 'World'
    end
    r:puts((('Hello, %s!\n'):format(name)) )
end

Evolving because the dependency on libapreq2 will probably go away. Exactly how the API will be structured then is unknown. I like this way a lot, so it may just be that we’ll test for apreq’s availability and do it with apreq if avail, and parse out in Lua if not avail, but API will be the same. One will just be more efficient and leave the args around for C modules
require 'string'

function handle(r)
    local args = r:parseargs()
    if args['name'] then
        name = args.name
    else
        name = 'World'
    end
    r:puts( ('Hello, %s!\n'):format(name) )
end
function handle(r)
    local _, full = r:parseargs()
    local names = full['name']
    for _, name in ipairs(names) do
        r:puts( ('Hello, %s!\n'):format(name) )
    end
end
function handle(r)
    local _, full = r:parseargs()
    local names = full['name']

    for _, name in ipairs(names) do
        r:puts( ('Hello, %s!\n'):format(name) )
    end
end

returning tuples, ir the parseargs()
traversing array
libapreq2 is becoming optional, may change to something like: apreq.parsearegs(r)
function logging_stuff(r)
    r:debug("This is a debug log message")
    r:info("This is an info log message")
    r:notice("This is an notice log message")
    r:warn("This is an warn log message")
    r:err("This is an err log message")
    r:alert("This is an alert log message")
    r:crit("This is an crit log message")
    r:emerg("This is an emerg log message")
end
function server_rec(r)
    local server = r.server -- server_rec
    r:puts(server.server_hostname)
end

struct server_rec {
    /* ... */
    char *server_hostname;
    /* ... */
}

Evolving -- things other than request_rec are fuzzy as changing how we map
LuaRoot "/@www/ac_us_07/"

LuaQuickHandler lib/hooks.lua quick
LuaHookTranslateName lib/hooks.lua trans_name
LuaHookMapToStorage lib/hooks.lua mapstorage
LuaHookAccessChecker lib/hooks.lua hosty
LuaHookCheckUserID lib/hooks.lua authn
LuaHookAuthChecker lib/hooks.lua authz
LuaHookTypeChecker lib/hooks.lua typecheck
LuaHookFixups lib/hooks.lua fixup

AddHandler lua-script .lua

MapLuaHandler ^/(\w+)_/(\w+)/$ lib/$1.lua h_$2

Names are likely to be normalized Real Soon Now
TODO: Get ordering here to match order of execution
On the C Side

(Briefly)
require "myputs"

function handle(r)
    local msg = r:myputs("Hello", " ", "world")
    r:debug(msg)
end
<table>
<thead>
<tr>
<th>userdata: [Apache2.Request]</th>
</tr>
</thead>
<tbody>
<tr>
<td>string: &quot;hello&quot;</td>
</tr>
<tr>
<td>string: &quot; &quot;</td>
</tr>
<tr>
<td>string: &quot;world &quot;</td>
</tr>
</tbody>
</table>
static int my_special_puts(lua_State* L) {
    request_rec* r = check_request_rec(L, 1);

    int argc = lua_gettop(L);
    int i;
    for (i=2;i<=argc;i++) {
        ap_rputs(luaL_checkstring(L, i), r);
    }
    lua_pushstring(L, "Thank You!");
    return 1;
}
static int my_special_puts(lua_State* L) {
    request_rec* r = check_request_rec(L, 1);
    int argc = lua_gettop(L);
    int i;
    for (i=2;i<=argc;i++) {
        ap_rputs(luaL_checkstring(L, i), r);
    }
    lua_pushstring(L, "Thank You!");
    return 1;
}
static int my_special_puts(lua_State* L) {
    request_rec* r = check_request_rec(L, 1);

    int argc = lua_gettop(L);
    int i;
    for (i=2;i<=argc;i++) {
        ap_rputs(luaL_checkstring(L, i), r);
    }
    lua_pushstring(L, "Thank You!");
    return 1;
}
static int my_special_puts(lua_State* L) {
    request_rec* r = check_request_rec(L, 1);

    int argc = lua_gettop(L);
    int i;
    for (i=2;i<=argc;i++) {
        ap_rputs(luaL_checkstring(L, i), r);
    }
    lua_pushstring(L, "Thank You!");
    return 1;
}
string: "Thank You! "
function handle(r)
    local msg = r:myputs("Hello", " ", "world")
    r:debug(msg)
end
int luaopen_myputs(lua_State *L) {
    luaL_getmetatable(L, "Apache2.Request");
    lua_pushcfunction(L, my_special_puts);
    lua_setfield(L, -2, "mypsuts");
    lua_newtable(L);
    return 1;
}
int luaopen_myputs(lua_State *L) {
    luaL_getmetatable(L, "Apache2.Request");
    lua_pushcfunction(L, my_special_puts);
    lua_setfield(L, -2, "myputs");
    lua_newtable(L);
    return 1;
}

this is a BAD BAD BAD module
How Wombat Works

Differently then others

As mentioned, one of the driving forces behind mod_wombat is working well with Apache’s threading model.
Two things to review before diving in. The first is how the Worker MPM works...
lua_State *L = apw_rgetvm(r, spec);
lua_getglobal(L, d->function_name);
if (lua_pcall(L, 1, 0, 0)) {
    report_lua_error(L, r);
}
Connection Scope
Server Scope
NEVER, STAT, FOREVER
Once and Stat are the defaults
typedef struct {
    const char *file;
    int code_cache_style;
    int scope;

    // APW_SCOPE_ONCE only
    apr_pool_t *pool;

    apr_array_header_t* package_paths;
} apw_vm_spec;
function configure(cmd, dir)
    dir:match_handler {
        pattern = "^/server-says-hi$",
        file = "htdocs/config_tests.lua",
        func = "handle_server_vm",
        scope = "server"
    }
end
State (and Future) of the Wombat
Wombat is a wee baby.

setback: The three folks who had contributed the most all had kids. Stallman would turn in his grave. If he was dead. Having been killed by ninjas.
LuaRoot "/www/ac_us_07/"

LuaConfig httpd_config.lua configure
LuaQuickHandler lib/hooks.lua quick
LuaHookTranslateName lib/hooks.lua trans_name

```lua
function configure(cmd, dir)
    dir:match_handler {
        pattern = "^/server-says-hi$",
        file = "htdocs/config_tests.lua",
        func = "handle_server_vm",
        scope = "server"
    }
end
```

Configuration is evolving
The way the apache C structs are packaged into Lua will probably change (Thank you Rici Lake!) How wombat itself is packaged as well!
The URL

http://svn.apache.org/repos/asf/httpd/mod_wombat

NO web presence. Seriously. It is evolving quickly so there is just the svn repo. This is it. There are docs, which are kept up to date, in the svn repo.
Resources & Docs

http://www.lua.org/

http://www.lua.org/manual/5.1/

http://www.lua.org/pil/

Programming in Lua, Roberto Ierusalimschy

The Apache Modules Book, Nick Kew
That’s All Folks!

Brian McCallister
http://kasparov.skife.org/wombat _ac_us_07.pdf
Bonus!

Because we have spare time...
local mu = require "moonunit"
local http = require "helpers"

http.base_url = "http://localhost:8000"

local test = mu.TestCase:new{}

function test:basic_get()
  local body, code = http.get "/basic"
  assert(200 == code,
    "expected 200, got " .. code)
  assert(body:find("hello Lua world"))
end

test:run()

Moon Unit is a micro-xUnit test harness which is part of mod_wombat. It is pretty.
That’s all you need to get this nice test suite!
That’s (Really) All Folks!

Brian McCallister
http://kasparov.skife.org/wombat_ac_us_07.pdf