Rewriting LuaJIT: Why and How?

Anton Soldatov, IPONWEB

Lua Workshop
Kaunas, 06.09.2018
About IPONWEB

- Building platforms for real-time advertising
- Workloads up to 6M requests per second
- Lua is used for more than 10 years for implementing business logic in our projects
Lua in IPONWEB

<table>
<thead>
<tr>
<th>Business logic (<strong>Lua</strong>, sandboxed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Libraries (<strong>Lua</strong>)</td>
</tr>
</tbody>
</table>

- Application server core (C++): multithreading, networking, coroutine management, etc.
LuaJIT in IPONWEB

- Experience with LuaJIT 1.1 and 2.0
LuaJIT in IPONWEB

- Experience with LuaJIT 1.1 and 2.0
- Definitely benefit from the faster interpreter
LuaJIT in IPONWEB

- Experience with LuaJIT 1.1 and 2.0
- Definitely benefit from the faster interpreter
- Generally benefit from the JIT compiler
LuaJIT in IPONWEB

- Experience with LuaJIT 1.1 and 2.0
- Definitely benefit from the faster interpreter
- Generally benefit from the JIT compiler
  - ...but we do not avoid pairs at all costs
LuaJIT in IPONWEB

- Experience with LuaJIT 1.1 and 2.0
- Definitely benefit from the faster interpreter
- Generally benefit from the JIT compiler
  - ...but we do not avoid \texttt{pairs} at all costs
- Sandboxing partly reduces Lua/LuaJIT incompatibility tension
LuaJIT in IPONWEB

- Experience with LuaJIT 1.1 and 2.0
- Definitely benefit from the faster interpreter
- Generally benefit from the JIT compiler
  - ...but we do not avoid pairs at all costs
- Sandboxing partly reduces Lua/LuaJIT incompatibility tension
- Limited experience with FFI
Data feeds

• Inventory lists
• Rules for decision making
Data feeds: memory consumption
Our main issue with LuaJIT 2.0

We have eventually hit the memory limit on x86-64:

```c
void *ptr = mmap((void *)MMAP_REGION_START,
                 size, MMAP_PROT,
                 MAP_32BIT | MMAP_FLAGS, -1, 0);
```
Tried a work-around with FFI

- Decompose feeds into simpler data structures
- Map into native memory
- Build accessors with FFI
Tried a work-around with FFI

- Decompose feeds into simpler data structures
- Map into native memory
- Build accessors with FFI
- Performance has degraded
Possible solutions

- Migrate to PUC-Rio Lua
Possible solutions

- Migrate to PUC-Rio Lua
- Migrate to LuaJIT 2.1
Possible solutions

- Migrate to PUC-Rio Lua
- Migrate to LuaJIT 2.1
- Start an own implementation
Implementation requirements

- Fix the memory limit
- Become not slower than LuaJIT 2.0
- Target only Linux x86-64
Implementation requirements

- Fix the memory limit
- Become not slower than LuaJIT 2.0
- Target only Linux x86-64
- No changes to the language
- Stay as close to Lua 5.1 as LuaJIT 2.0
Fixing the memory limit

- **TValue is 16 bytes** ($\text{uint64}_t + \text{uint64}_t$)
Fixing the memory limit

- **TValue is 16 bytes** \((\text{uint64}_t + \text{uint64}_t)\)
- Support for true 64-bit pointers in both VM and JIT
Fixing the memory limit

- **TValue** is 16 bytes \((\text{uint64}_t + \text{uint64}_t)\)
- Support for true 64-bit pointers in both VM and JIT
- **LJ_FR2** trick not needed
Fixing the memory limit

- **TValue** is **16 bytes** (`uint64_t + uint64_t`)
- Support for true 64-bit pointers in both VM and JIT
- **LJ_FR2** trick not needed
- A multiplier of 2 was "baked" into the byte code to regain the SIB mode benefits
Fixing the memory limit: results

- About 30% faster than the FFI work-around for data feeds
- Approximately the same performance in most of other cases
Fixing the memory limit: timeline

- **Q2 2015** – Decision to build a new implementation
- **2015-2016** – Development, stabilisation and validation; pilot migrations
- **Q1 2017** – More than 95% production servers moved to the new implementation
Testing: What we started with

- Integrational tests with the application server
- Test stands
Testing: What was missing

- Language compliance tests
Testing: What was missing

- Language compliance tests
- There are test suites for some implementations, but they are scattered around
Testing: What was missing

- Language compliance tests
- There are test suites for some implementations, but they are scattered around
- Functional tests for the implementation
Testing: results

- Continuously write own tests
Testing: results

- Continuously write own tests

- Integrate third party suites into the source tree:
  - Lua 5.1 official test suite
  - Mike Pall's test suite for LuaJIT
  - François Perrad's test suite shipped with lua-TestMore
Testing: results

- Continuously write own tests

- Integrate third party suites into the source tree:
  - Lua 5.1 official test suite
  - Mike Pall's test suite for LuaJIT
  - François Perrad's test suite shipped with lua-TestMore
  - Part of Laurent Deniau's test suite from the MAD project
Extending the implementation
Data feeds: memory consumption
Objects from a data feed in memory
ujit.seal(data)
Properties of sealed objects

- GC traverses objects until the first sealed object
- Thus, all sealed objects must be immutable
ujit.seal(data)
seal = "seal per se" + immutable
Introducing immutable objects

```lua
local t = {foo = "bar"}
ujit.immutable(t)
```
immutable: Example 1

local t = ujit.immutable({{foo = "bar"}})

-- All of the following throw:

t[1].new = "baz"  -- Add

(continues)
immutable: Example 2

ujit.immutable(\_G)
Going further: more features

- Sharing read-only data between instances of the VM
Going further: more features

- Sharing read-only data between instances of the VM
- Interruptible coroutines (non-resumable and resumable)
Going further: more features

- Sharing read-only data between instances of the VM
- Interruptible coroutines (non-resumable and resumable)
- Extended Lua and C API for working with tables
Going further: more features

- Sharing read-only data between instances of the VM
- Interruptible coroutines (non-resumable and resumable)
- Extended Lua and C API for working with tables
- Tweaks in the compiler
Going further: tools

- Sampling profiler

- bit.ly/dumpanalyz – a tool for analyzing debugging info produced by the JIT compiler (\texttt{-j dump} in LuaJIT)
Conclusions

● It is possible to build an implementation of Lua based on LuaJIT, but your motivation should be strong enough.

● Be prepared to multiple challenges (and fun) while reworking the core.

● Be prepared to more challenges when it comes to testing and tools.
Thank you! Questions?
Links

- [bit.ly/dumpanalyze](bit.ly/dumpanalyze)

- asoldatov@iponweb.net
- [https://t.me/igelhaus](https://t.me/igelhaus)