Ad-hoc Big-Data Analysis with Lua

And LuaJIT

LogicEditor

Alexander Gladysh <ag@logiceditor.com>
@agladysh

Lua Workshop 2015
Stockholm

Outline

Introduction

The Elephant

Questions?

Alexander Gladysh

- CTO, co-owner at LogicEditor
- In löve with Lua since 2005

The Problem

- You have a big dataset to analyze
- that makes casual analysis tools explode or be too slow
- and you don't have resources to set up and maintain (or pay for) Hadoop, Google Big Query etc.
- but you have some processing power available.

Goal

- Pre-process the data so it can be handled by R or Excel or your favorite analytics tool (or Lua!).
- If the data is dynamic, then *learn to* pre-process it and build a data processing pipeline (which is outside of the scope of this talk).

An approach

- Use Lua!
- And (semi-)standard tools, available on Linux.
- Go minimalistic while exploring,
- Then move to an industrial solution that fits your newly understood requirements
- Or roll your own ecosystem ;-)

Start small!

- Always run your scripts on small representative excerpts from your datasets, not only while developing them locally, but on actual data-processing nodes too.
- Saves time and helps you learn the bottlenecks.
- Sometimes large run still blows in your face though.

Discipline!

- Many moving parts, large turn-around times, hard to keep tabs.
- Keep journal: Write down what you run and what time it took.
- Store actual versions of your scripts in a source control system.
- Don't forget to sanity-check the results you get!

LuaJIT?

- Up to a point:
- 2.1 helps to speed things up,
- FFI bogs down development speed.
- Go plain Lua first (run it with LuaJIT),
- then roll your own ecosystem as needed ;-)

Hardware?

- As usual, more is better: Cores, cache, memory speed and size, HDD speeds, networking speeds...
- But even a modest VM (or several) can be helpful.
- Your fancy gaming laptop is good too ;-)

OS

- Linux (Ubuntu) Server.
- Approach will, of course, work for other setups.

Data format

- Plain text
- Column-based (csv-like), optionally with free-form data in the end
- Typical example: web-server log files

Data layout

- Ideally, have data copies on each processing node, using identical layouts. Fast network should work too.
- Sort!
- TODO

The Tools

- parallel
- sort, uniq, grep
- cut, join, comm
- pv
- compression utilities
- LuaJIT

Parallel

- xargs for parallel computation
- can run your jobs in parallel on a single machine
- or on a "cluster"

Sort

- Sorted files are the key to your task

Why Lua?

Perl, AWK are traditional alternatives to Lua, but, if you're not very disciplined and experienced, they are much less maintainable.

Advice

Pre-sort everything!

Advice

Monitor resource utilization at run-time.

Compression

- gzip: default, bad
- lxc: fast, large files
- pigz: fast, parallelizable
- xz: good compression, slow
- ...and many more,
- be on lookout for new formats!

Bash script example

```bash
time pv /path/to/uid-time-url-post.gz\
| pigz -cdp 4 \
| cut -d$'\t' -f 1,3 \
| parallel --gnu --progress -P 10 --pipe --block=16M \
  $(cat <<"EOF"
    luajit ~me/url-to-normalized-domain.lua
EOF
  ) \
| LC_ALL=C sort -u -t$'\t' -k2 --parallel 6 -S20% \
| luajit ~me/reduce-key-counter.lua \
| LC_ALL=C sort -t$'\t' -nrk2 --parallel 6 -S20% \
| pigz -cp4 >/path/to/domain-uniqs_count-merged.gz
```

Lua Script Example: url-to-normalized-domain.lua

```lua
for l in io.lines() do
  local key, value = l:match("^([^\t]+)\t(.*)")
  if value then
    value = url_to_normalized_domain(value)
  end
  if key and value then
    io.write(key, "\t", value, "\n")
  end
end
```

Lua Script Example: reduce-key-counter.lua 1/3

```lua
-- Assumes input sorted by VALUE
-- a  foo --> foo  3
-- a  foo     bar  2
-- b  foo     quo  1
-- a  bar
-- c  bar
-- d  quo
```

Lua Script Example: reduce-key-counter.lua 2/3

```lua
local last_key = nil, accum = 0

local flush = function(key)
  if last_key then
    io.write(last_key, "\t", accum, "\n")
  end
  accum = 0
  last_key = key -- may be nil
end
```

Lua Script Example: reduce-key-counter.lua 3/3

```lua
for l in io.lines() do
  -- Note reverse order!
  local value, key = l:match("^(.-)\t(.*)$")
  assert(key and value)

  if key ~= last_key then
    flush(key)
    collectgarbage("step")
  end

  accum = accum + 1
end

flush()
```

Questions?

Alexander Gladysh, ag@logiceditor.com