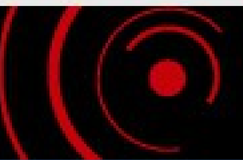
Elixir and Phoenix Performance

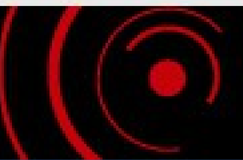
Elixir Taiwan Meetup
June 12, 2017

Jake Morrison <jake@cogini.com>



Agenda

- Architecture
- Logging
- Metrics
- Performance Tuning

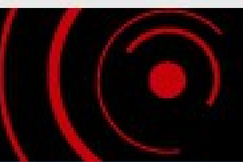


Architecture

"We do not have ONE web-server handling 2 millions sessions. We have 2 million webservers handling one session each."
- Joe Armstrong

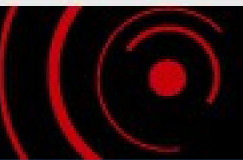
There is no magic:

- Find the real system bottlenecks: disk and network I/O, CPU, RAM
- Trade thing you have more of for thing that you do not, e.g. memory cache for db



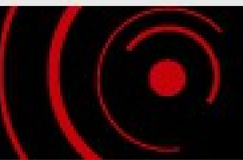
Architecture

- Anything shared is a bottleneck
- GenServer is a code smell
- Shared nothing is the best
- "Logical" three tier: libraries for different parts of your app, not processes
- Database is usually the ultimate bottleneck
- Lock contention inside the database limits number of simultaneous requests



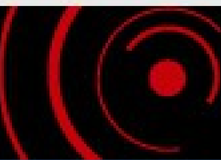
ETS is Your Friend

- Elixir data is immutable, ETS is the mechanism for mutability
- Typically 1 microsecond to read or write
- Useful for caching immutable data
- https://dockyard.com/blog/2017/05/19/optimizing-elixir-and-phoenix-with-ets



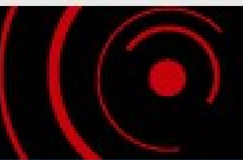
Case study: geoip lookups

- Figure out which country IP address is in
- 65 MB data file
- Started with gen_server, hit bottleneck
- Switched to pool of gen_servers, hit bottleneck
- Put it in ETS
  - Query time now 5 µs, worst case
  - Added second level "result cache" at 1 µs
- Binary data is shared out of process



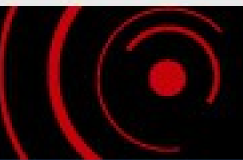
Logging is not free

- Can be the most resource intensive thing your app does
- Disk I/O and CPU
- Serializing your application through the log file, e.g. via a GenEvent server
- Have to store and move logs around
- Someone has to look at them = log blindness



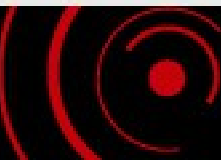
Logging is not free

- Processes send messages to the GenServer (GenEvent)
- When the GenServer mailbox fills up, your application dies
- Erlang disk_log FTW
- Separate optimized disk writing process
- 100K log records per second
- Whatever problem you have, Ericsson had it 20 years ago at BT



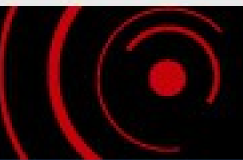
Better Logging

- Targeted logging, e.g. just requests and responses, everything else you can recreate
- Log only when there is a failure
- Erlang error logging gives you everything needed to replicate a problem
- Only log actionable information



Log Levels

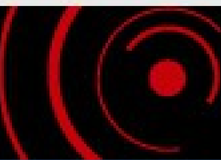
- Critical: Wake me up in the middle of the night
- Error: will look at it first thing tomorrow
- Warning: Display in staging environment
- Debug: Display on developer's machine



Log Levels

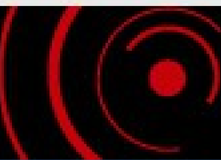
- Error: Something is broken, if it happens too much, monitoring system will tell me
- Warning: Invalid data
- Notice: Things that happen on startup or occasionally
- Info: A line of data for each request about what the system did
- Debug: Useful for developers, too much work for production

We typically run at “notice” level in production, info in test / canary, debug in dev



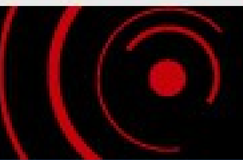
Metrics

- I don’t care about logs, what I care about is:
  - How is the system performing?
  - Where are the problems?
  - Where are the bottlenecks?
  - Are we meeting SLAs?
  - Business level metrics, e.g. signups per hour, orders per hour
- Alert on user visible symptoms, not technical failures



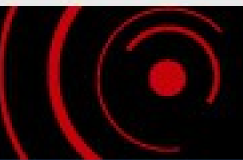
Metrics

- Counters, gauges, durations (histograms)
- Average duration vs 99% duration
- Every time you write a log message, write a counter to see how often it happens



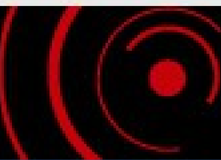
Metrics

- Number of requests
- Number of errors
- Processing duration / latency



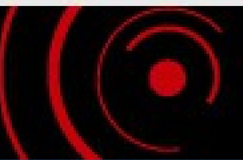
USE Method

- Utilization: “the percentage time that the resource was busy servicing work” e.g. CPU 50% busy or disk 90% full
- Saturation: “the degree to which the resource has extra work which it can’t service”, e.g. load average (task ready to run) or queue depth
- Errors: Percentage of requests with an error
- http://www.brendangregg.com/usemethod.html
- Batch processes



Measurement

- Ideally: Measure at the client and on the server
- Measure at a lower level than your application
  - Cowboy middleware

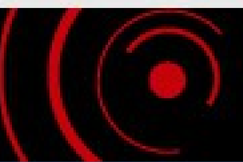


Tools

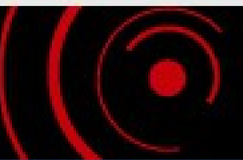
- Prometheus / Grafana

  https://prometheus.io/docs/practices/instrumentation/

- Some crazy expensive service
- Cost of cloud vs dedicated hardware
  - Log aggregation with Logstash / Elasticsearch / Kibana (ELK)
  - Tested with 60 Mbps of traffic = $600/month in AWS
  - 4 x $50/month cheap dedicated servers with i7 CPU and 32 GB RAM, 2 TB bandwidth per month = $200 for multiples of traffic

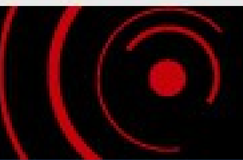


Performance Tuning



Observer

- Good overall view of what your application is doing http://erlang.org/doc/apps/observer/
  - Process structure
  - Resource usage: CPU, RAM
  - Mailbox queue size
- Recon: http://ferd.github.io/recon/
- observer_cli: https://github.com/zhongwencool/observer_cli
  - “top” for Erlang VM



Observer

nonode@nohost

System | Load Charts | Applications | Processes | Table Viewer | Trace Overview

System and Architecture

System Version:	17
Erts Version:	6.0
Compiled for:	x86_64-apple-darwin10.8.0
Emulator Wordsize:	4
Process Wordsize:	4
Smp Support:	true
Thread Support:	true
Async thread pool size:	10

Memory Usage

Total:	9526 kB
Processes:	2687 kB
Atoms:	209 kB
Binaries:	162 kB
Code:	3637 kB
Ets:	323 kB

CPU's and Threads

Logical CPU's:	2
Online Logical CPU's:	2
Available Logical CPU's:	unknown
Schedulers:	2
Online schedulers:	2
Available schedulers:	2

Statistics

Up time:	6 Mins
Max Processes:	262144
Processes:	41
Run Queue:	0
IO Input:	3656 kB
IO Output:	60 kB

Allocator Type	Block size (kB)	Carrier size (kB)
total	9525	15106
temp_alloc	0	384
sl_alloc	0	192
std_alloc	510	704
ll_alloc	7939	10752
eheap_alloc	583	1280
ets_alloc	299	704
fix_alloc	19	192

Observer

nonode@nohost

System | Load Charts | Applications | Processes | Table Viewer | Trace Overview

Scheduler Utilization (%)

10
5
0
60 s 50 s 40 s 30 s 20 s 10 s 0 s

Scheduler: 1 2

Memory Usage (KB)

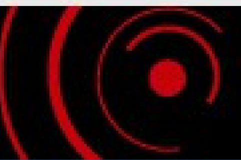
9765
4882
0
60 s 50 s 40 s 30 s 20 s 10 s 0 s

Total Processes Atom Binary Code Ets

IO Usage (B)

10
5
0
60 s 50 s 40 s 30 s 20 s 10 s 0 s

Input Output



Observer

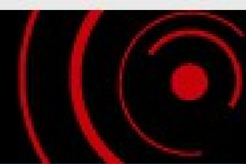
nonode@nohost

System | Load Charts | Applications | Processes | Table Viewer | Trace Overview

kernel

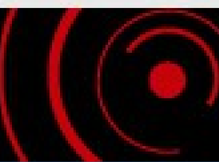
<0.9.0> — <0.10.0> — kernel_sup

- <0.22.0> — user — user_drv — <0.25.0>
- <0.27.0>
- code_server
- file_server_2
- global_group
- global_name_server — <0.14.0>, <0.15.0>
- inet_db
- kernel_safe_sup — timer_server
- rex
- standard_error_sup — standard_error



Observer

Pid	Name or Initial Func	Reds	Memory	MsgQ	Current Function
<10615.29374.220>	mochiweb_acceptor:init/4	38511700	47928152	0	lists:foldl/3
<10615.7373.1>	background_gc	25081692	22445632	0	erlang:garbage_collect/2
<10615.8133.1>	gen:init_it/6	18130030	8229680	0	erlang:hibernate/3
<10615.248.0>	rabbit_memory_monitor	2739512	11176272	47	erlang:receive_emd/3
<10615.8511.1>	rabbit_mgmt_db	1222060	67976	0	gen_server2:process_next_msg/1
<10615.26.0>	file_server_2	808859	6665264	0	gen_server:loop/6
<10615.6.0>	error_logger	788539	42576	0	gen_event:fetch_msg/5
<10615.11558.216>	cowboy_protocol:init/4	744173	21608	0	cowboy_websocket:handler_loop/4
<10615.8156.1>	gen:init_it/6	743241	625464	0	erlang:hibernate/3
<10615.7835.1>	gen:init_it/6	642578	619864	0	erlang:hibernate/3
<10615.7813.1>	gen:init_it/6	627225	23848	0	gen_server2:process_next_msg/1
<10615.11057.220>	cowboy_protocol:init/4	516332	21608	0	cowboy_websocket:handler_loop/4
<10615.7809.1>	gen:init_it/6	508818	1970440	0	erlang:hibernate/3
<10615.1068.123>	cowboy_protocol:init/4	462871	21608	0	cowboy_websocket:handler_loop/4
<10615.25943.211>	cowboy_protocol:init/4	459697	21608	0	cowboy_websocket:handler_loop/4
<10615.16823.113>	cowboy_protocol:init/4	451274	21608	0	cowboy_websocket:handler_loop/4
<10615.25695.190>	cowboy_protocol:init/4	451181	21608	0	cowboy_websocket:handler_loop/4
<10615.73.0>	mnesia_locker	450877	11848	0	mnesia_locker:loop/1
<10615.10449.188>	cowboy_protocol:init/4	450852	21608	0	cowboy_websocket:handler_loop/4
<10615.3291.136>	cowboy_protocol:init/4	450648	21608	0	cowboy_websocket:handler_loop/4
<10615.27694.216>	cowboy_protocol:init/4	449181	21712	1	gen:do_call/4
<10615.5781.137>	cowboy_protocol:init/4	447757	21608	0	cowboy_websocket:handler_loop/4
<10615.25982.195>	cowboy_protocol:init/4	446619	21608	0	cowboy_websocket:handler_loop/4
<10615.9100.148>	cowboy_protocol:init/4	443630	21608	0	cowboy_websocket:handler_loop/4
<10615.23436.182>	cowboy_protocol:init/4	443095	21608	0	cowboy_websocket:handler_loop/4
<10615.8015.129>	cowboy_protocol:init/4	441507	21608	0	cowboy_websocket:handler_loop/4
<10615.27683.219>	cowboy_protocol:init/4	441466	21608	0	cowboy_websocket:handler_loop/4

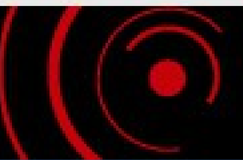


observer_cli

```
|o(OBSERVER)|e(ETS/SYSTEM)|a(ALLOCATOR)|db(MNESIA)|h(HELP)|recon:proc_count(memory, 28, 0) Refresh:5000ms                  0Days 0:0:21|
|Erlang R16B03 (erts-5.10.4) [source] [64-bit] [smp:24:1] [async-threads:10] [kernel-poll:false]                                      |
|System       | Count/Limit            | System Switch         | State                      | Memory Info           | Megabyte             |
|Proc Count   | 26/262144              | Smp Support           | true                       | Allocted Mem          | 31.0961M      100.0% |
|Port Count   | 4/65536                | Multi Scheduling      | enabled                    | Use Mem               | 15.0643M      48.94% |
|Ets Limit    | 2053                   | Logical Processors    | 24                         | Unuse Mem             | 16.0317M      51.05% |
|Memory       | State                  | Memory                | State                      | Memory                | Interval: 2500ms     |
|Total        | 15.0297M          100% | Binary                | 0.0126M             00.82% | IO Output             | 0.0000M              |
|Process      | 4.0845M         31.67% | Code                  | 4.0287M             28.02% | IO Input              | 0.0000M              |
|Atom         | 0.0178M         01.16% | Reductions            | 84350                      | Gc Count              | 1                    |
|Ets          | 0.0243M         01.58% | Run Queue             | 0                          | Gc Words Reclaimed    | 9820                 |
|01[||||||||||||||||||||||||||||||100.0%] | 09[                        01.29%] |17[                                01.29%]|
|02[                               02.12%] | 10[                        01.33%] |18[                                01.29%]|
|03[                               01.39%] | 11[                        01.31%] |19[                                01.65%]|
|04[                               01.37%] | 12[                        01.32%] |20[                                01.30%]|
|05[                               01.32%] | 13[                        01.32%] |21[                                01.30%]|
|06[                               01.34%] | 14[                        01.29%] |22[                                01.28%]|
|07[                               01.35%] | 15[                        01.27%] |23[                                01.28%]|
|08[                               01.34%] | 16[                        01.30%] |24[                                01.34%]|
|Pos|Pid           |      Memory| Name or Initial Call             | Reductions|Msg Queue |Current Function                      |
|1  |<0.29.0>      |      230392| erlang:apply/2                   |       3563|0         |shell:shell_rep/4                     |
|2  |<0.20.0>      |      142680| code_server                      |      98817|0         |code_server:loop/1                    |
|3  |<0.3.0>       |       88432| erl_prim_loader                  |     191970|0         |erl_prim_loader:loop/3                |
|4  |<0.26.0>      |       34424| group:server/3                   |     111349|0         |group:more_data/5                     |
|5  |<0.0.0>       |       26384| init                             |       4338|0         |init:loop/1                           |
|6  |<0.24.0>      |       21600| user_drv                         |      40984|0         |user_drv:server_loop/5                |
|7  |<0.7.0>       |       18544| application_controller           |        464|0         |gen_server:loop/6                     |
|8  |<0.11.0>      |       12144| kernel_sup                       |       1655|0         |gen_server:loop/6                     |
|9  |<0.33.0>      |        8696| erlang:apply/2                   |        548|0         |io:wait_io_mon_reply/2                |
|10 |<0.9.0>       |        6896| proc_lib:init_p/5                |         44|0         |application_master:main_loop/2        |
|11 |<0.6.0>       |        6896| error_logger                     |        220|0         |gen_event:fetch_msg/5                 |
|12 |<0.23.0>      |        5760| proc_lib:init_p/5                |         81|0         |gen_server:loop/6                     |
|13 |<0.16.0>      |        5712| inet_db                          |        226|0         |gen_server:loop/6                     |
|14 |<0.25.0>      |        2784| user                             |         36|0         |group:server_loop/3                   |
|15 |<0.13.0>      |        2784| global_name_server               |         50|0         |gen_server:loop/6                     |
|16 |<0.22.0>      |        2744| standard_error                   |          9|0         |standard_error:server_loop/1          |
|17 |<0.21.0>      |        2744| standard_error_sup               |         41|0         |gen_server:loop/6                     |
|18 |<0.19.0>      |        2744| file_server_2                    |         81|0         |gen_server:loop/6                     |
|19 |<0.28.0>      |        2704| kernel_safe_sup                  |         58|0         |gen_server:loop/6                     |
|20 |<0.27.0>      |        2704| proc_lib:init_p/5                |        286|0         |gen_server:loop/6                     |
|INPUT: q(quit)  p(pause/unpause)  r/rr(reduction) m/mm(memory) b/bb(binary memory) t/tt(total heap  size) jpos(jump to process pos)|
```

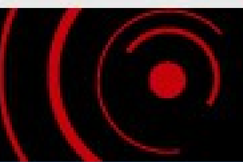
Measure, Don't Guess

- Your intuition may be wrong
- Don't optimize things that don't matter
- Optimize the hot path
- Driver for performance is often abuse use cases, e.g. DDOS



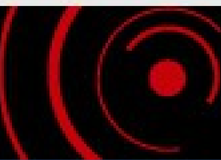
Lots of tools

- http://homeonrails.com/2016/05/profiling-in-erlang/
- http://www.snookles.com/erlang/ef2015/slf-presentation.html

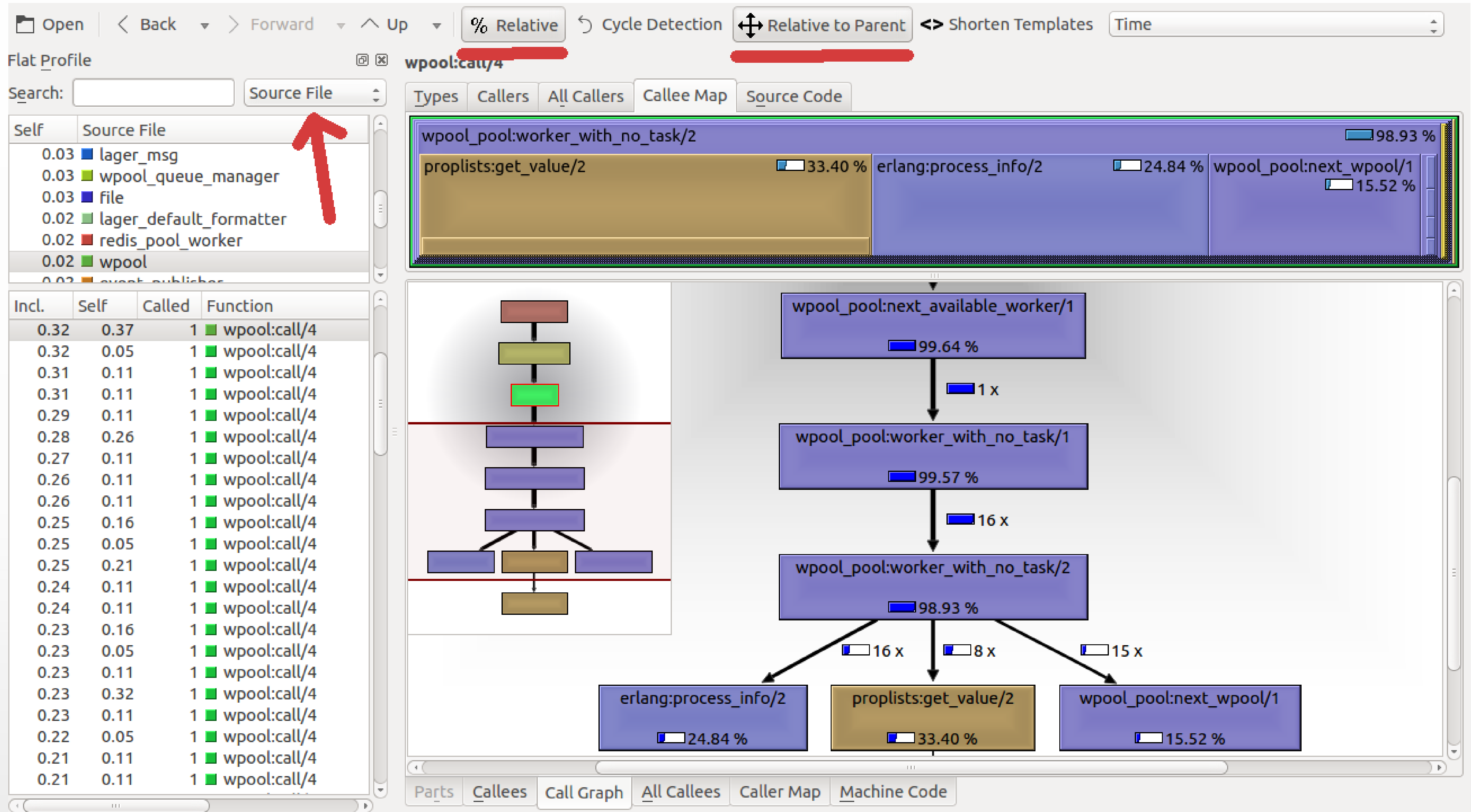


Lots of tools

- Micro:
  - timer:tc
  - Benchee: https://github.com/PragTob/benchee
- Macro
  - fprof
    - http://erlang.org/doc/man/fprof.html
    - https://github.com/isacssouza/erlgrind
    - brew install qcachegrind --with-graphviz
  - Flame graphs: https://github.com/slfritchie/eflame
- Tsung for load generation

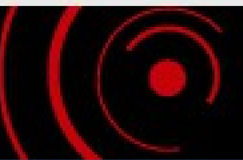


Fprof + erlgrind + cachegrind



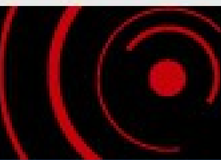
Surprising things: inspect

- Does a lot of work to introspect big data structures like conn
- Throws it away if debug message in production



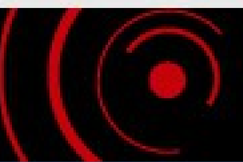
Surprising things: uuid generation

- Globally unique request id, e.g. 63edd89e-4f45-11e7-9424-2fc1a54ffaf3
- Depends on MAC address, time, pid, random number
- Lists all the network interfaces
- Reads the clock
- Stateful, by pid: use process dictionary
- Time went from worst case of 500 μs down to less than one μs



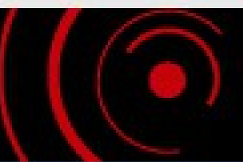
Surprising things: iolists

- Erlang I/O functions use more efficient OS functions (writev vs write). One reason Phoenix is so fast.
- "foo" <> "bar" vs ["foo", "bar"]
- Don't unnecessarily flatten data
- Make your APIs iolist friendly
- Law of leaky abstractions: https://www.joelonsoftware.com/2002/11/11/the-law-of-leaky-abstractions/
- https://www.bignerdranch.com/blog/elixir-and-io-lists-part-1-building-output-efficiently/
- http://www.evanmiller.org/elixir-ram-and-the-template-of-doom.html



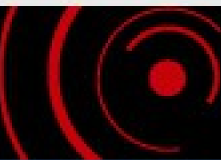
OS and TCP/IP Tuning: open files

- Increase number of open files for user, for OS as a whole
- Starts at 1024, much too small
- Ends at 4M :-)



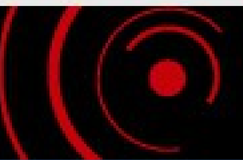
OS and TCP/IP Tuning

- Phoenix behind Nginx
- TCP connection is identified by four things
  - source ip + source port + destination ip + destination port
  - 127.0.0.1 + xxx + 127.0.0.1 + 4000
  - There are 64K ports, 16-bit integer
  - TCP/IP stack won't reuse a port for 2 x maximum segment lifetime = 2 minutes
  - 60000 ports / 120 sec = 500 requests per sec max
  - 1024 / 120 = 8.53 rps with default file handle limit
  - Symptom: app thinks everything is fine, but you measure latency at Nginx, you get some requests that take 5 sec waiting for a port
- Add HTTP "Connection: close" header, particularly for abuse



OS and TCP/IP Tuning

- http://theerlangelist.com/article/phoenix_latency
- http://www.phoenixframework.org/blog/the-road-to-2-million-websocket-connections



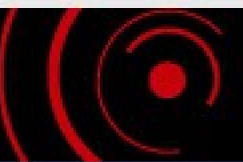
Erlang VM tuning

- Async threads: set +A parameter to at least 12 threads per core on which your node is deployed on. e.g. 128 on an 8 core

  +A 128

- kernel-poll = more efficient socket interface

  +K true



Questions?