Performance Optimization 101

Louis-Philippe Gauthier
Team leader @ AdGear Trader

adgear

Exercise

adgear

HTTP API server

API

GET /date - returns today's date
GET /time - returns the unix time in seconds

```
curl -i 'http://localhost:9090/date'
HTTP/1.1 200 OK
Content-Length: 10
Connection: Keep-Alive

2014-03-05
```

```
curl -i 'http://localhost:9090/time'
HTTP/1.1 200 OK
Content-Length: 10
Connection: Keep-Alive

1394059603
```

adgear

HTTP API server

TODO

- accepting connections
- parsing http requests
- routing
- building responses

adgear

HTTP API server

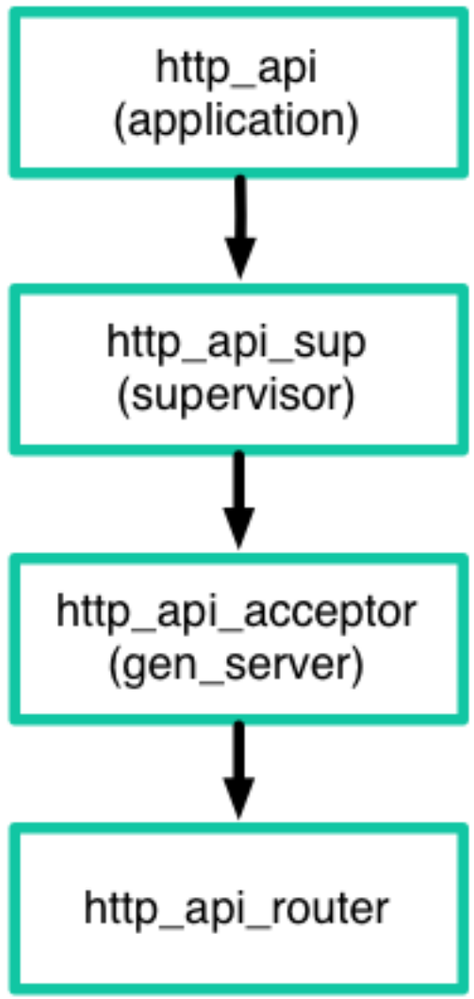
accepting connections

```
start() ->
    {ok, ListenSocket} = gen_tcp:listen(9090, []),
    accept_loop(ListenSocket).

accept_loop(ListenSocket) ->
    case gen_tcp:accept(ListenSocket) of
        {ok, Socket} ->
            Pid = spawn(?MODULE, handle_request, [Socket]),
            gen_tcp:controlling_process(Socket, Pid),
            accept_loop(ListenSocket);
        {error, Reason} ->
            exit(Reason)
    end.

handle_request(Socket) ->
    gen_tcp:close(Socket).
```

adgear

HTTP API server

accepting connections



adgear

HTTP API server

accepting connections

- gen_tcp:controlling_process/2 is slow
- spawn worker with ListenSocket
- worker accepts and ack's listener

adgear

HTTP API server

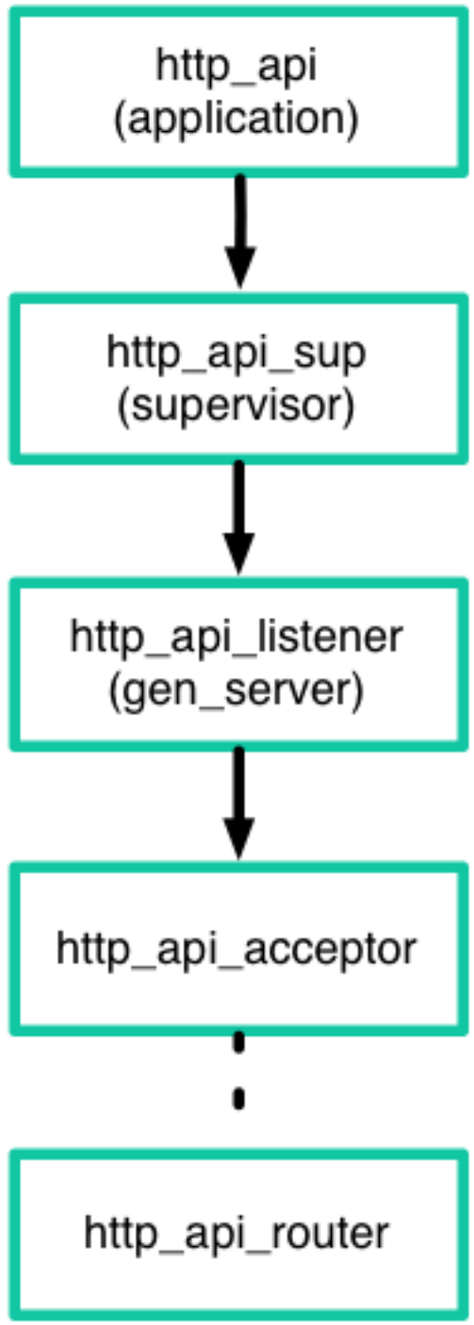
accepting connections

```
start() ->
    {ok, ListenSocket} = gen_tcp:listen(9090, []),
    self() ! {ack, self()},
    listener_loop(ListenSocket).

listener_loop(ListenSocket) ->
    Pid = self(),
    receive
        {ack, Pid} ->
            spawn(?MODULE, acceptor, [ListenSocket, Pid]),
            listener_loop(ListenSocket)
    end.

acceptor(ListenSocket, Pid) ->
    case gen_tcp:accept(ListenSocket) of
        {ok, Socket} ->
            Pid ! {ack, Pid},
            erlang:yield(),
            handle_request(Socket);
        {error, Reason} ->
            exit(Reason)
    end.
```

adgear

HTTP API server

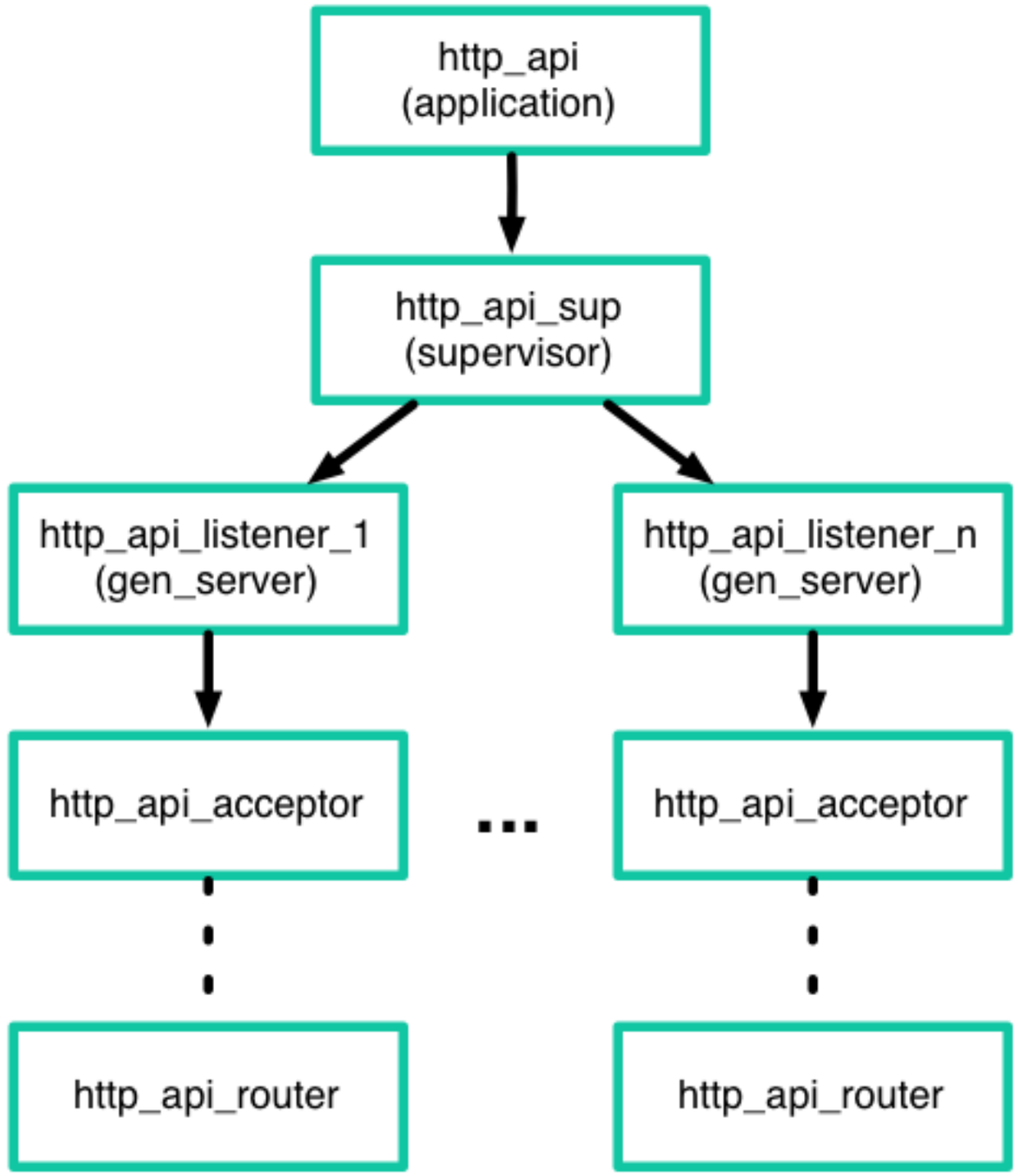
accepting connections



adgear

HTTP API server

accepting connections



adgear

HTTP API server

accepting connections

- use proc_lib instead of gen_server
- socket options:
  - binary
  - {backlog, 4196}
  - {raw, 6, 9, <<30:32/native>>}

adgear

HTTP API server

parsing request

```
request(Socket, Req) ->
    case gen_tcp:recv(Socket, 0, 30000) of
        {ok, {http_request, Method, Uri, Version}} ->
            headers(Socket, Req#req{
                method = Method,
                uri = Uri,
                http_vsn = Version
            }, []);
        {error, {http_error, "\r\n"}} ->
            request(Socket, Req);
        {error, {http_error, "\n"}} ->
            request(Socket, Req)
    end.

headers(Socket, Req, Headers) ->
    case gen_tcp:recv(Socket, 0, 30000) of
        {ok, {http_header, _, Header, _, Val}} ->
            headers(Socket, Req, [{Header, Val} | Headers]);
        {error, {http_error, "\r\n"}} ->
            headers(Socket, Req, H);
        {error, {http_error, "\n"}} ->
            headers(Socket, Req, H);
        {ok, http_eoh} ->
            body(Socket, Req#req{headers = lists:reverse(H)})
    end.
```

adgear

HTTP API server

parsing request

```
parse_method(<<"GET ", Rest/binary>>, Req) ->
    parse_uri(Rest, Req#req {method = 'GET'});
parse_method(<<"POST ", Rest/binary>>, Req) ->
    parse_uri(Rest, Req#req {method = 'POST'});
parse_method(_Else, Req) ->
    Req.

parse_uri(<<" ", Rest/binary>>, Req) ->
    parse_version(Rest, Req);
parse_uri(<<A:1/binary, Rest/binary>>, #req {uri = undefined} = Req) ->
    parse_uri(Rest, Req#req {uri = A});
parse_uri(<<A:1/binary, Rest/binary>>, #req {uri = Uri} = Req) ->
    parse_uri(Rest, Req#req {uri = <<Uri/binary, A/binary>>}).

parse_version(<<"HTTP/1.1\r\n", Rest/binary>>, Req) ->
    parse_headers(Rest, Req#req {http_vsn = {1,1}});
parse_version(<<"HTTP/1.0\r\n", Rest/binary>>, Req) ->
    parse_headers(Rest, Req#req {http_vsn = {1,0}});
parse_version(_Request, Req) ->
    Req.
```

adgear

HTTP API server

parsing request

- binary matching is very powerful!
- working with binaries is more memory efficient
  - binaries over 64 bytes are shared (not copied)
- faster than the built-in http parser (BIF) when running on many cores and using hipe
- keep state in a record
  - O(1) lookups

adgear

HTTP API server

routing

```
route(#req {method = Method, uri = Uri} = Req) ->
    route(Method, Uri, Req).

route('GET', <<"/date">>, Req) ->
    {ok, Req#req {
        status_code = 200,
        resp_headers = [{<<"Connection">>, <<"Keep-Alive">>}],
        resp_body = date_binary()
    }};
route('GET', <<"/time">>, Req) ->
    {ok, Req#req {
        status_code = 200,
        resp_headers = [{<<"Connection">>, <<"Keep-Alive">>}],
        resp_body = unix_time_binary())
    }};
route(_Method, _Uri, Req) ->
    {ok, Req#req {
        connection = close,
        status_code = 404,
        resp_body = <<"not found">>
    }}.
```

adgear

HTTP API server

routing

pattern matching is awesome!

adgear

HTTP API server

building response

```
date_binary() ->
    {Y, M, D} = date(),
    list_to_binary(integer_to_list(Y) ++ "-" ++
        integer_to_list_with_pad(M) ++ "-" ++
        integer_to_list_with_pad(D)).

integer_to_list_with_pad(Integer) when Integer < 10 ->
    "0" ++ integer_to_list(Integer);
integer_to_list_with_pad(Integer) ->
    integer_to_list(Integer).

unix_time_binary() ->
    {Mega, Sec, _Micro} = erlang:now(),
    integer_to_binary(Mega * 1000000 + Sec).
```

adgear

HTTP API server

building response

```
date_binary() ->
    {Y, M, D} = date(),
    <<(integer_to_binary(Y))/binary, "-",
      (integer_to_binary_with_pad(M))/binary, "-",
      (integer_to_binary_with_pad(D))/binary>>.

integer_to_binary_with_pad(Integer) when Integer < 10 ->
    <<"0", (integer_to_binary(Integer))/binary>>;
integer_to_binary_with_pad(Integer) ->
    integer_to_binary(Integer).

unix_time_binary() ->
    {Mega, Sec, _Micro} = os:timestamp(),
    integer_to_binary(Mega * 1000000 + Sec).
```

adgear

HTTP API server

building response

```
date_binary() ->
    [{date_binary, Date}] = ets:lookup(?CACHE, date_binary).
    Date.

unix_time_binary() ->
    [{unix_time_binary, Time}] = ets:lookup(?CACHE, unix_time_binary).
    Time.
```

adgear

HTTP API server

building response

- ETS is your friend!
- cache time date in ETS public table
  - {read_concurrency, true}
- if you store a binary over 64 bytes, it won't get copied!
- have a gen_server update the cache
  - every second for the time
  - every day for the date

adgear

HTTP API server

building response

- do not try to rewrite everything
- use community projects and contribute back!
- often your application will spend most of its time talking to external services
- premature optimization is usually bad

adgear

Gotchas

slow functions / modules

- erlang:now/0 vs os:timestamp/0
- proplists:get_value() vs lists:keyfind()
- timer:send_after() vs erlang:send_after()
- gen_udp:send() vs erlang:port_command()
- avoid erlang:controlling_process() if you can
- avoid base64, string, unicode modules

adgear

Tools

Profiling

info

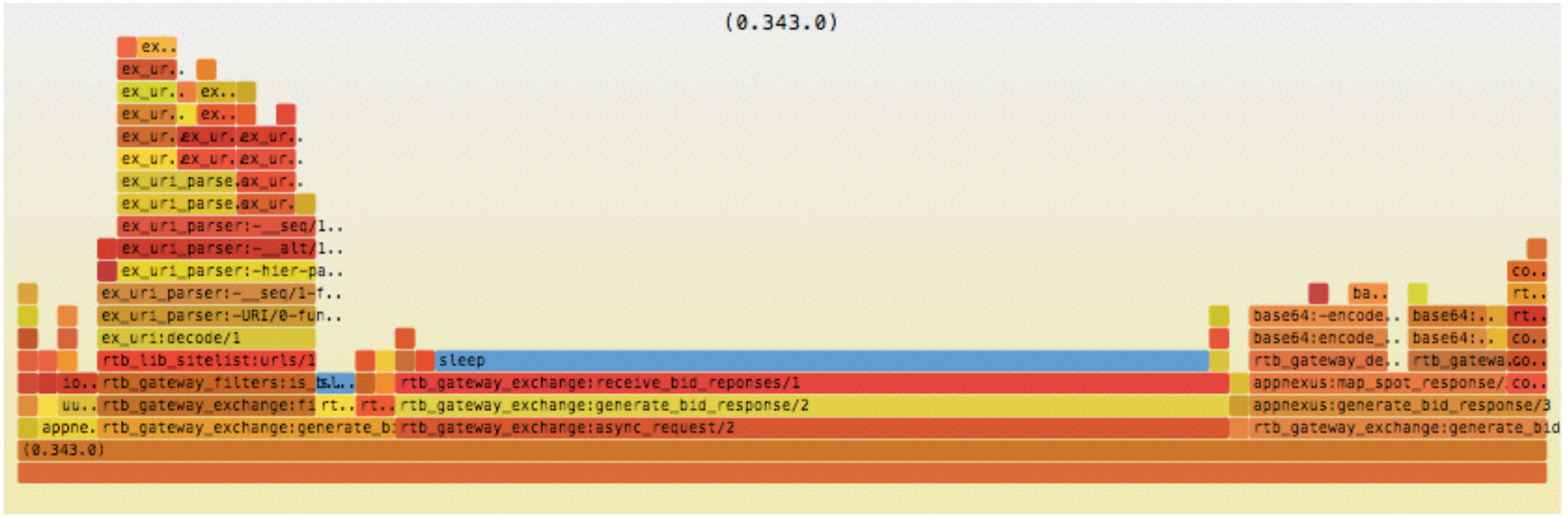
- useful to find slow code paths
- fprof
  - uses erlang:trace/3
  - output is really hard to understand
  - erlgrind to read in kcachegrind
- eflame
  - also uses erlang:trace/3
  - nice graphical output

adgear

Eflame

info

- flamechart.pl (from Joyent)
- makes it visually easy to find slow function calls



adgear

Eflame

how to

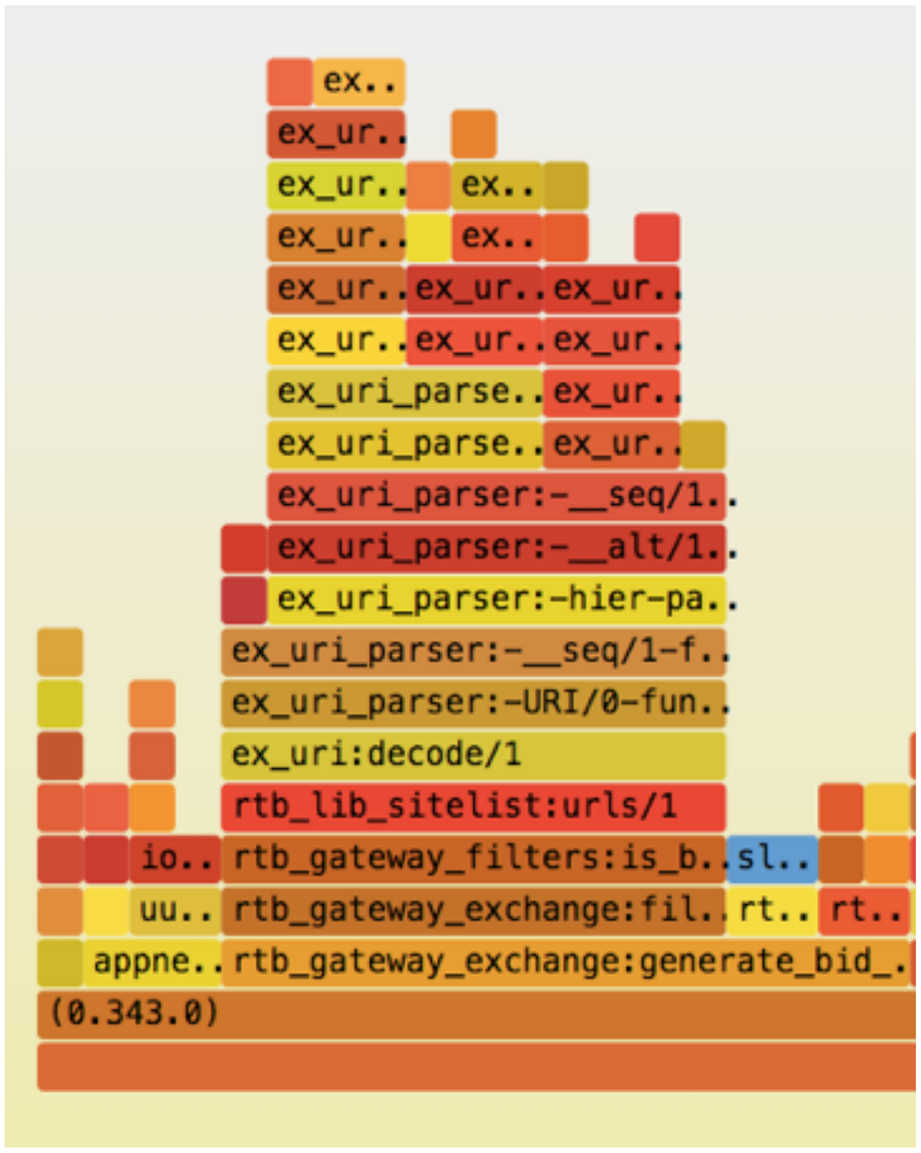
```
eflame:apply(normal_with_children, "stacks.out", M, F, A)

./stacks_to_flames.sh stacks.out
```

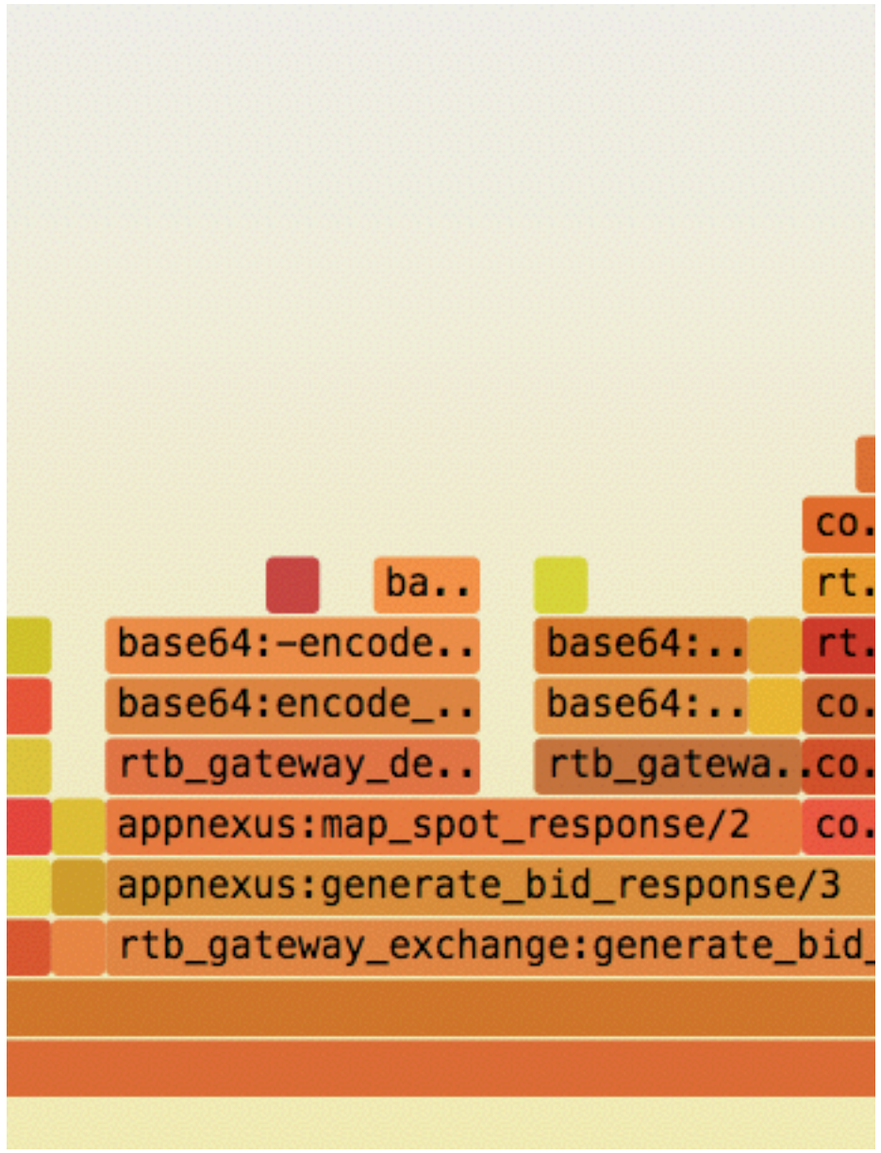
```
% eflame.erl
-define(DEFAULT_MODE, normal_with_children).
-define(DEFAULT_OUTPUT_FILE, "stacks.out").
-define(DEFAULT_RESOLUTION, 1). %% us
```

adgear

Eflame

info





adgear

Micro benchmarks

info

- start with profiling
- useful for experimentation and to validate hypothesis
- small benchmarking library called timing
  - uses the excellent bear (statistics) library

adgear

Micro benchmarks

how to

```
timing:function(fun () -> erlang:now() end, 100000, 1000).
[{min,0},
 {max,330339},
 {arithmetic_mean,664.4683888147711},
 {geometric_mean,12.826192995528386},
 {harmonic_mean,6.631306040286247},
 {median,15},
 {variance,107540428.49053556},
 {standard_deviation,10370.170128331336},
 {skewness,22.220345603769573},
 {kurtosis,566.1172633985375},
 {percentile,[{50,15},
              {75,17},
              {90,22},
              {95,33},
              {99,96},
              {999,202448}]},
{histogram,[{400,740829},
            {800,12},
            {1200,349},
            {1600,17},
            {2000,133},
            {2400,226},
            {2800,6},
            {4000,179},
            {5000,156},
            {6000,127},
            {7000,118},
            {8000,170},
            {9000,103},
            {10000,...},
            {...}|...]},
{n,748058}]
```

adgear

Micro benchmarks

info

# parallel processes	erlang:now/0	os:timestamp/0
1	0.99	0.87
10	22.87	2.54
100	168.23	16.99
1000	664.46	51.98

adgear

Hipe

info

- native, {hipe, [o3]}
- doesn't mix with NIFs
  - on_load
- switching between non-native and native code is expensive
  - different call stacks
  - might overload the code_server (bug?)
- —enable-native-libs
- hipe_bifs (sshhh)

adgear

Hipe

how to

```
5> Request = <<"GET /date HTTP/1.1\r\nUser-Agent: curl/7.30.0\r\nHost:
localhost:9090\r\nAccept: */*\r\n\r\n">>.

6> c(http_api_protocol).
{ok,http_api_protocol}

7> timing:function(fun () -> http_api_protocol:parse_request(Request) end).
{arithmetic_mean,46.83904100069493}.

8> c(http_api_protocol, [native, {hipe, [o3]}]).
{ok,http_api_protocol}

9> timing:function(fun () -> http_api_protocol:parse_request(Request) end).
{arithmetic_mean,24.6554829742877}.
```

adgear

NIFs

info

- function that is implemented in C instead of Erlang
- can be dangerous…
  - crash VM (segfault)
  - OOM (memory leak)
  - must return < 500 us (to be safe…)
- ideally should yield and use enif_consume_timeslice
  - what is a reduction?
- dirty schedulers (R17)
  - finally!

adgear

Process Tuning

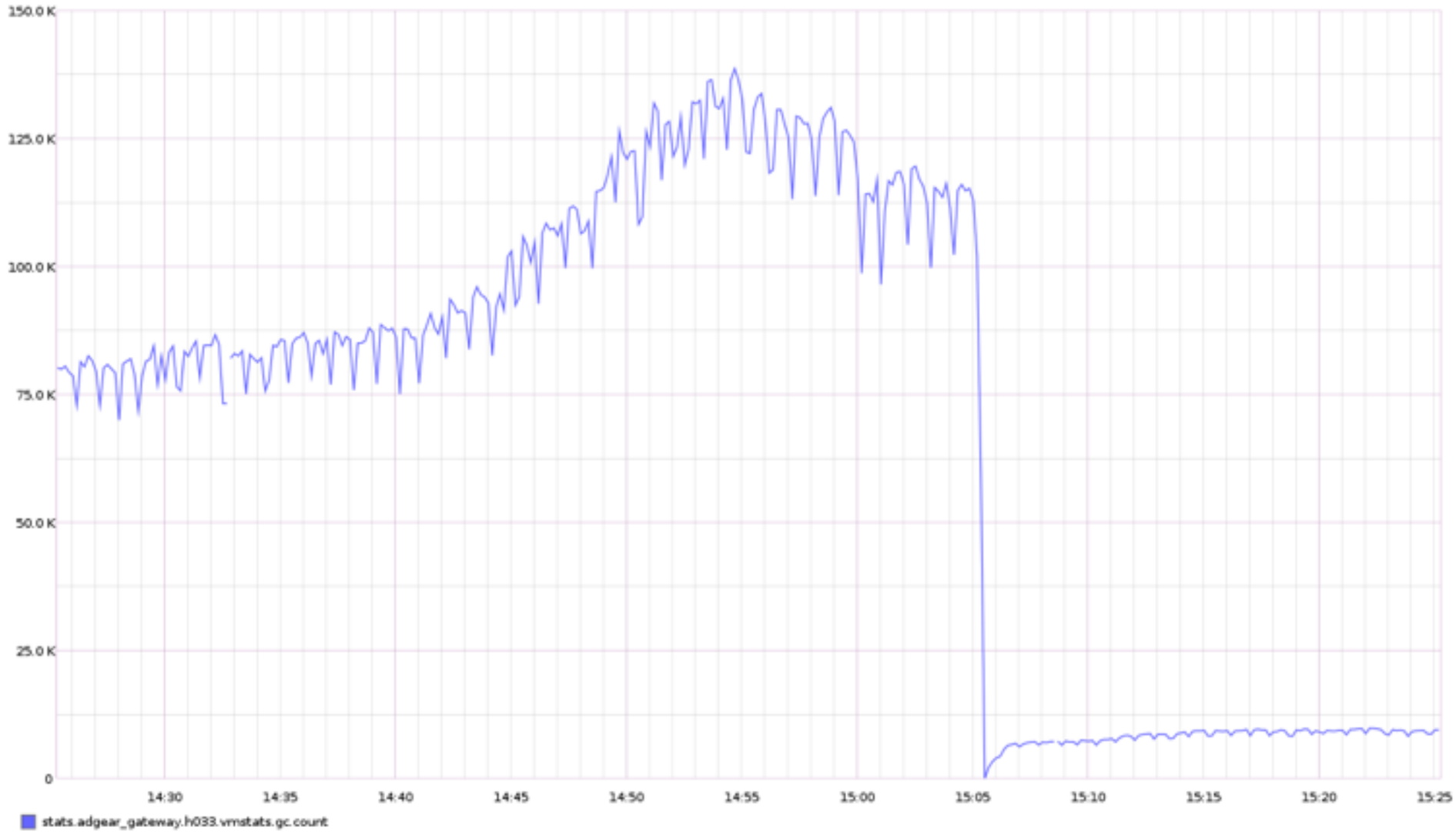
info

- tune min_heap_size on spawn
- fullsweep_after if you have memory issues
- force gc
- +hms (set default min_heap_size)

adgear

Process Tuning

info



adgear

Monitoring

info

- statsderl for application metrics
- vmstats for VM metrics
- system_stats for OS metrics
- erlang:system_monitor/2
- entop for live system exploration

adgear

Statsderl

info

- statsd client
- very cheap to call (async)
- offers 3 kinds of metrics:
  - counters - for counting (e.g QPS)
  - gauges - for absolute values (e.g. system memory)
  - timers - similar to gauges but with extra statistics

adgear

Statsderl

how to

```
statsderl:decrement("test.decrement", 1, 0.5).

statsderl:increment("test.increment", 10, 0.5).

statsderl:gauge("test.gauge", 333, 1.0).

Timestamp = os:timstamp(),
% code you want to time
timer:now_diff(os:timestamp(), Timestamp),
statsderl:timing("test.timing", 5, 0.5).

Result = statsderl:timing_fun("test.timing", fun() -> timer:sleep(100) end, 0.5).

Timestamp = erlang:now(),
% code you want to time
statsderl:timing_now("test.timing", Timestamp, 0.5).
```

adgear

VM Stats

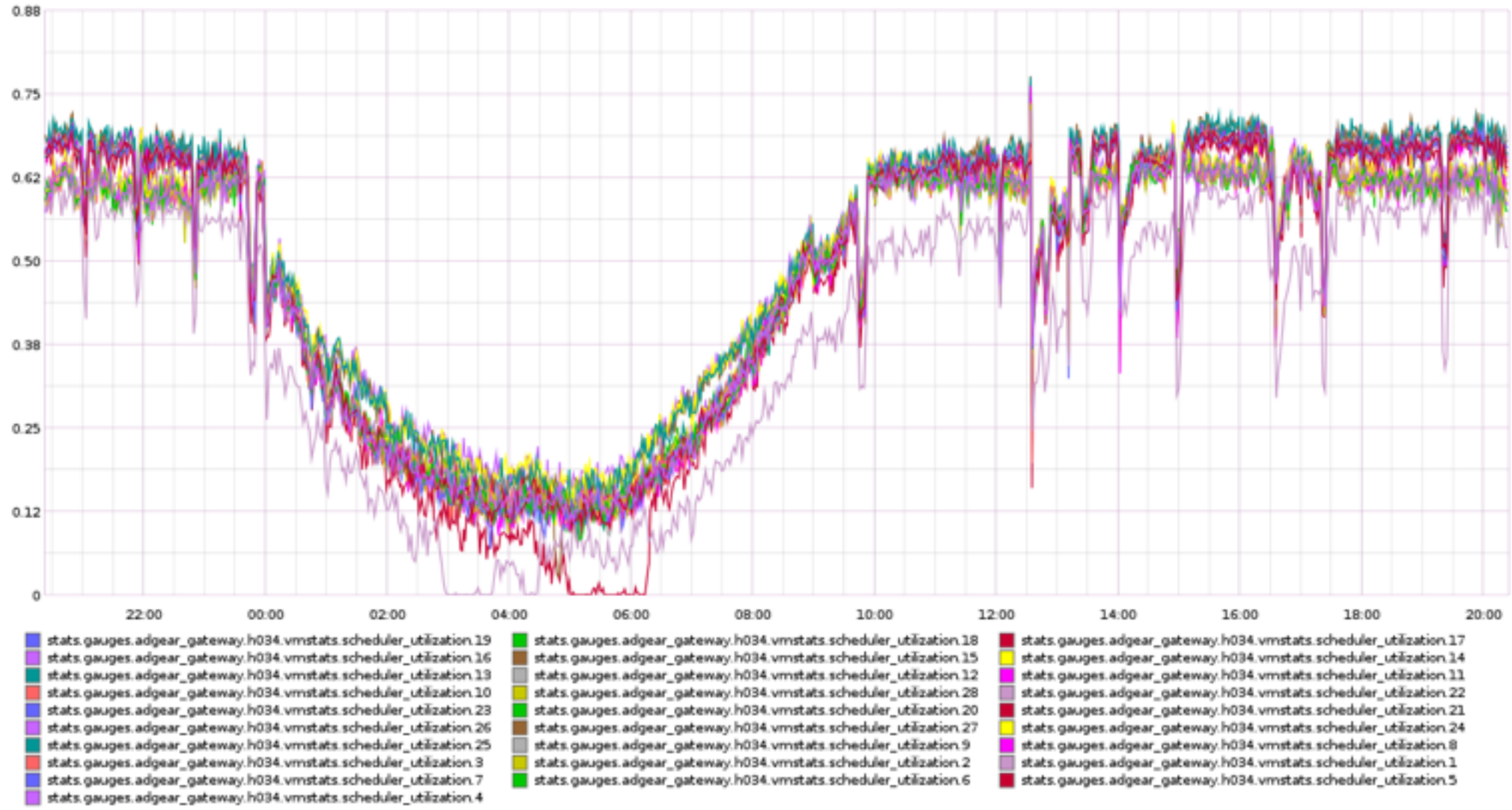
info

- process count
- messages in queues
- run queue length
- memory (total, proc_used, atom_used, binary, ETS)
- scheduler utilization (per scheduler)
- garbage collection (count, words reclaimed)
- reductions
- IO bytes (in/out)

adgear

VM Stats

info



adgear

System Stats

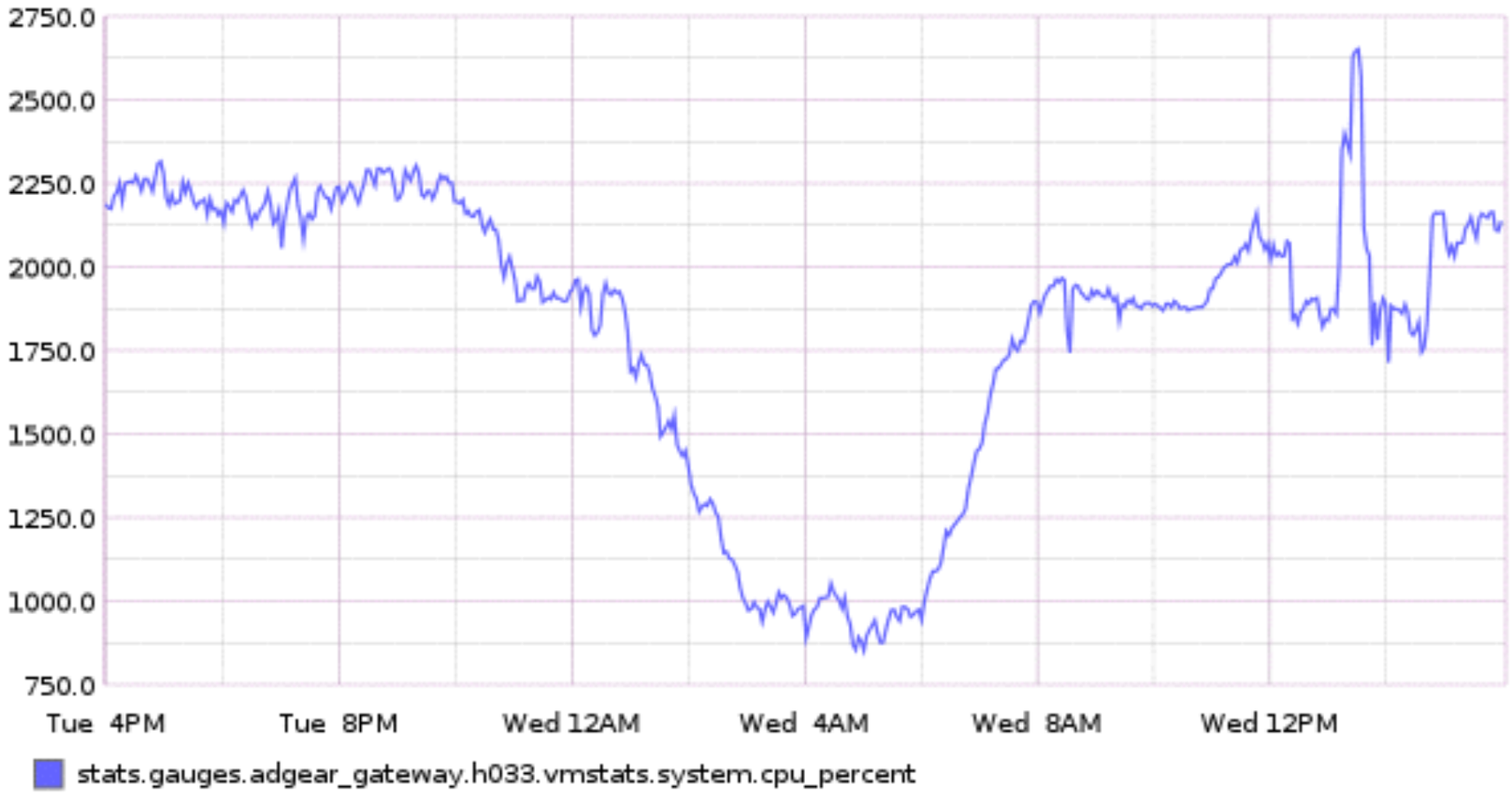
info

- load1, load5, load15
- cpu percent
  - can be misleading because of spinning schedulers
- virtual memory size
- resident memory size
  - very useful to track those OOM crashes

adgear

System Stats

info



adgear

System Monitor

info

- monitoring for:
  - busy_port
  - busy_dist_port
  - long_gc
  - long_schedule
  - large_heap
- riak_sysmon + lager / statsderl handler

adgear

System Monitor

how to

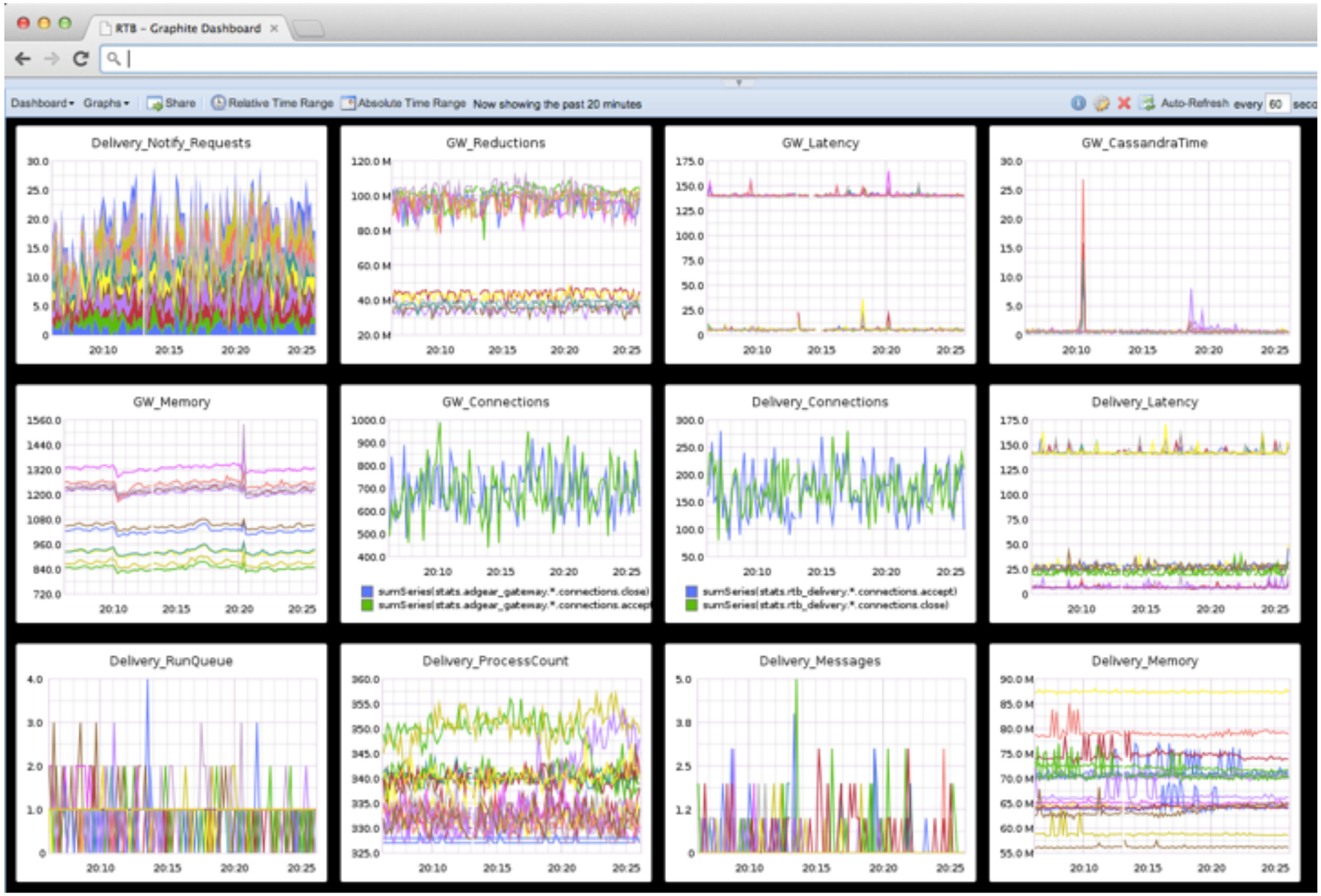
```
ok = gen_event:add_sup_handler(riak_sysmon_handler, rtb_lib_sysmon_handler, [])
```

```
handle_event({monitor, _Pid, long_schedule, Opts} = Event, #state{count = Count} = State) ->
    statsderl:increment(<<"sysmon.long_schedule">>, 1, 0.2),
    statsderl:timing(<<"sysmon.long_schedule">>, ?LOOKUP(timeout, Opts), 0.2),
    lager:warning("~p~n", [Event]),
    {ok, State#state{count = Count + 1}};
handle_event(Event, #state{count = Count} = State) ->
    lager:warning("~p~n", [Event]),
    {ok, State#state{count = Count + 1}}.
```

adgear

Dashboard

info



adgear

Entop

info

- top(1)-like tool for the Erlang VM
- can be used remotely
- gives per process:
  - pid / name
  - reductions
  - message queue length
  - heap size

adgear

Entop

info

```
./entop node_name -setcookie my_secret
```

```
Node: 'http_api@Louis-Philippes-MacBook-Air' (Connected) (R16B03/5.10.4) unix (darwin 1
Time: local time 16:00:31, up for 000:00:06:42, 0ms latency,
Processes: total 144 (RQ 2) at 5100659 RpI using 5103.2k (5137.1k allocated)
Memory: Sys 8469.7k, Atom 186.1k/197.7k, Bin 240.4k, Code 4443.6k, Ets 274.3k

Interval 1000ms, Sorting on "Reductions" (Descending), Retrieved in 2ms
        Pid Registered Name      Reductions   MQueue HSize  SSize  HTot
  <0.368.0> -                    946865       0      2586   7      2962
  <0.369.0> -                    932895       0      2586   7      2962
  <0.370.0> -                    903400       0      2586   7      2962
  <0.372.0> -                    879539       0      2586   7      2962
  <0.373.0> -                    848467       0      2586   7      2962
  <0.374.0> -                    820068       0      2586   7      2962
  <0.376.0> -                    787581       0      2586   7      2962
  <0.377.0> -                    759779       0      2586   7      2962
  <0.378.0> -                    711939       0      2586   7      2962
  <0.379.0> -                    687939       0      2586   7      2962
  <0.380.0> -                    623411       0      2586   7      2962
  <0.381.0> -                    594057       0      2586   7      2962
  <0.382.0> -                    516240       0      2586   7      2962
  <0.384.0> -                    470099       0      2586   7      2962
    <0.3.0> erl_prim_loader      395737       0      610    6      3196
  <0.385.0> -                    376834       0      2586   7      2962
  <0.386.0> -                    331733       0      2586   7      2962
  <0.387.0> -                    246482       0      2586   7      2962
  <0.389.0> -                    209602       0      2586   7      2962
  <0.390.0> -                    170531       0      2586   7      2962
  <0.396.0> -                    145771       0      2586   7      2962
   <0.25.0> code_server          122629       0      6772   3      17730
    <0.6.0> error_logger         70719        0      4185   9      10957
   <0.11.0> kernel_sup           49074        0      2586   9      20317
   <0.30.0> user                 42568        0      4185   4      8370
    <0.0.0> init                 12965        0      2586   2      3196
   <0.12.0> rex                  10966        0      28690  9      29066
   <0.32.0> -                    3840         0      10958  16     21916
    <0.7.0> application_controll 3692         0      1598   7      4184
  <0.50.0> http_api_sup          2642         0      2586   9      6771
```

adgear

VM Tuning

info

- +K true (kernel polling)
- +sct db (scheduler bind)
- +scl false (disable load distribution)
- +sfwi 500 (force sheduler wakeup NIFs)
- +spp true (port parallelism)
- +zdbbl (distribution buffer busy limit)
- test with production load (synthetic benchmarks can be misleading)

adgear

Cset

info

- tool to help create cpusets
- reduces non voluntary context-switches
- reserve first two CPUs for interrupts and background jobs
- reserve rest of CPUs for the Erlang VM
- linux only

```
           CPU0       CPU1
  0:         43          0   IO-APIC-edge      timer
  3:        551          0   IO-APIC-edge      serial
  8:         58          0   IO-APIC-edge      rtc0
  9:          0          0   IO-APIC-fasteoi   acpi
 10:   14650468          0   IO-APIC-edge      ipmi_si
 22:         40          0   IO-APIC-fasteoi   ehci_hcd:usb2
 23:        162          0   IO-APIC-fasteoi   ehci_hcd:usb1
104:          0          0   PCI-MSI-edge      aerdrv
105:          0          0   PCI-MSI-edge      aerdrv
106:          0          0   PCI-MSI-edge      aerdrv
107:          0          0   PCI-MSI-edge      aerdrv
108:          0          0   PCI-MSI-edge      aerdrv
109:          0          0   PCI-MSI-edge      aerdrv
113:          0          0   PCI-MSI-edge      aerdrv
114:    7677705          0   PCI-MSI-edge      megasas
115:    9340356          0   PCI-MSI-edge      megasas
116:     268980          0   PCI-MSI-edge      megasas
117:     337767          0   PCI-MSI-edge      megasas
118:     273028          0   PCI-MSI-edge      megasas
119:     346907          0   PCI-MSI-edge      megasas
120:     278645          0   PCI-MSI-edge      megasas
121:     344484          0   PCI-MSI-edge      megasas
122:     278411          0   PCI-MSI-edge      megasas
123:     353350          0   PCI-MSI-edge      megasas
124:     294236          0   PCI-MSI-edge      megasas
125:     367201          0   PCI-MSI-edge      megasas
126:    2120612          0   PCI-MSI-edge      megasas
127:    4263222          0   PCI-MSI-edge      megasas
128:    2283602          0   PCI-MSI-edge      megasas
129:    4525298          0   PCI-MSI-edge      megasas
130: 2645884022          0   PCI-MSI-edge      eno1-0
131: 2691589528          0   PCI-MSI-edge      eno1-1
132: 2421966360          0   PCI-MSI-edge      eno1-2
133: 2369964707          0   PCI-MSI-edge      eno1-3
134: 3864773064          0   PCI-MSI-edge      eno1-4
NMI:          0          0   Non-maskable interrupts
LOC:   58208657   75199933   Local timer interrupts
SPU:          0          0   Spurious interrupts
PMI:          0          0   Performance monitoring interrupts
IWI:     235378     276891   IRQ work interrupts
RTR:         31          0   APIC ICR read retries
RES: 1566991201   57717439   Rescheduling interrupts
CAL: 4294967292    9123328   Function call interrupts
TLB:      53267      80252   TLB shootdowns
TRM:          0          0   Thermal event interrupts
THR:          0          0   Threshold APIC interrupts
MCE:          0          0   Machine check exceptions
MCP:       5472       5472   Machine check polls
```

adgear

Cpuset

how to

```
cset shield -c 1,3-15,17,19-31
/usr/bin/cset shield -e your_program
```

```
pidstat -w -p 11021 -t -T ALL 1 | awk '{if ($5 > 0.0) print}'

16:15:35          TGID       TID   cswch/s nvcswch/s  Command
16:15:36             -     11038   1017.00    313.00  |__beam.smp
16:15:36             -     11039   2522.00   1140.00  |__beam.smp
16:15:36             -     11040   1003.00    411.00  |__beam.smp
16:15:36             -     11041   2427.00    485.00  |__beam.smp
16:15:36             -     11042    959.00    680.00  |__beam.smp
16:15:36             -     11043   2410.00    686.00  |__beam.smp
16:15:36             -     11044    989.00    432.00  |__beam.smp
16:15:36             -     11045   2335.00    287.00  |__beam.smp
16:15:36             -     11046   1029.00    359.00  |__beam.smp
16:15:36             -     11047   2417.00    433.00  |__beam.smp
16:15:36             -     11048   1050.00   1252.00  |__beam.smp
16:15:36             -     11049   2770.00    434.00  |__beam.smp
16:15:36             -     11050   1026.00    621.00  |__beam.smp
16:15:36             -     11051   3133.00    886.00  |__beam.smp
16:15:36             -     11052    921.00   1504.00  |__beam.smp
16:15:36             -     11053   3564.00   1553.00  |__beam.smp
16:15:36             -     11054   1047.00    105.00  |__beam.smp
16:15:36             -     11055   2425.00    255.00  |__beam.smp
16:15:36             -     11056   1066.00    298.00  |__beam.smp
16:15:36             -     11057   2347.00    371.00  |__beam.smp
16:15:36             -     11058    965.00     24.00  |__beam.smp
16:15:36             -     11059   2319.00    458.00  |__beam.smp
16:15:36             -     11060   1055.00    327.00  |__beam.smp
16:15:36             -     11061   2379.00    447.00  |__beam.smp
16:15:36             -     11062   1032.00    151.00  |__beam.smp
16:15:36             -     11063   2313.00    213.00  |__beam.smp
16:15:36             -     11064   1665.00   4921.00  |__beam.smp
16:15:36             -     11065    973.00   2597.00  |__beam.smp
```

```
pidstat -w -p 11021 -t -T ALL 1 | awk '{if ($5 > 0.0) print}'

19:02:15          TGID       TID   cswch/s nvcswch/s  Command
19:02:16             -     13794      5.00      1.00  |__beam.smp
19:02:16             -     13800   1029.00     45.00  |__beam.smp
19:02:16             -     13801    984.00     15.00  |__beam.smp
19:02:16             -     13802    972.00     36.00  |__beam.smp
19:02:16             -     13803   1028.00     11.00  |__beam.smp
19:02:16             -     13804   1048.00     38.00  |__beam.smp
19:02:16             -     13805   1018.00     24.00  |__beam.smp
19:02:16             -     13806   1047.00     26.00  |__beam.smp
19:02:16             -     13807    997.00     31.00  |__beam.smp
19:02:16             -     13808   1023.00     52.00  |__beam.smp
19:02:16             -     13809   1016.00     22.00  |__beam.smp
19:02:16             -     13810    993.00     38.00  |__beam.smp
19:02:16             -     13811   1073.00     28.00  |__beam.smp
19:02:16             -     13812   1072.00      6.00  |__beam.smp
19:02:16             -     13813   1056.00      9.00  |__beam.smp
19:02:16             -     13814   1110.00      1.00  |__beam.smp
19:02:16             -     13815   1064.00      3.00  |__beam.smp
19:02:16             -     13816   1175.00     10.00  |__beam.smp
19:02:16             -     13817   1147.00      4.00  |__beam.smp
19:02:16             -     13818   1129.00      1.00  |__beam.smp
19:02:16             -     13819   1084.00      1.00  |__beam.smp
19:02:16             -     13820   1119.00      5.00  |__beam.smp
19:02:16             -     13821   1124.00      5.00  |__beam.smp
19:02:16             -     13822   1141.00      2.00  |__beam.smp
19:02:16             -     13823   1073.00      1.00  |__beam.smp
19:02:16             -     13824   1072.00      2.00  |__beam.smp
19:02:16             -     13825   1109.00      2.00  |__beam.smp
19:02:16             -     13826   1003.00      2.00  |__beam.smp
19:02:16             -     13827   1110.00      1.00  |__beam.smp
```

adgear

Lock counter

info

```
lock_count() ->
    lcnt:start(),
    lcnt:rt_opt({copy_save, true}),
    lcnt:clear(),
    timer:sleep(250),
    lcnt:collect(),
    lcnt:swap_pid_keys(),
    lcnt:conflicts().
```

lock	id	#tries	#collisions	collisions [%]	time [us]	duration [%]
timeofday	1	51583	11273	21.8541	38909	15.4867
pollset	1	12810	1666	13.0055	32772	13.0440
process_table	1	9777	815	8.3359	23984	9.5462
run_queue	32	193060	3639	1.8849	9679	3.8525
drv_ev_state	16	12703	254	1.9995	4873	1.9396
hipe_mfait_lock	1	29462	1910	6.4829	4111	1.6363
timer_wheel	1	19376	805	4.1546	2677	1.0655
proc_link	11146	31243	104	0.3329	677	0.2695
proc_main	11146	77526	856	1.1041	626	0.2492
proc_msgq	11146	62390	107	0.1715	396	0.1576
proc_status	11146	97533	40	0.0410	186	0.0740
bif_timers	1	728	14	1.9231	54	0.0215
sys_msg_q	1	25	1	4.0000	27	0.0107
make_ref	1	4801	37	0.7707	21	0.0084
pollset_rm_list	1	10267	20	0.1948	12	0.0048
db_hash_slot	384	4362	4	0.0917	10	0.0040

adgear

Other tools

info

- system limits
  - ulimit -n
  - sysctl
- dtrace / systemtap
  - application + OS tracing

adgear

Links

info

- https://github.com/proger/eflame
- https://github.com/lpgauth/timing
- https://github.com/lpgauth/statsderl
- https://github.com/ferd/vmstats
- https://github.com/lpgauth/system-stats
- https://github.com/mazenharake/entop
- https://github.com/ratelle/cpuset

adgear

Thank you!

github: lpgauth
irc: lpgauth (@erlounge)
twitter: lpgauth

adgear