Dynamic tracing in Erlang using DTrace/systemtap

Paweł Chrząszcz
pawel.chrzaszcz@erlang-solutions.com

Dynamic tracing tools

- **DTrace** – dynamic tracing framework
  - Works on Solaris, Mac OS X, FreeBSD, NetBSD, Oracle Linux
- **systemtap** – a tracing and probing tool
  - The Linux answer to Dtrace
  - Works on multiple Linux distributions (Red Hat, CentOS)
- These tools can be used to debug and monitor Erlang systems
  - Very minimal overhead – can be used on production
  - Capable of debugging and profiling the Erlang VM code

Erlang SOLUTIONS

Debugging with systemtap: problem

Problem example: a bug in Ejabberd

- Steps to reproduce
  - 10000+ users connected using TLS
  - They all disconnect at the same time
- The server becomes unresponsive for a long time
  - Up to 10 minutes for 100k users, a few hours for 1M users (!)
- How to debug it?
  - *Dbg* or *redbug* will slow down the system even more and not give any valuable results.
  - Erlang profiling tools (*fprof*, *eep*) seem to give some results... but they are incorrect
  - You can use full C-level debugging (gdb)... but it is not easy
  - Observer/etop are useless (cannot connect to node)
  - Monitoring can give some overview...

Erlang SOLUTIONS

Debugging with systemtap: solving it...

- Debugging using systemtap (Erlang has to be compiled with -g)
  - Suspicious part: processes exiting
  - Looking for long executing functions using global locks

```
Void erts_continue_exit_process(Process *p)
{

(...)

if (p->flags & F_USING_DDLL) {
    erts_ddll_proc_dead(p, ERTS_PROC_LOCK_MAIN);
    p->flags &= ~F_USING_DDLL;
    }

(...)

}
```

Erlang SOLUTIONS

Debugging with systemtap: the script

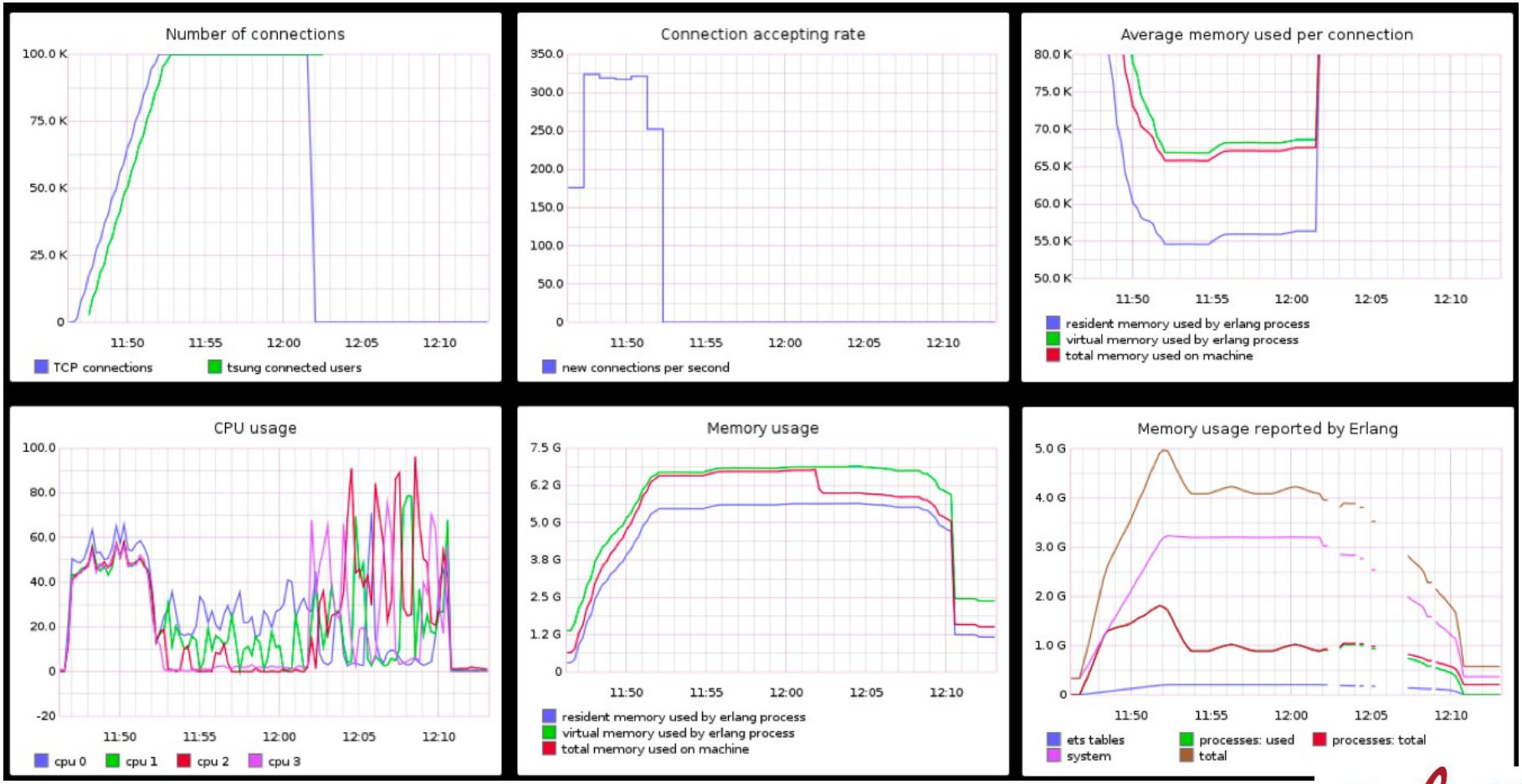
- The systemtap script

```
global start, intervals

probe process("/home/tapir/erlang/r16b02_stap/lib/erlang/erts-
5.10.3/bin/beam.smp").function(@1) {
  t = gettimeofday_us()
  printf("call: %d %s\n", t, $$parms)
  start[tid()] = t
}

probe process("/home/tapir/erlang/r16b02_stap/lib/erlang/erts-
5.10.3/bin/beam.smp").function(@1).return {
  t = gettimeofday_us()
  printf("return: %d\n", t)
  old_t = start[tid()]
  if (old_t) intervals <<< t - old_t
  delete start[tid()]
}

probe end {
  printf("intervals min:%dus avg:%dus max:%dus count:%d total:%dus\n",
         @min(intervals), @avg(intervals), @max(intervals),
         @count(intervals), @sum(intervals))
  print(@hist_log(intervals));
}
```

Erlang SOLUTIONS

Debugging with systemtap: results

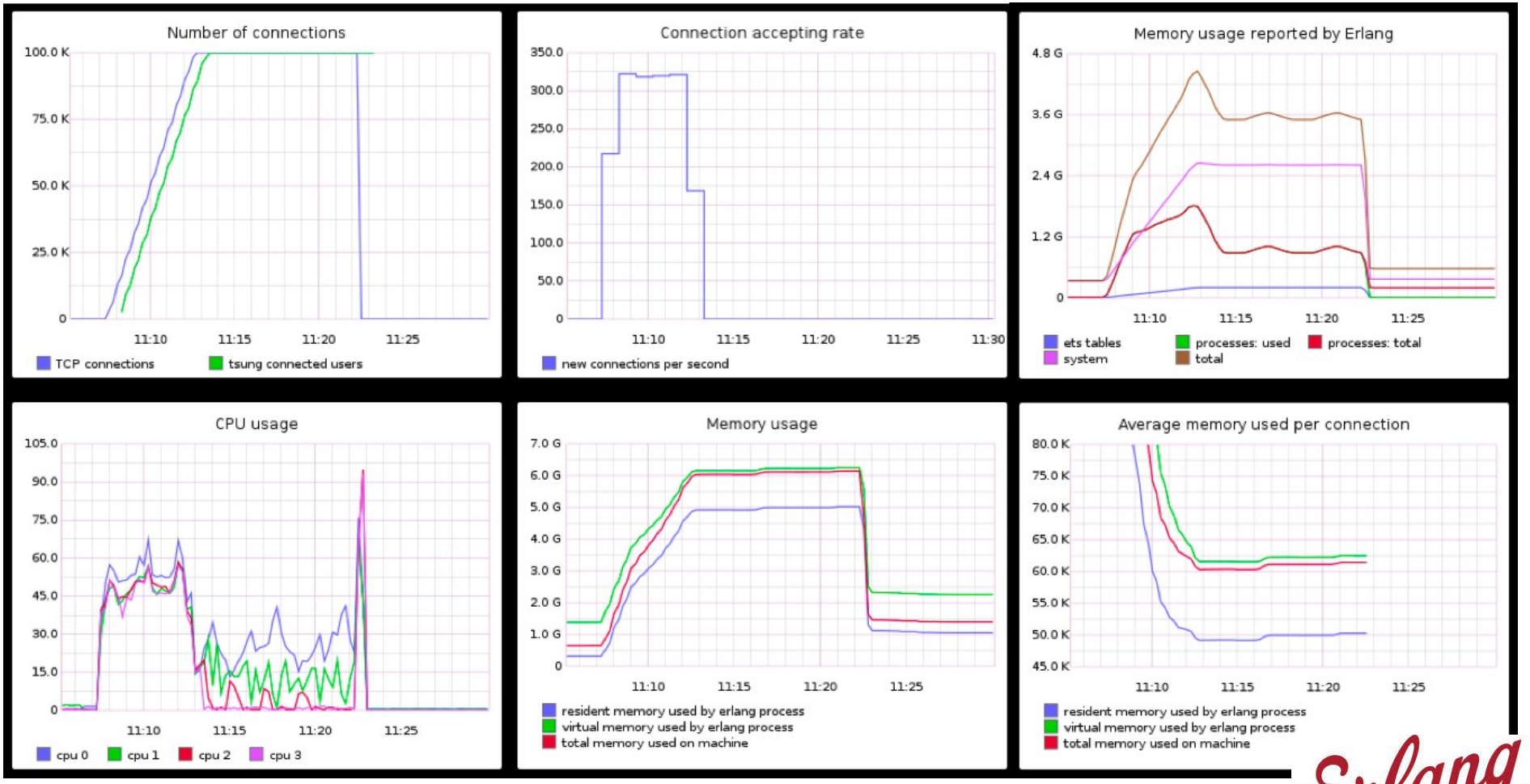
Before applying the fix ...



Erlang SOLUTIONS

Debugging with systemtap: results

… and after it:



Erlang SOLUTIONS

Debugging with systemtap: the fix

Showing **1 changed file** with **0 additions** and **8 deletions**

src/tls/tls.erl

src/tls/tls.erl

```
...  ...  @@ -81,10 +81,6 @@
 81   81      gen_server:start_link({local, ?MODULE}, ?MODULE, [], []).
 82   82
 83   83   init([]) ->
 84        -      case erl_ddll:load_driver(ejabberd:get_so_path(), tls_drv) of
 85        -          ok -> ok;
 86        -          {error, already_loaded} -> ok
 87        -      end,
 88   84      Port = open_port({spawn, "tls_drv"}, [binary]),
 89   85      Res = port_control(Port, ?SET_CERTIFICATE_FILE_ACCEPT, "./ssl.pem" ++ [0]),
 90   86      case Res of
...  ...  @@ -127,10 +123,6 @@
127  123   tcp_to_tls(TCPSocket, Options) ->
128  124      case lists:keysearch(certfile, 1, Options) of
129  125          {value, {certfile, CertFile}} ->
130       -              case erl_ddll:load_driver(ejabberd:get_so_path(), tls_drv) of
131       -                  ok -> ok;
132       -                  {error, already_loaded} -> ok
133       -              end,
134  126              Port = open_port({spawn, "tls_drv"}, [binary]),
135  127              Flags =
136  128                  case lists:member(verify_none, Options) of
```

Erlang SOLUTIONS

Dynamic tracing: Erlang probes

- **Erlang/OTP** is instrumented with Dtrace/systemtap probes since R15B01
  - It has to be compiled either *--with-dynamic-trace=dtrace* or *–with-dynamic-trace=systemtap*

- The support is considered experimental
  - What does it mean?
  - After testing we found out:
    - It does compile... and often does not work
    - The bugs suggest that nobody has even tried to run (some parts of) it

- There is an ongoing attempt to fix all the bugs in the VM probes
  - https://github.com/chrzaszcz/otp

Erlang SOLUTIONS

Dynamic tracing in Erlang - example

Problem example:

- Count messages for each (sender, receiver) pair
- Should work on Solaris, Mac OS X, FreeBSD and Linux.
- Lowest possible impact on the system.

Solution:

- Write the DTrace and systemtap scripts
- Start DTrace/systemtap, play with the Erlang node
- Stop DTrace/systemtap, read (or parse) the results

Issues:

- It is a tedious task to write and maintain both scripts.
- Parsing the results and starting/stopping DTrace/systemtap is also a hassle

Erlang SOLUTIONS

tracerl makes dynamic tracing easy

- https://github.com/esl/tracerl

- Write the script specification as an Erlang term
- Call `tracerl:start_trace(ScriptSpecification, TracedNode, Handler)`
- Tracerl takes care of the rest:
  - generates the DTrace/systemtap script for you
  - runs DTrace/systemtap automatically
  - receives the results and parses them to Erlang terms
  - provides the resulting terms to your handler (fun or pid)

Erlang SOLUTIONS

tracerl example

Let's solve the problem we have mentioned before with tracerl.

```
ScriptSpec = [
     {probe, 'begin',[{print_term, start}]},
     {probe, "message-send",
      [{count, msg, [sender_pid, receiver_pid]}]},
     {probe, 'end',
      [{print_term, {sent, '$1'},
        [{stat, {"%s", "%s", "%@d"}, msg}]}
      ]}
],

{ok, Pid} = tracerl:start_link(ScriptSpec, Node, self()).
```

Erlang SOLUTIONS

tracerl example: results

Let's wait until the dynamic tracing is up and running.

```
receive start -> ok end.
```

Now we send some messages on the monitored node...

```
tracerl:stop(Pid).

receive Stat -> Stat end.

{sent,[stat,
        {"<0.268.0>","<0.267.0>",5},
        {"<0.268.0>","<0.268.0>",1},
        {"<0.268.0>","<0.266.0>",10},
        {"<0.269.0>","<0.266.0>",20},
        {"<0.269.0>","<0.267.0>",5},
        {"<0.269.0>","<0.269.0>",1}]}
```

Erlang SOLUTIONS

tracerl example: generated DTrace script

```
erlang836:::process-exit
/ copyinstr(arg0) == "<0.239.0>" /
{
  exit(0);
}

BEGIN {
  printf("start.\n");
}

erlang836:::message-send
{
  @msg[copyinstr(arg0), copyinstr(arg1)] = count();
}

END {
  printf("{sent,[stat");
  printa(",{\"%s\",\"%s\",%@d}", @msg);
  printf("]}.\n");
}
```

Erlang SOLUTIONS

tracerl example: generated systemtap script

```
global msg
probe process("/home/tapir/erlang/r16b01_stap/erts-5.10.2/bin/beam")
.mark("process-exit") {
  if (pid() == 4857 && (user_string($arg1) == "<0.272.0>")) {
    exit()
  }
}
probe begin {
  printf("start.\n")
}
probe process("/home/tapir/erlang/r16b01_stap/erts-
5.10.2/bin/beam").mark("message-send") {
  if (pid() == 4857) {
    msg[user_string($arg1), user_string($arg2)] <<< 1
  }
}
probe end {
  printf("{sent,[stat")
  foreach([key1,key2] in msg)
    printf(",{\"%s\",\"%s\",%d}", key1, key2, @count(msg[key1,key2]))
  printf("]}.\n");
}
```

A short demo...

Conclusions

- DTrace and systemtap can be used to debug and profile production Erlang systems
- They can be used to debug the Erlang/OTP source code
- Erlang has built-in dynamic probes, but they are experimental
  - There is an ongoing effort at ESL to fix them
    - https://github.com/chrzaszcz/otp
- Using the probes directly is a tedious task
  - Tracerl aims at simplifying using them.
    - https://github.com/esl/tracerl

Dynamic tracing in Erlang using DTrace/systemtap

Paweł Chrząszcz
pawel.chrzaszcz@erlang-solutions.com

Thank You!

Erlang SOLUTIONS