Profiling Applications using DTrace

Mark Allen
mrallen1@yahoo.com
@bytemeorg
https://github.com/mrallen1
https://speakerdeck.com/mrallen1

What is DTrace?

Dynamic Tracing

"Observability technology"[1]

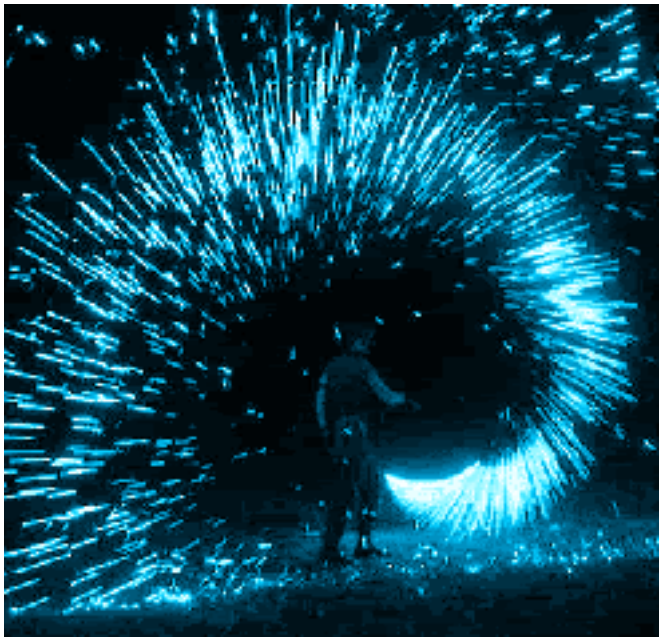
[1] *DTrace: Dynamic Tracing in Oracle Solaris*, Prentice Hall, 2nd ed, 2012.

Dynamic versus

```
time_it(Module, Func, Args) ->
  S = now(),
  Result = Module:Func(Args),
  log_elapsed_time(S, now()),
  Result.
```

This OTP Life

When I show dtrace & dynamic tracing in production to ruby on rails users



February 25, 2014

http://tmblr.co/ZqJL4s18SkeBK

I work at Alert Logic*.

I write a lot of Erlang for work and sometimes its nice to know why an application is doing that one thing that seems to talk a long time.

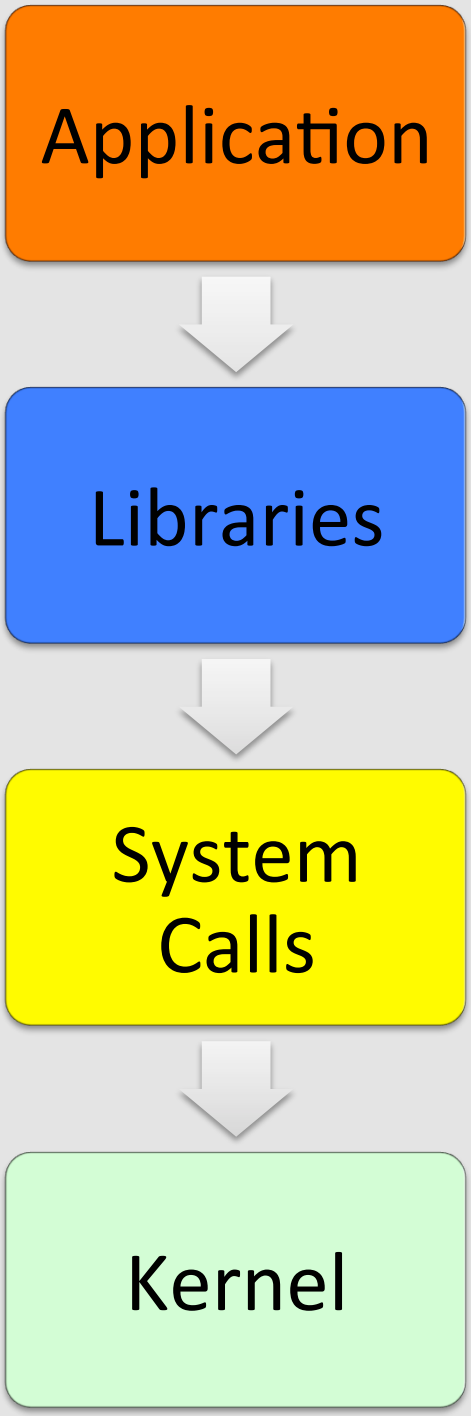
* We're hiring.

DTrace vs. profiler

DTrace ~~vs.~~ and profiler

DTrace vs. debugger

DTrace ~~vs.~~ and debugger


Application
Libraries
System Calls
Kernel

http://flic.kr/p/aQDPrZ

It's turtles all the way down...

DTrace enabled OSes:

- Solaris 10+
- ~~OpenSolaris~~Illumos
  - SmartOS (Joyent)
  - OmniOS (OmniTI)
- FreeBSD
- Mac OS X

What about Linux?

It's Complicated.
Sorry.

DTrace Terms

Provider

Manages probes in a subsystem

Provider

In our case, this is the application language itself.

(More on this soon)

Probes

DTrace instrumentation or "observables"

Consumer

A user-mode program that calls into DTrace

D Language Basics

D Language Overview

- awk-like
  - Define a probe, optional predicate and optional actions in a braced clause
  - Supports BEGIN and END blocks
  - Local variables (**`this->foo = 42`**)
  - Aggregate/associative variables (prefixed with @)
  - One liner support in the form

```
dtrace -n 'probe /predicate/ {action}'
```

Example

```
#!/usr/sbin/dtrace -qs

erlang*:::process-spawn
{
    printf("pid %s mfa %s\n", copyinstr(arg0), copyinstr(arg1));
}

erlang*:::process-exit
{
    printf("pid %s reason %s\n", copyinstr(arg0), copyinstr(arg1));
}

erlang*:::process-exit_signal
{
    printf("sender %s -> pid %s reason %s\n",
       copyinstr(arg0), copyinstr(arg1), copyinstr(arg2));
}
```

erl shell input

```
mallen-rMBP:~ mallen$ erl
Erlang R16B01 (erts-5.10.2) [source] [smp:8:8] [async-threads:10] [hipe] [kernel
-poll:false] [dtrace]

Eshell V5.10.2  (abort with ^G)
1> dyntrace:available().
true
2> F = fun() -> timer:sleep(2000) end.
#Fun<erl_eval.20.80484245>
3> spawn(fun() -> timer:sleep(2000) end).
<0.37.0>
4> spawn(fun() -> timer:sleep(2000) end).
<0.39.0>
5> spawn(fun() -> timer:sleep(2000) end).
<0.41.0>
```

spawn-exit.d output

```
mallen-rMBP:~ mallen$ pid <0.35.0> reason {ok,[{match,1,{var,1,'F'},{fun,1,{clau
ses,[{clause,1,□,□,[{call,1,{remote,1,{atom,1,timer},{atom,1,sleep}},[{integer
,1,2000}]}]}]}}}}]}
sender <0.35.0> -> pid <0.26.0> reason {o
pid <0.36.0> mfa erlang:apply/2
pid <0.36.0> reason {ok,[{call,1,{atom,1,spawn},[{fun,1,{clauses,[{clause,1,□,[
],[{call,1,{remote,1,{atom,1,timer},{atom,1,sleep}},[{integer,1,2000}]}]}]}}]}]}
sender <0.36.0> -> pid <0.26.0> reason {o
pid <0.37.0> mfa erlang:apply/2
pid <0.38.0> mfa erlang:apply/2
pid <0.38.0> reason {ok,[{call,1,{atom,1,spawn},[{fun,1,{clauses,[{clause,1,□,[
],[{call,1,{remote,1,{atom,1,timer},{atom,1,sleep}},[{integer,1,2000}]}]}]}}]}]}
sender <0.38.0> -> pid <0.26.0> reason {o
pid <0.39.0> mfa erlang:apply/2
pid <0.40.0> mfa erlang:apply/2
pid <0.37.0> reason normal
pid <0.40.0> reason {ok,[{call,1,{atom,1,spawn},[{fun,1,{clauses,[{clause,1,□,[
],[{call,1,{remote,1,{atom,1,timer},{atom,1,sleep}},[{integer,1,2000}]}]}]}}]}]}
sender <0.40.0> -> pid <0.26.0> reason {o
pid <0.41.0> mfa erlang:apply/2
pid <0.42.0> mfa erlang:apply/2
pid <0.39.0> reason normal
pid <0.41.0> reason normal
```

Common aggregation functions

- avg
- count
- lquantize (linear)
- quantize (log - power of 2)
- sum
- min
- max

Enabling DTrace in Erlang

Build your own with **kerl**

```
$ KERL_CONFIGURE_OPTIONS="--with-dynamic-trace=dtrace" \
  kerl build R16B03 \
    r16b03-dtrace
```

Build your own with **kerl**

```
$ kerl install \
    r16b03-dtrace ~

$ ~`whoami`/activate
```

Build your own with **erlbrew***

```
$ ERLBREW_CONFIGURE_OPTIONS="--with-dynamic-trace=dtrace" \
    erlbrew install R16B03
```

* I wrote erlbrew because reasons

erltrace:
DTrace from inside Erlang

https://github.com/project-fifo/erltrace

Write DTrace scripts and process DTrace output from inside Erlang.

Why?

- Push into folsom or <insert your metrics collection tools here>
- Send an alert to a human that "the thing is happening again"

Might be too expensive depending on environment/load.

Interpreting DTrace Output

erl shell input

```
mallen-rMBP:~ mallen$ erl
Erlang R16B01 (erts-5.10.2) [source] [smp:8:8] [async-threads:10] [hipe] [kernel
-poll:false] [dtrace]

Eshell V5.10.2  (abort with ^G)
1> dyntrace:available().
true
2> F = fun() -> timer:sleep(2000) end.
#Fun<erl_eval.20.80484245>
3> spawn(fun() -> timer:sleep(2000) end).
<0.37.0>
4> spawn(fun() -> timer:sleep(2000) end).
<0.39.0>
5> spawn(fun() -> timer:sleep(2000) end).
<0.41.0>
```

Intrepreting DTrace Output

spawn-exit.d output

```
mallen-rMBP:~ mallen$ pid <0.35.0> reason {ok,[{match,1,{var,1,'F'},{fun,1,{clau
ses,[{clause,1,□,□,[{call,1,{remote,1,{atom,1,timer},{atom,1,sleep}},[{integer
,1,2000}]}]}]}}}}]}
sender <0.35.0> -> pid <0.26.0> reason {o
pid <0.36.0> mfa erlang:apply/2
pid <0.36.0> reason {ok,[{call,1,{atom,1,spawn},[{fun,1,{clauses,[{clause,1,□,[
],[{call,1,{remote,1,{atom,1,timer},{atom,1,sleep}},[{integer,1,2000}]}]}]}}]}]}
sender <0.36.0> -> pid <0.26.0> reason {o
pid <0.37.0> mfa erlang:apply/2
pid <0.38.0> mfa erlang:apply/2
pid <0.38.0> reason {ok,[{call,1,{atom,1,spawn},[{fun,1,{clauses,[{clause,1,□,[
],[{call,1,{remote,1,{atom,1,timer},{atom,1,sleep}},[{integer,1,2000}]}]}]}}]}]}
sender <0.38.0> -> pid <0.26.0> reason {o
pid <0.39.0> mfa erlang:apply/2
pid <0.40.0> mfa erlang:apply/2
pid <0.37.0> reason normal
pid <0.40.0> reason {ok,[{call,1,{atom,1,spawn},[{fun,1,{clauses,[{clause,1,□,[
],[{call,1,{remote,1,{atom,1,timer},{atom,1,sleep}},[{integer,1,2000}]}]}]}}]}]}
sender <0.40.0> -> pid <0.26.0> reason {o
pid <0.41.0> mfa erlang:apply/2
pid <0.42.0> mfa erlang:apply/2
pid <0.39.0> reason normal
pid <0.41.0> reason normal
```

Intrepreting DTrace Output

```
#!/usr/sbin/dtrace -qs

erlang*:::process-spawn
{
    printf("pid %s mfa %s\n", copyinstr(arg0), copyinstr(arg1));
}

erlang*:::process-exit
{
    printf("pid %s reason %s\n", copyinstr(arg0), copyinstr(arg1));
}

erlang*:::process-exit_signal
{
    printf("sender %s -> pid %s reason %s\n",
       copyinstr(arg0), copyinstr(arg1), copyinstr(arg2));
}
```

process-spawn

(PID, MFA)

process-exit

(PID, REASON)

process-exit_signal

(SEND_PID, RECV_PID, REASON)

Intrepreting DTrace Output

spawn-exit.d output

```
mallen-rMBP:~ mallen$ pid <0.35.0> reason {ok,[{match,1,{var,1,'F'},{fun,1,{clau
ses,[{clause,1,□,□,[{call,1,{remote,1,{atom,1,timer},{atom,1,sleep}},[{integer
,1,2000}]}]}]}}}}]}
sender <0.35.0> -> pid <0.26.0> reason {o
pid <0.36.0> mfa erlang:apply/2
pid <0.36.0> reason {ok,[{call,1,{atom,1,spawn},[{fun,1,{clauses,[{clause,1,□,[
],[{call,1,{remote,1,{atom,1,timer},{atom,1,sleep}},[{integer,1,2000}]}]}]}}]}]}
sender <0.36.0> -> pid <0.26.0> reason {o
pid <0.37.0> mfa erlang:apply/2
pid <0.38.0> mfa erlang:apply/2
pid <0.38.0> reason {ok,[{call,1,{atom,1,spawn},[{fun,1,{clauses,[{clause,1,□,[
],[{call,1,{remote,1,{atom,1,timer},{atom,1,sleep}},[{integer,1,2000}]}]}]}}]}]}
sender <0.38.0> -> pid <0.26.0> reason {o
pid <0.39.0> mfa erlang:apply/2
pid <0.40.0> mfa erlang:apply/2
pid <0.37.0> reason normal
pid <0.40.0> reason {ok,[{call,1,{atom,1,spawn},[{fun,1,{clauses,[{clause,1,□,[
],[{call,1,{remote,1,{atom,1,timer},{atom,1,sleep}},[{integer,1,2000}]}]}]}}]}]}
sender <0.40.0> -> pid <0.26.0> reason {o
pid <0.41.0> mfa erlang:apply/2
pid <0.42.0> mfa erlang:apply/2
pid <0.39.0> reason normal
pid <0.41.0> reason normal
```

(More) Perilous Live Demos

Erlang DTrace resources:

- `$ERL_INSTALL/runtime_tools-*/examples`
- `http://www.erlang.org/doc/apps/runtime_tools/DTRACE.html`
- `http://vimeo.com/33999876`

DTrace resources:

- **http://dtracehol.com/#Intro**
- **http://dtrace.org/guide/preface.html**
- **http://dtracebook.com/index.php/Languages**
- **http://www.amazon.com/dp/0132091518**
- **https://github.com/erlang/otp/tree/master/lib/runtime_tools/examples**

Go see Rick Reed's talk at 14:00 to get a taste for how DTrace helps you squeeze every last juicy drop of performance from hardware

Thank you!

Appendix:
Erlang DTrace Probes

message-send (local)

(SEND_PID, RECV_PID, SIZE, LABEL, PREV_TOKEN_CNT, CURRENT_TOKEN_CNT)

PIDs are **strings** to DTrace, not ints

message-send_remote

(SEND_PID, RECV_PID, SIZE, LABEL, PREV_TOKEN_CNT, CURRENT_TOKEN_CNT)

PIDs are **strings** to DTrace, not ints

message-send_remote

```
(SEND_PID, NODE_NAME,
RECV_PID, SIZE, LABEL,
PREV_TOKEN_CNT,
CURRENT_TOKEN_CNT)
```

PIDs are **strings** to DTrace, not ints

message-queued

message-receive

(RECV_PID, SIZE, Q_LEN, TOKEN, PREV_TOKEN_CNT, CURRENT_TOKEN_CNT)

PIDs are **strings** to DTrace, not ints

copy-struct
(SIZE)

copy-object

(RECV_PID, SIZE)

local-function_entry

global-function_entry

function-return

(PID, MFA, DEPTH)

bif-entry

bif-return

nif-entry

nif-return

(PID, MFA)

gc_major-start

gc_minor-start

(PID, NEEDED_HEAP_WORDS)

gc_major-end

gc_minor-end

(PID,RECLAIMED_SPACE)

process-spawn

(PID, MFA)

process-exit

(PID, REASON)

process-exit_signal

(SEND_PID, RECV_PID, REASON)

process-exit_signal_remote

```
(SEND_PID, NODE_NAME,
RECV_PID, REASON, TOKEN,
PREV_TOKEN_CNT,
CURR_TOKEN_CNT)
```

process-scheduled

process-hibernate

(PID, MFA)

process-unscheduled

(PID)

process-port_unblocked

(PID, PORT)

process-heap_grow

process-heap_shrink

(PID, OLD_SIZE, NEW_SIZE)

dist-monitor

```
(NODE_NAME, EVENT_TYPE,
MONITORED_NODE_NAME,
NODE_TYPE, REASON)
```

dist-port_busy

(NODE_NAME, PORT, REMOTE_NODE_NAME, BLOCKED_PID)

dist-output

dist-outputv

(NODE_NAME, PORT, REMOTE_NODE, BYTE_CNT)

dist-port_not_busy

(NODE_NAME, PORT, REMOTE_NODE)

port-open

(PID, PORT_NAME, PORT)

port-command

(PID, PORT, PORT_NAME, CMD_TYPE)

port-control

(PID, PORT, PORT_NAME, CMD_NUM)

port-exit

(PID, PORT, PORT_NAME, REASON)

port-connect

(PID, PORT, PORT_NAME, NEW_PID)

port-busy

port-not_busy

(PORT)

driver-init

(NAME, MAJOR, MINOR, FLAGS)

driver-finish

(NAME)

driver-start

driver-stop

driver-flush

(PID, NAME, PORT)

driver-stop

(PID, NAME, PORT)

driver-output

driver-outputv

(PID, PORT, PORT_NAME, BYTES)

driver-control

driver-call

(PID, PORT, PORT_NAME, CMD, BYTES)

driver-event

driver-ready_input

driver-ready_output

driver-timeout

driver-ready_async

driver-process_exit

driver-call

(PID, PORT, PORT_NAME)

driver-stop_select

(DRVR_NAME)

aio-pool_add

aio-pool_get

(PID, Q_LEN)

efile_drv-entry

```
(THREAD_ID, DRV_TAG, USER_CMD,
CMD_NUM, STR_ARG0, STR_ARG1,
INT_ARG0, INT_ARG1, INT_ARG2,
INT_ARG3, PORT_ID)
```

efile_dvr-int_entry

efile_dvr-int_return

(THREAD_ID, DRV_TAG, CMD_NUM)

efile_dvr-return

(THREAD_ID, DRV_TAG, USER_TAG, CMD_NUM, SUCCESS_BOOL, ERRNO)

Make your own DTrace probes *in Erlang* by using

dyntrace:p()

Comes with the standard install.

user_trace-n0 .. user_trace-n950

```
(PID, USER_TAG, INT_ARG0,
INT_ARG1, INT_ARG2, INT_ARG3,
STR_ARG0, STR_ARG1, STR_ARG2,
STR_ARG3)
```

DTrace and Erlang

```
Erlang R16B01 (erts-5.10.2) [source] [smp:8:8] [async-threads:10] [hipe] [kernel
-poll:false] [dtrace]

Eshell V5.10.2  (abort with ^G)
1> dyntrace:available().
true
2> dyntrace:put_tag("GOOOOOOAALLLL!!!1").
undefined
3> dyntrace:p(9, 7, 8, "one", "hi mom").
true
```

```
mallen-rMBP:~ mallen$ sudo dtrace -q -s ./lib/runtime_tools-1.8.11/examples/user
-probe.d
Password:
<0.32.0> GOOOOOOAALLLL!!!1 9 7 8 0 'one' 'hi mom' ''
```