

Trexplorer, a trace exploration tool for Erlang

Judit Kőszegi

University of Kent
Eötvös Loránd University

February 17, 2010

Judit Kőszegi

koszegijudit@gmail.com

kojqaaai@inf.elte.hu

Supervisor:

Olaf Chitil

O.Chitil@kent.ac.uk

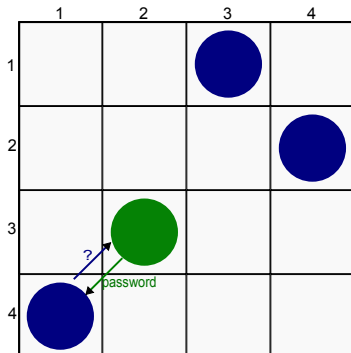
Overview

- 1 Introduction
- 2 Erlang - distribution and concurrency
- 3 Tracing
- 4 Trexplorer, the first steps
- 5 Debugging with Trexplorer
- 6 Summary

A sample problem

The secret password

- A given sized room
- Given number of persons (random walking, communication)
- A counter

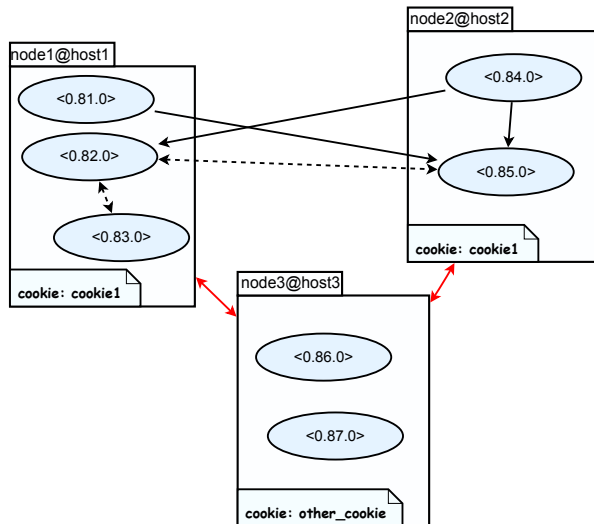


- I made a mistake \implies the program can't terminate
- Could have a lot of reasons, e.g.:
 - The counter process didn't get messages from the persons \implies can't stop them
 - Communication fault between the persons
 - Fault with the random walking

- A tool, which is suitable for
 - Exploring what happened with the processes during the execution
 - Stepping independently of the time arrow
 - Concentrating on causal relationships
 - Finding the reason of the faulty behaviour

Erlang - distribution and concurrency

Erlang is a general-purpose functional programming language and runtime environment.



Definition of tracing

- Tracing, logging = collecting informative messages about the execution of a program at run time
- *Logging*
 - Have to insert log statements
 - The result is a readable text file
- *Tracing*
 - Don't have to modify the source code
 - The result is often a binary file
 - Mainly for debugging purpose

Tracing Erlang programs

Erlang provides built-in functions (BIFs) for low-level tracing.

- Usage: without trace-compiling the code
- *Tracer process*: receives trace events from other processes
- Settings:
 - Which processes you want to trace
 - Which trace events you want to generate
- Trace events:
 - Global and local function calls
 - Garbage collection and memory usage
 - **Process-related activities and message passing**

Existing tracing tools in Erlang I.

- **dbg Tracer**

- Based on trace BIFs
- Provides a user-friendly interface
- It has small impact on system performance \implies good for large live systems
- Its output can be printed out or diverted

- **Inviso**

- Also based on trace BIFs
- Has many more possibilities
- Designed for distributed applications. One can:
 - Generate a trace log for every nodes separately
 - Collect and merge the logs
 - Divert the output to a *worker function*
- *A lot of steps to initialize, start and finish tracing*
- *Too many options*
 \implies *hard to choose the suitable ones*

- **Onviso**

- An extension of Inviso with a wrapper
- Provides almost the same functionalities
- Has an easy to use API
- Setting up the trace across multiple nodes and merging trace log files: only with two functions.

I chose Onviso to generate the process-related trace events for my tool.

Trexplorer, the first steps

- 1 **Initialize** and **start** tracing
 - Nodes - in a list
 - Processes: pid() / all / new / existing
- 2 **Execute** the program
- 3 **Collect** and **merge** trace informations from the generated trace log files and **store** them into data tables

Trace events to table records

Spawn-exit table

- *record*:

```
{spawn_exit, Ts, Node, Pid, Pid2, Mfa, Exit_ts, Exit_reason}
```

- *trace events*:

```
{trace_ts, Pid, spawn, Pid2, MFA, Ts}
```

```
{trace_ts, Pid2, exit, Exit_reason, Exit_ts}
```

Message passing table

- *record*:

```
{message_passing, Ts, Node, Pid, Message, Node2, Pid2, Received_ts}
```

- *trace events*:

```
{trace_ts, Pid, send, Message, Pid2, Ts}
```

```
{trace_ts, Pid, send_to_non_existing_process, M, Pid2, Ts}
```

```
{trace_ts, Pid2, 'receive', Message, Received_ts}
```

Linking table

Registering table

Debugging

Debugging: Methodical process of finding and fixing bugs in a program

- Log-based debugging
 - Logging statements in the source code
 - It exposes the actual history of execution
 - *Manual analyses of huge traces is hard*
- Breakpoint-based debugging
 - Pauses the execution at a determined point, inspects memory contents
 - Continues execution step-by-step
 - *Previous information is not reachable*
- Omniscient (back-in time) debugging
 - Records the events that occur during the execution
 - The user can navigate through the trace
 - *Combines the advantages of the two previous debugging methods*

How to navigate through the trace?

- Define various queries and combine them
- Menu-based and command-based interface
- Facilities:
 - List normal or abnormal exited processes
 - List message passings between given processes
 - Choose a process and list all related messages, linkings and other informations with special filters

Back to our first example...

- Choose the counter process... Are there enough received messages?
 - Yes \implies *messages are in improper format*
 - No...

Trexplorer debugging example

Back to our first example...

```
<test@PCSBJSJK311>16> command:start<>.
Trexplorer command-based client
=====
<?> proc -not
=== Not-exited processes ===
      Process
'counter / <0.90.0>'
'room / <0.91.0>'
          <0.92.0>
          <0.93.0>
          <0.94.0>
          <0.95.0>
-----
<?> choose counter
<?> <0.90.0> received
=== Message passings ===
Message: got_the_pass
      Process                               Node                               Time
Sent:      <0.93.0>                         test@PCSBJSJK311                 <<2010,2,17>,<0,1,28>>
Received:  'counter / <0.90.0>'             test@PCSBJSJK311                 <<2010,2,17>,<0,1,28>>
-----
Message: got_the_pass
      Process                               Node                               Time
Sent:      <0.92.0>                         test@PCSBJSJK311                 <<2010,2,17>,<0,1,28>>
Received:  'counter / <0.90.0>'             test@PCSBJSJK311                 <<2010,2,17>,<0,1,28>>
-----
Message: got_the_pass
      Process                               Node                               Time
Sent:      <0.94.0>                         test@PCSBJSJK311                 <<2010,2,17>,<0,1,28>>
Received:  'counter / <0.90.0>'             test@PCSBJSJK311                 <<2010,2,17>,<0,1,28>>
-----
<?> <0.90.0> _
```

Trexplorer debugging example

Back to our first example...

```
(test@PCSBJSJK311)10> command:start<>.

Trexplorer command-based client
=====

<?> proc -not
=== Not-exited processes ===

      Process
'counter / <0.66.0>'
'room / <0.67.0>'
          <0.68.0>
          <0.69.0>
          <0.70.0>
          <0.71.0>
-----

<?> choose counter
<?> <0.66.0> received
=== Message passings ===

Message: got_the_password
      Process                                     Node                                     Time
Sent:      <0.69.0>                               test@PCSBJSJK311 <<2010.2.16>,<23.53.50>>
Received:  'counter / <0.66.0>'                   test@PCSBJSJK311 <<2010.2.16>,<23.53.50>>
-----

Message: got_the_password
      Process                                     Node                                     Time
Sent:      <0.70.0>                               test@PCSBJSJK311 <<2010.2.16>,<23.54.6>>
Received:  'counter / <0.66.0>'                   test@PCSBJSJK311 <<2010.2.16>,<23.54.6>>
-----

<?> <0.66.0> choose <0.68.0>
<?> <0.68.0> =
```

Trexplorer debugging example

Back to our first example...

- Choose the counter process... Are there enough received messages?
 - Yes \implies *messages are in improper format*
 - No...
- List the received messages of a person... Is there a message with the password?
 - Yes \implies *the process forgot to send the message to the counter or sent it to an other process*
 - No \implies the other persons couldn't send the password... Why?
- List the sent messages of another person...

Trexplorer is useful, when...

- our Erlang program works with numerous processes and communication

because...

- we don't have to analyse traces by hand
- we can easily find the answer for our questions.

But can't use when ...

- we got purely functional code
- the execution is too large, the result text of a query can be more than one page

Future plans, opportunities

- Test the tool in larger, industrial environment, then correct and improve the tool
- Grouping the tightly coupled processes
- Visualisation using graphs
- Make an extension which deals with function calls in a similar principle

Thank you!

Questions?