Automatic Assessment of Failure Recovery in Erlang Applications

Jan Henry Nyström
henry.nystrom@erlang-consulting.com

UPPSALA UNIVERSITET

Overview

- Why is fault tolerance important
- How can we achieve fault tolerance
- How do we do fault tolerance in Erlang
- So what is the problem with that then?
- A solution to the problems
- Extracting the information
- What do we do with the information
- Correctness concerns
- The tool

Why is fault tolerance important

- Obvious where a failure can have life threatening effects
  - Nuclear power plant control systems
  - Aircraft control systems
  - Vending machine control systems

- But the dependence on 24/7 available systems is pervasive
  - Banking
  - Travel agent
  - Health services
  - Bookies
  - Games

How can we achieve fault tolerance

Problems:

- Hardware fails
- Nonconformance to specification
- Changed interfaces
- Idiots (users)
- There are three things in life we cannot avoid
  - Death
  - Taxes
  - Bugs (code, specification and requirements)

How can we achieve fault tolerance

Solution 1:

- Ensure there are no errors in the
  - Code
  - Specification
  - Requirements
- Check everything the environment sends
- Handle all the possible external failures
- Handle all combinations of external failures

- If you believe in that, let me tell you that Father Christmas was my grandfather in a silly costume.

How can we achieve fault tolerance

Solution 2:

- Handle the unexpected
- Best effort diagnosis that things are bad
- Restart the subsystem that is naughty

How do we do fault tolerance in Erlang

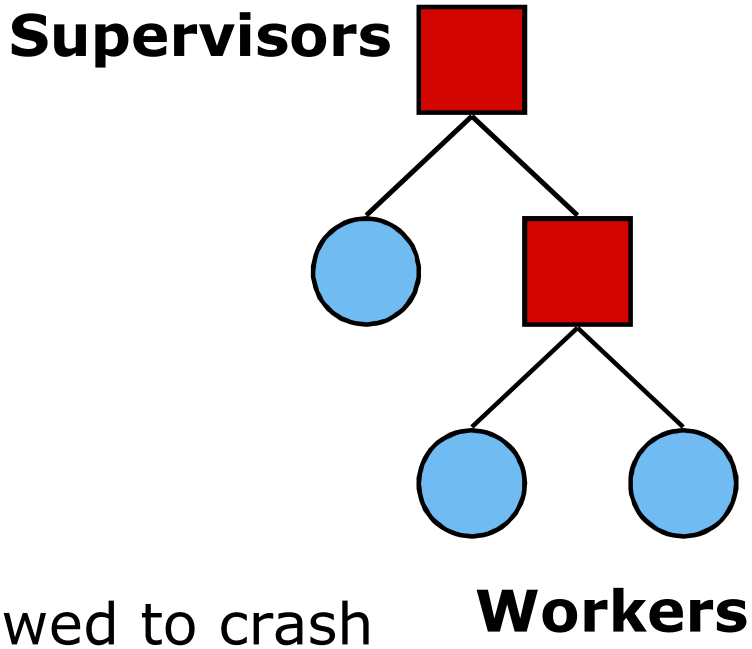
Solution 2:

- Handle the unexpected
  - Let it crash!
  - Exceptions are only caught when you can solve the problem here and now
- Best effort diagnosis that things are bad
  - Assertional style of programming, stating the correct case rather than hedging your bets. See previous point
  - When some process has crashed it is trivial really
- Restart the subsystem that is naughty
  - Supervision of processes by other processes that restarts them, then iterate

How do we do fault tolerance in Erlang (OTP)

Solution 2:

- Library support for hierarchical supervision structures
- Configurable how each child is treated
  - Permanent
  - Transient
  - Temporary
- Handles depencies between children
  - one_for_all
  - rest_for_one
  - one_for_one
- Configurable how often children are allowed to crash
  - Maximum number during Maximum time



How do we do fault tolerance in Erlang (Real Life)

Solution 2:

"Since cut-over of the first nodes in BT's network in January 2002, only one minor fault has occurred, resulting in 99.9999999% availability."

"As a matter of fact, the network performance has been so reliable that there is almost a risk that our field engineers do not learn maintenance skills"

Bert Nilsson, Director
NGS-Programs Ericsson

Ericsson Contact, Issue 19 2002

So what is the problem with that then?

- We have a nice framework to build fault tolerant systems in
- It has been proved in practice that it can be used to build large scale fault tolerant systems
- But, how do we now that we have actually built a system with good fault tolerance properties?

- Test…

  good but we have to handle all combinations of failures

- Understanding your supervision structure really, really helps

  That is actually what is talk is all about

A solution to the problem

- So we want to understand our supervision structure, how do we find it?
- The design documentation

Do you recognise the gentlemen on the right?

- The next port of call is the implementation
  - But it is spread out everywhere
  - The specification for the supervisors is generated by code
  - It can be modified by configurations

- We have computers,
  let them do all the legwork

Extracting the information

- So we want to have a tool that extracts the supervision structure from the source code of an Erlang system
- Unfortunately, it is not only impossible in the general case, it is intractable in most of the actual cases.
- Let's run it and inspect it
  - We need all the possible interleavings of starting of processes and messages between them
  - Potentially only one structure out of many depending on configuration
  - We have to have a runnable system before we can even start inspecting it

Extracting the information (Take 2)

- Limit analysis to OTP systems
  - Well defined supervisors provided (you got the code)
  - Synchronised start up sequence (behaviours required)
  - It is sufficient to look at processes in isolation
  - Well defined initialisation part of processes that are behaviours
- Symbolic evaluation
  - Enables us to handle the case that we lack information
- Restrict the analysis to the "static part of the system"
  - Otherwise when do we stop
- The unit of interest is the application
  - Natural unit in OTP

Extracting the information (In practice)

Missing information

- Code does not exist or compile, missing configuration
- We introduce an “unknown” value
  - top element of the semantic domain
- But we have to be able to handle that value
- We introduce non determinacy in the evaluation
- When we have a choice depending on an unknown value we explore all the possible evaluations

Extracting the information (In practice)

Evaluation Depth

- Potentially the code does not terminate when we have an unknown value
- When should we give up, configurable
- Simple cycle detection

Extracting the information (In practice)

Setup

- Paths
- Inclusion directories
- Node name

Extracting the information (In practice)

Global state

- We cannot let the analysed system interact directly with the real world
- We have to be able to reset the global state for an alternative evaluation
- We simulate the global state of the system for the parts we cannot do without and assume no knowledge of the rest
- Simulated are (and possibly initialised):
  - Registry, Erlang Term Storage(ets), Mnesia, File System
- The generation of global states has to have a good chaching mechanism

Extracting the information (In practice)

Exception Handling

- The non determinacy and exceptions are not good bedfellows
- Many constructs make it necessary to consider a possible exception when we evaluate with an unknown value present
- This can cause an intractable growth in the number of evaluations that has to be considered
- Three strategies to deal with this:
  - Hope for the best
  - Ignore uncaught exceptions, they would just crash the system anyway
  - Ignore all exceptions, this is unsound

Extracting the information (In practice)

Behaviours

- The behaviours' behaviour is supposed to be well defined
- We do not want to analyse that
- The behaviours are embedded in the evaluator for efficiency

What do we do with the information

- Read and inspect it manually
  But we have computers

- We can animate the supervision structure and actually show what happens when a selected process fails
- We can combine information and deduct properties
  - How often a particular process has to terminate to cause the entire structure to fail
- We can formally reason about properties
- What properties should we consider?
  - Repair
  - No concealment
  - Good design principles

What do we do with the information

The **Repair** property:

Whenever a process that takes part in the supervision structure fails, the supervision structure returns to the process structure prior to the failure after a reasonable delay.

What do we do with the information

The **Repair** property:

- We need to check:
  - Each processes that fails is replaced by an equivalent "restarted" process in the structure
  - Each process replaced by a restarted process has indeed terminated
- Side orders
  - A non-supervisor that is linked to a failed process does not trap exits
  - The initialisation of a restarted process creates the same structure as the process it replaces

What do we do with the information

The **Non concealment** property:

When the cause of a failure is not transient or sufficiently infrequent to let the application function acceptably, only a small number of recoveries should occur before the supervision structure fails.

.

What do we do with the information

Good design principles

- All processes should create their children using spawn_link
- The max restart frequency of intermediate supervisors is 0
- Only supervisors should have shutdown time infinity

- But most importantly one can check design rules of the company/devivsion/team

Correctness concerns

Why should we trust the extracted structure?

- A four stage rocket
  - Formalise a semantics for the needed subset of Erlang
  - Formalise a semantics for Erlang that has an unknown value and where an evaluation may have several results
  - Prove that for the supervision structure that is generated by the "real" semantics is included in the set of structures generated by the "abstract" semantics
  - Base the symbolic evaluation on the "abstract" semantics

- In fact it was done the other way around,
  almost

The tool

Yes there is a tool

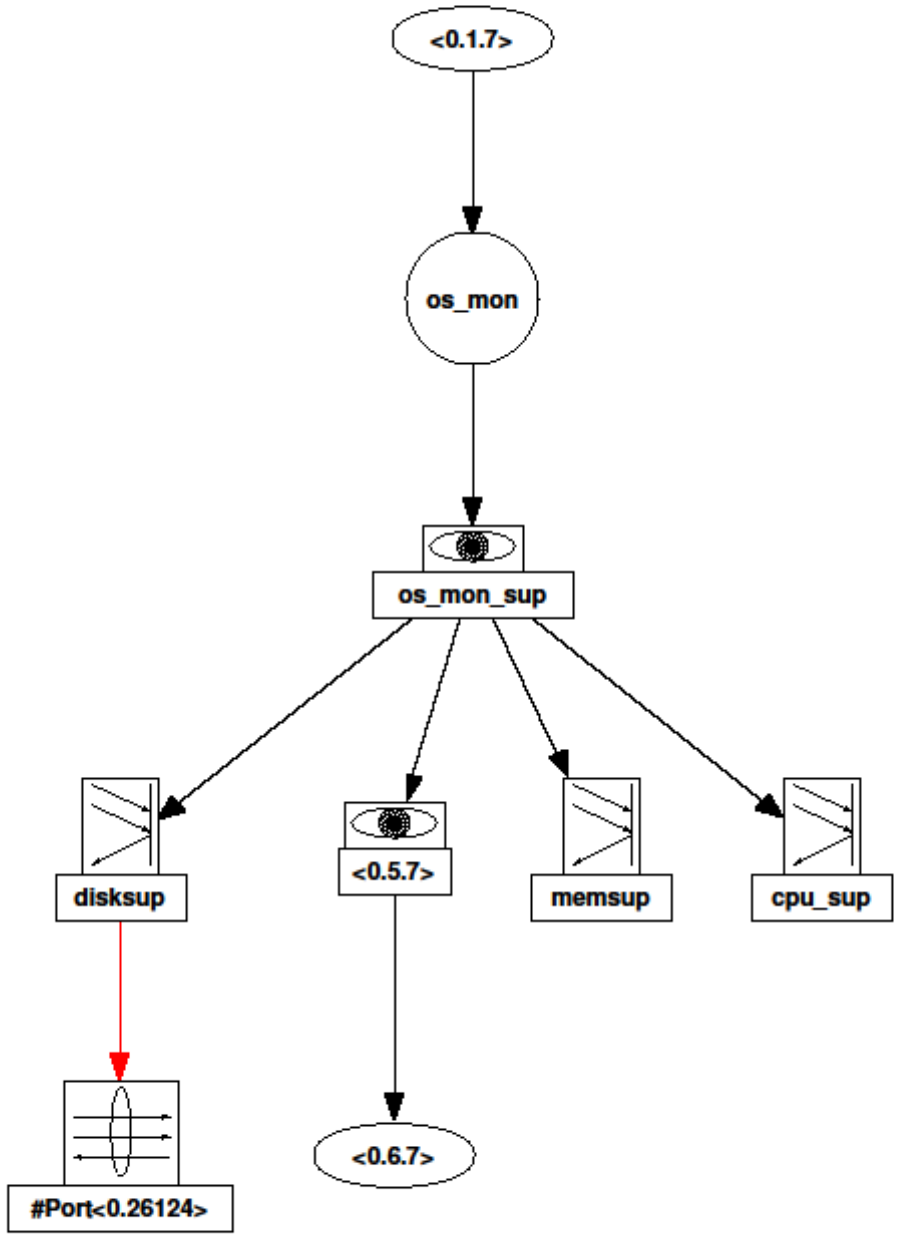
- It is pre-alpha
- Can analyse a number of OTP applications
- Has been used to analyse a number of applications in AXD301
  - The analysis results were confirmed by the designers
- Needs several major rewrites before going beta
  - Packages are still experimental ☹
  - Extended support for the global state
  - More OTP libraries needs to be embedded in the evaluator
  - Configuration simplified
  - User interface (sigh)

The tool

Yes there is a tool

- The extracted supervision structure can either be pretty printed or displayed graphically
- The graphical display can be interacted with to:
  - Choose the information displayed
  - Effects of termination coloured in for a specific process
- The graphical display is ugly and intended changes are :
  - New web interface
  - The ability to animate restart sequences
  - The possibility to draw the supervision structure
  - Show the difference between structures
  - Integrated with a evaluator (analyser) GUI

The tool


<0.1.7>
os_mon
os_mon_sup
disksup
<0.5.7>
memsup
cpu_sup
#Port<0.26124>
<0.6.7>

Further reading

My Ph.D. Thesis:

Official university site

- http://uu.diva-portal.org/smash/record.jsf?pid=diva2:213697

All my papers

- http://sites.google.com/site/janhenrynystrom/Home

The ETC pages (TBD)

- http://www.erlang-consulting.com

Workshop paper

- http://www.erlang.org/workshop/2009/