Safe Sessions for Erlang

Adrián Palacios
(joint work with Ivan Lanese, Naoki Nishida and Germán Vidal)

Technical University of Valencia

PLDI 2017 Student Research Competition
June 21, 2017
Barcelona, Spain
Breaking News!

**Erlang** systems are fault-tolerant!

...thanks to the *Let It Crash* philosophy
Breaking News!

Erlang systems are fault-tolerant!

...thanks to the *Let It Crash* philosophy
The Let It Crash philosophy

“Do not handle errors in your programs. If a process is about to crash, let it crash and restart it immediately”

Made possible by a process supervision tree where:

- **Workers**: Do all the hard work
- **Supervisors**: Restart workers if they crash
The Let It Crash philosophy

“Do not handle errors in your programs. If a process is about to crash, let it crash and restart it immediately”

Made possible by a process supervision tree where:

- **Workers**: Do all the hard work
- **Supervisors**: Restart workers if they crash
Supervision tree

- **Workers**: Do all the hard work
- **Supervisors**: Restart workers if they crash
Our proposal

Supervisors do not make any verification when restarting workers
→ inconsistent system state

Safe Sessions

We propose safe sessions, an automatic recovery strategy for Erlang, as a complement to the Let It Crash philosophy.

In safe sessions, concurrent actions are registered, so that the system can return to a safe state in case of error.

Based on the reversible semantics for Erlang from [LOPSTR’16].
Supervisors do not make any verification when restarting workers
→ inconsistent system state

Safe Sessions
We propose safe sessions, an automatic recovery strategy for Erlang, as a complement to the Let It Crash philosophy

In safe sessions, concurrent actions are registered, so that the system can return to a safe state in case of error.

Based on the reversible semantics for Erlang from [LOPSTR’16].
Our proposal

Supervisors do not make any verification when restarting workers
→ inconsistent system state

Safe Sessions

We propose **safe sessions**, an automatic recovery strategy for Erlang, as a complement to the *Let It Crash* philosophy.

In safe sessions, **concurrent actions are registered**, so that the system can return to a safe state in case of error.

Based on the reversible semantics for **Erlang** from [LOPSTR’16].
The **Erlang** language
Erlang’s features

The **Erlang** language has:

- functional and concurrent features
- concurrency based on the actor model

These features make it appropriate for distributed applications

Ericsson  WhatsApp  Messenger (Facebook chat)  Ejabberd
Concurrent actions: Spawn

**Spawn:** Create a new process
Concurrent actions: Send

Send: Send a message to another process
Concurrent actions: Receive

**Receive:** Suspend execution until a message from the mailbox matches any of the receive clauses
Example

Many things can **go wrong**!
Example

Many things can **go wrong**!
Safe Sessions
Safe Sessions

We add a new construct to **Erlang**

```
  safety expr end
```

If `expr` goes wrong, we **restore** the process
Safe Sessions

If \( expr \) goes wrong, we \textbf{restore} the process

Causal Consistency

An action may be undone only if every action caused by that action has not been executed yet or has been undone.

\textbf{Restoring} the state is not enough to ensure causal consistency, we must also undo the effects of its

- spawn actions
- send actions

We solve this by “propagating the safety” to dependent processes.
If \( \text{expr} \) goes wrong, we \texttt{restore} the process.

Causal Consistency

An action may be undone only if every action caused by that action has not been executed yet or has been undone.

Restoring the state is not enough to ensure \texttt{causal consistency}, we must also undo the effects of its

- spawn actions
- send actions

We solve this by “propagating the safety” to dependent processes.
Safe session (algorithm)

When \( p \) enters the safety block...

1. We take a snapshot of \( p \) before the evaluation of \( expr \).
2. If another process \( q \) is sent a message from \( p \), or is spawned by \( p \), we take a snapshot of \( q \) (safety propagation).
3. If the evaluation of \( expr \) fails:
   - We restore the state from \( p \).
   - We restore the state of processes sent a message or spawned by \( p \) (safety propagation).
   - Go to step 3.
4. We discard all the snapshots.
Implementation

Safe sessions can be implemented using:

- **Monitors:**
  - Intercept incoming and outgoing messages
  - Send signals between themselves to propagate the safety
- **Instrumentation:**
  - Enable interaction of processes with their monitors


Program Instrumentation

\[
\begin{align*}
\text{[safety expr end]}_M & \rightarrow M \langle \text{start_session} \rangle, \\
\text{[expr]}_M, \\
M & \langle \text{end_session} \rangle \\
\text{[spawn(\ldots)]}_M & \rightarrow M \langle \text{spawn(\ldots)} \rangle, \\
& \text{receive } \langle \text{spawn_with}, P \rangle \rightarrow P \text{ end} \\
\text{[self()]}_M & \rightarrow M \\
\text{[Pid ! expr]}_M & \rightarrow M \langle \text{send}(Pid), [expr]_M \rangle, \\
& \text{receive } \langle \text{sent_as}, E \rangle \rightarrow E \text{ end} \\
\text{[receive clauses end]}_M & \rightarrow M \langle \text{receive}, clauses \rangle, \\
& \text{Arg} = \text{receive} \\
& \langle \text{rec_msg}, Msg \rangle \rightarrow Msg \text{ end}, \\
& \text{case Arg of } [\text{clauses}]_M \text{ end}
\end{align*}
\]

Concurrent actions are replaced by queries to the monitor.
Program Instrumentation

\[
\begin{align*}
\text{[safety expr end]}_M & \rightarrow M \! \langle \text{start_session} \rangle, \\
\text{[expr]}_M, \\
M & \! \langle \text{end_session} \rangle \\
\text{[spawn(\ldots)]}_M & \rightarrow M \! \langle \text{spawn(\ldots)} \rangle, \\
& \text{receive } \langle \text{spawn_with}, P \rangle \rightarrow P \text{ end} \\
\text{[self()]}_M & \rightarrow M \\
\text{[Pid ! expr]}_M & \rightarrow M \! \langle \text{send(Pid)}, [expr]_M \rangle, \\
& \text{receive } \langle \text{sent_as}, E \rangle \rightarrow E \text{ end} \\
\text{[receive clauses end]}_M & \rightarrow M \! \langle \text{receive, clauses} \rangle, \\
Arg & = \text{receive} \\
& \langle \text{rec_msg}, Msg \rangle \rightarrow Msg \text{ end,} \\
& \text{case Arg of } [\text{clauses}]_M \text{ end}
\end{align*}
\]

Concurrent actions are replaced by queries to the monitor.
Example with safe sessions
Safe Sessions

client

proxy

server
Safe Sessions

client

proxy

server

request

result

result
Safe Sessions

client

proxy

server

request

result

result
Safe Sessions

client

proxy

server
Safe Sessions

client  request  proxy  server
Safe Sessions

A Palacios (Valencia, Spain)

Safe Sessions for Erlang

PLDI 2017 SRC

38 / 42
Conclusions
Conclusions

Some related work:

- **Field and Varela [POPL’05]:**
  - checkpoint-based approach with some similarities
  - they aim at defining a new language, rather than extending one

- **Neykova and Yoshida [CC’17]:**
  - interprocedural recovery strategy based on session types
  - not so fine-grained, we could define an intraprocedural strategy

In the future, we will:

- refine our design of safe sessions
- develop an implementation
- compare our implementation against other approaches
Thanks for your attention!