Let it Recover: Multiparty Protocol-Induced Recovery

Rumyana Neykova, Nobuko Yoshida
Imperial College London
“Fail fast and recover quickly”

Erlang proverb

“Fail fast and recover quickly and safely”

OPCT proverb (after this talk)
Part One
Background
The Erlang programming language

factorial(0) -> 1;
factorial(X) when X > 0 -> X * factorial(X-1).
Erlang’s coding philosophy

A problem has been detected and windows has been shut down to prevent damage to your computer.

The problem seems to be caused by the following file: SPCMDCON.SYS

PAGE_FAULT_IN_NONPAGED_AREA

If this is the first time you've seen this stop error screen, restart your computer. If this screen appears again, follow these steps:

_LET_IT_CRASH_

If problems continue, disable or remove any newly installed hardware or software. Disable BIOS memory options such as caching or shadowing. If you need to use Safe Mode to remove or disable components, restart your computer, press F8 to select Advanced Startup Options, and then select Safe Mode.

Technical information:

*** STOP: 0x00000050 (0xFD3094C2,0x00000001,0xFBFE7617,0x00000000)

*** SPCMDCON.SYS – Address FBFE7617 base at FBFE5000, DateStamp 3d6dd67c
Let it crash: Erlang’s fault tolerance model

- Organise your processes in supervision trees

- Do not program defensively, let the process crash
- In case of error, the process is automatically terminated
- Processes are linked. When a process crashes linked process are notified and (can be) restarted.

Supervision Strategies
- one-for-one
- all-for-one
- rest-for-one

Recently adopted by Scala, Akka, & Google
Supervision strategies: Drawbacks

- Supervision strategies are: statically defined, error-prone

- A recovery may cause deadlocks, orphan messages, reception errors
How to generate *sound and efficient* supervision strategies?

By using Session Types!
Session Types Overview

- **Global protocol (session type)**
  \[ G = A \to B : \langle U_1 \rangle. B \to C : \langle U_2 \rangle. C \to A : \langle U_3 \rangle \]

- **Local protocol (session type)**
  - Slice of global protocol relevant to one role
  - Mechanically derived from a global protocol
  \[ T_A = !\langle B, U_1 \rangle. ?\langle C, U_3 \rangle \]

- **Process language**
  - Execution model of I/O actions by roles

- A system of *well-behaved processes* is free from deadlocks, orphan messages and reception errors

- The framework has been applied to Java, Python, MPI/C, Go…
Part Two

Let It Recover
A **recovered system** is free from deadlocks, orphan messages and reception error.

Outperforms one of the built-in recovery strategies in Erlang
This talk: Safe Recovery for Session Protocols

### Approach

- **Recovery algorithm** to analyse a global protocol as to calculate the dependencies of a failed process.

- Local supervisors *monitor* the state of the process in the protocol

- Protocol supervisors use the algorithms *at runtime* to decide which process to recover
\( \prec_{io} \) - input-output dependencies (assert the order between a reception of a message and a send action) should recover

\[ A \rightarrow B; \quad B \rightarrow C; \quad n_1 \prec_{io} n_2 \]

\( \preceq \) - precedence dependencies (represent the order between two nodes which have a common participant) should recover

\[ A \rightarrow B; \quad C \rightarrow B; \quad n_3 \preceq n_4; \quad n_3 \not\prec_{io} n_4 \]

Queue of B
Causalities

\(\prec_{io}\)-input-output dependencies (assert the order between a reception of a message and a send action) should recover

\[\begin{align*}
A & \rightarrow B; \\
& B \rightarrow C; \\
\hline
n_1 & \prec_{io} n_2
\end{align*}\]

\(\prec\)-precedence dependencies (represent the order between two nodes which have a common participant) should recover

\[\begin{align*}
A & \rightarrow B; \\
& C \rightarrow B; \\
\hline
n_3 & \prec n_4 \\
& n_3 \not\prec_{io} n_4
\end{align*}\]

\(\prec_{\uparrow}\)-guarded dependencies (represent dependencies of the failed node) should not recover

\[\begin{align*}
A & \rightarrow B; \\
& B \rightarrow C; \\
\hline
n_1 & \prec_{\uparrow} n_2
\end{align*}\]
Part Three

Recovery Algorithm
Recovery Algorithm

**Algorithm** Calculating affected nodes

**Input:** $n_i$ (a failed node), $p$ (a failed role)

**Output:** $\mathcal{N}$ (a set of affected nodes)

1. $\mathcal{N} = \mathcal{N}^{\rightarrow} = \{n | n_i \prec n \land n = r \rightarrow p\} \cup \{n_i\}$
2. $\mathcal{I} = \{n | ((n_i \prec n' \land n' = p \rightarrow r) \lor n' = n_i) \land n' \leftarrow_{I0} n\} \setminus \{n_i\}$
3. **repeat**
4. $\mathcal{N}^{\leftarrow} = \{n | n \leftarrow_{I0} n' \lor (n \prec n' \land n \in \mathcal{I}) \land n' \in \mathcal{N}^{\rightarrow}\}$
5. $\mathcal{N}^{\rightarrow} = \{n | n' \prec n \land n' \in \mathcal{N}^{\leftarrow}\} \cup (\mathcal{N} \cup \mathcal{I})$
6. $\mathcal{N} = \mathcal{N} \cup \mathcal{N}^{\leftarrow}$
7. $\mathcal{I} = \mathcal{I} \setminus \mathcal{N}^{\leftarrow}$
8. **until** $\mathcal{N}^{\leftarrow} = \mathcal{N}^{\rightarrow} = \emptyset$
9. **return** $\mathcal{N}$
Recovery Algorithm

- Step 1: Initialise the $\prec_t$ dependencies of the failed node
- Step 2: Backward traversal of $\prec_i$ dependencies
- Step 3: Forward Traversal of $\prec_o$ dependencies
- Step 4: Repeat 2-3 until no new dependencies are added
Step 1: Initialise the $\leq_\uparrow$ dependencies of the failed node

Step 2: Backward traversal of $\leq_{io}$ dependencies

Step 3: Forward Traversal of $\geq$ dependencies

Step 4: Repeat 2-3 until no new dependencies are added

\[ \begin{align*}
1: & B \rightarrow E; \\
3: & B \rightarrow A; \\
4: & C \rightarrow A; \\
5: & A \rightarrow D; \\
6: & D \rightarrow E; \\
7: & E \rightarrow B;
\end{align*} \]

Initialise

- $\leq_\uparrow$: 5, 6, 7

Final Condition

- $\leq_{io}$: 3, 4

- $\geq$: 3, 4

- not done

- done
Recovery points

- **recovery point**: take the top node from the set of recovery nodes

  - 1: B → C; 2: C → E;
  - 3: B → A; 4: C → A;

- **Global Recovery Table**

<table>
<thead>
<tr>
<th>Failure</th>
<th>Recovery points</th>
</tr>
</thead>
<tbody>
<tr>
<td>3, A</td>
<td>A: 3, B: 3, C: 4</td>
</tr>
<tr>
<td>3, B</td>
<td>C: 2, E: 2</td>
</tr>
<tr>
<td>4, C</td>
<td>C: 1, B: 1, ...</td>
</tr>
<tr>
<td>4, A</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Main Results: Transparency and Safety (informally)

**Theorem: Transparency**

*The recovered protocol is a reduction of the initial protocol.* The configuration of the system after a failure is reachable from the initial configuration.

**Theorem: Safety**

Any reachable configuration which is an initial configuration of well-formed global protocol is free from deadlock, an orphan massage and a reception error.
Part Four

Recovery Implementation
Enabling Protocol Recovery in

- **protocol supervisor** (recover processes)
- **local supervisors** (monitor the process behaviour)
- **gen_server** (used to implement processes)

**gen_server**
stores recovery tables

**protocol specification**
Enabling Protocol Recovery in Erlang: Example

```
% Handlers for C and D
quote({msg, Val}, State) →
  role:send(State#state.role, ?E, quote, Val).

% Handlers for E
quote({msg, Val}, State) when State.prev == undef →
  {noreply, State#state{prev=Val}};
quote({msg, Val}, State) when State#state.prev > Val →
  role:send(State#state.role, ?C, reject, empty),
  role:send(State#state.role, ?D, accept, empty),
  {noreply, State};
quote({msg, Val}, State) when State#state.prev < Val →
  role:send(State#state.role, ?C, accept, empty),
  role:send(State#state.role, ?D, reject, empty),
  {noreply, State}.
```

```
global protocol Trading (role A, role B, role C, role D) {
  quote(int) from A to C;
  quote(int) from B to D;
  quote(int) from C to E;
  quote(int) from D to E;
  choice at E {
    accept() from E to C;
    accept() from E to D;
    or {
      reject() from E to C;
      reject() from E to D;
    }
  }
}```
Evaluation: Web Crawler Example

- A process is chosen at random at the start
- Improvement when several failures occur
- By mistake initially we implemented all-for-one that introduced a deadlock

source: http://foat.me/articles/crawling-with-akka/
Evaluation: Concurrency Patterns

- 52% improvement when
  - intense local computation
  - disconnected interactions
- Up to 7% overhead when all roles are restarted

<table>
<thead>
<tr>
<th>Example</th>
<th>#roles</th>
<th>#states</th>
<th>GRT (sec)</th>
<th>affected roles</th>
</tr>
</thead>
<tbody>
<tr>
<td>MapReduce [21]</td>
<td>n+1</td>
<td>n+2</td>
<td>0.11</td>
<td>W[1] ... W[n]</td>
</tr>
<tr>
<td>Ring [21]</td>
<td>n</td>
<td>2*n</td>
<td>0.16</td>
<td>W[1] ... W[n]</td>
</tr>
<tr>
<td>Calculator [18]</td>
<td>n+1</td>
<td>4*n</td>
<td>0.75</td>
<td>A[1]</td>
</tr>
</tbody>
</table>

![Bar graph showing performance comparison for Ring, MapReduce, and Calculator.](image-url)
Future work & Resources

Framework summary

- Ensure processes are safe and conform to a protocol (even in cases of failures)
- Create supervision trees and link processes dynamically based on a protocol structure

Future work

- Support for stateful processes
- Integration with checkpoints
- Replications and recovery actions

Additional Resources

- Scribble webpage: scribble.doc.ic.ac.uk
- Project source: https://gitlab.doc.ic.ac.uk/rn710/codeINspire
- MRG webpage: http://mrg.doc.ic.ac.uk/
Q & A

THANK YOU FOR CRASHING MY PARTY
Future work & Resources

Framework summary

- Ensure processes are safe and conform to a protocol (even in cases of failures)
- Create supervision trees and link processes dynamically based on a protocol structure

Future work

- Support for stateful processes
- Integration with checkpoints
- Replications and recovery actions

Additional Resources

- Scribble webpage: scribble.doc.ic.ac.uk
  Project source: https://gitlab.doc.ic.ac.uk/rn710/codeINspire
  MRG webpage: http://mrg.doc.ic.ac.uk/