Let it Recover: Multiparty Protocol-Induced Recovery

Rumyana Neykova, Nobuko Yoshida
Imperial College London
Session Type Mobility Group

www.mrg.doc.ic.ac.uk
Us ∈ Mobility Research Group

http://mrg.doc.ic.ac.uk/
• **TCS’16**: Monitoring Networks through Multiparty Session Types. Laura Bocchi, Tzu-Chun Chen, Romain Demangeon, Kohei Honda, Nobuko Yoshida

• **LMCS’16**: Multiparty Session Actors. Rumyana Neykova, Nobuko Yoshida

• **FMSD’15**: Practical interruptible conversations: Distributed dynamic verification with multiparty session types and Python. Romain Demangeon, Kohei Honda, Raymond Hu, Rumyana Neykova, Nobuko Yoshida

• **TGC’13**: The Scribble Protocol Language. Nobuko Yoshida, Raymond Hu, Rumyana Neykova, Nicholas Ng
Scribble: Describing Multi Party Protocols

Scribble is a language to describe application-level protocols among communicating systems. A protocol represents an agreement on how participating systems interact with each other. Without a protocol, it is hard to do meaningful interaction: participants simply cannot communicate effectively, since they do not know when to expect the other parties to send data, or whether the other party is ready to receive data. However, having a description of a protocol has further benefits. It enables verification to ensure that the protocol can be implemented without resulting in unintended consequences, such as deadlocks.
Online tool: [http://scribble.doc.ic.ac.uk/](http://scribble.doc.ic.ac.uk/)

```plaintext
module examples;

  global protocol HelloWorld(role Me, role World) {
    hello() from Me to World;
    choice at World {
      goodMorning1() from World to Me;
    } or {
      goodMorning1() from World to Me;
    }
  }
```

Load a sample  
Check  
Protocol: examples.HelloWorld  
Role: Me  
Project  
Generate Graph
Interactions with Industries

Strange Loop
SEPTEMBER 15-17 2016 / PEABODY OPERA HOUSE / ST. LOUIS, MO

Adam Bowen @adamnbowen · Sep 15
I didn’t even know that session types existed an hour ago, but thanks to Nobuko Yoshida’s great talk at #pwliconf, I want to learn more.

Nobuko Yoshida
Imperial College, London

DoC researcher to speak at Golang UK conference
by Vicky Kapogianni
20 July 2016

Static deadlock detector
Tool developed based on our research
github.com/mingli/dgo-hunter
• Static (compile-time) detection of deadlock
• Help prevent deadlocks
• Optimize research
• Analyze common concurrency patterns & open-source projects

@nicholascwng rocking on @GolangUKconf about static deadlock detection in #golang #gouk16

DoC researcher to speak at industry-focused Golang UK conference on results of concurrency research

The Golang UK Conference
Interactions with Industries

F#unctional Londoners Meetup Group

6 days ago · 6:30 PM
Session Types with Fahd Abdeljallal

43 Members

Synopsis: Session types are a formalism to codify the structure of a communication, using types to specify the communication protocol used. This formalism provides the... [LEARN MORE]

Distributed Systems vs. Compositionality
Dr. Roland Kuhn
@rolandkuhn — CTO of Actyx

Current State
- behaviors can be composed both sequentially and concurrently
- effects are not yet tracked
- Scribble generator for Scala not yet there
- theoretical work at Imperial College, London (Prof. Nobuko Yoshida & Alceste Scalas)
Selected Publications 2016/2017

- [CC’16] Nicholas Ng, NY: Static Deadlock Detection for Concurrent Go by Global Session Graph Synthesis.
- [POPL’16] Dominic Orchard, NY: Effects as sessions, sessions as effects.
Selected Publications 2016/2017


[CC’16] Nicholas Ng, NY: Static Deadlock Detection for Concurrent Go by Global Session Graph Synthesis.


Let’s Start
Let it Recover:
Multiparty Protocol-Induced Recovery
“Fail fast and recover quickly”
Erlang proverb

“Fail fast and recover quickly and safely”
CC proverb (after this talk)
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)**
  \[ G = A \rightarrow B : \langle U_1 \rangle . B \rightarrow C : \langle U_2 \rangle . C \rightarrow A : \langle U_3 \rangle \]

- **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.
Causalities

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

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

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

\[ A \rightarrow B; B \rightarrow C; \]

\(n_1 \prec_{io} n_2\)

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

\[ A \rightarrow B; C \rightarrow B; \]

\(n_3 \triangleright n_4\) \(n_3 \not\prec_{io} n_4\)

\(\prec^\dagger\) -guarded dependencies (represent dependencies of the failed node) should not recover

\[ A \rightarrow B; B \rightarrow C; \]

\(n_1 \prec^\dagger n_2\)
Part Three
Recovery Algorithm
Algorithm Calculating affected nodes

Input: $n_i$ (a failed node), $p$ (a failed role)
Output: $N$ (a set of affected nodes)

1. $N = N^\rightarrow = \{n | n_i < n \land n = r \rightarrow p\} \cup \{n_i\}$
2. $\mathcal{S} = \{n | ((n_i < n' \land n' = p \rightarrow r) \lor n' = n_i) \land n' \leftarrow I_0 n\} \setminus \{n_i\}$
3. repeat
4. $N^\leftarrow = \{n | n \leftarrow I_0 n' \lor (n < n' \land n \in \mathcal{S}) \land n' \in N^\rightarrow\}$
5. $N^\rightarrow = \{n | n' < n \land n' \in N^\leftarrow\} \setminus (N \cup \mathcal{S})$
6. $N = N \cup N^\leftarrow \quad \mathcal{S} = \mathcal{S} \setminus N^\leftarrow$
7. until $N^\leftarrow = N^\rightarrow = \emptyset$
8. return $N$
Recovery Algorithm

- Step 1: Initialise the $\prec \dagger$ dependencies of the failed node
- Step 2: Backward traversal of $\prec_{io}$ dependencies
- Step 3: Forward Traversal of $\ll$ dependencies
- Step 4: Repeat 2-3 until no new dependencies are added
Step 1: Initialise the $\prec_{\dagger}$ dependencies of the failed node
Step 2: Backward traversal of $\prec_{io}$ dependencies
Step 3: Forward Traversal of $\prec$ dependencies
Step 4: Repeat 2-3 until no new dependencies are added

1: $B \rightarrow E$; 2: $C \rightarrow E$;
3: $B \rightarrow A$; 4: $C \rightarrow A$; 5: $A \rightarrow D$;
6: $D \rightarrow E$; 7: $B \rightarrow E$;

Initialise
$\prec_{\dagger}$: 5, 6, 7

$\prec_{io}$
3
4

$\prec$
3, 4
4

Final condition
3, 4
3, 4
not done
done
**Recovery points**

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

1  2  3  4

- Global Recovery Table

<table>
<thead>
<tr>
<th>Failure</th>
<th>Recovery points</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: B → C; 2: C → E; 3: B → A; 4: C → A;</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>3, A</td>
<td>A:3, B:3, C:4</td>
</tr>
<tr>
<td>3, B</td>
<td>A:3, B:3, C:5</td>
</tr>
<tr>
<td>4, C</td>
<td>C:2, E:2</td>
</tr>
<tr>
<td>4, A</td>
<td>C:1, B:1, ...</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

```erlang
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;
    }
  }
}

% 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#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}.
```
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
Evaluation: Concurrency Patterns

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

**Examples**

<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>
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