Computing Optimal Self-Repair Actions: Damage Minimization versus Repair Time

Matthias Tichy, Holger Giese, Daniela Schilling, Wladimir Pauls
Motivation

www.railcab.de
Motivation

- Redundant implementations of important software components

- Required: reconfiguration
- Given: automatism to detect failed components
- Self-Repair Actions: automatic calculation of redeployment for failed components
Initial Deployment

- Map deployment constraints given as extended UML Deployment Diagrams to inequalities over boolean and integer variables
- Use constraint solver to calculate initial deployment

WOSS/FSE 2004:
Matthias Tichy, Daniela Schilling, Holger Giese: *Design of Self-Managing Dependable Systems with UML and Fault Tolerance Patterns*
Online Redeployment

- Node crash failure ⇒ all components running on this node fail too
- Compute Self-Repair Action
  - Find suitable nodes to redeploy failed components
- How to find suitable nodes?
- What to do if there is no suitable node?
  - Redeploy further (still running) components
- Damage: negative effects of unavailable components
- Costs
- Goal: minimize costs
  - Keep damage as low as possible
  - Reduce solving time

Components to be migrated
Calculating redeployment
Performing redeployment

Damage
Costs
Time
Components to be migrated

Daniela Schilling - May 2005
Online Redeployment
- 1. Solution -

- Remove crashed nodes from constraint system
- Solve complete constraint system again
Online Redeployment - 2. Solution -

- Remove crashed nodes from constraint system
- Add objective function (minimize damage caused by migration of running components) to the constraint system
- Solve complete system again
Online Redeployment
- Our Approach -

- Remove crashed nodes from constraint system
- Add objective function (minimize damage) to the constraint system
- Try to solve constraint systems for failed components only
- Until a solution is found: extend set of components that have to be redeployed/migrated
- Use Constraint solver
- Heuristic approach
Online Redeployment
- Our Approach -

damage

time
Choosing Components for Redeployment

- Example: 3 redundant copies of important components

- Algorithm:
  - Try to redeploy failed component
  - Until redeployment is possible:
    1. Choose components which are no redundant copies of failed components
    2. Choose components where only one of three redundant copies already failed
    3. Choose arbitrary components
Choosing Components for Redeployment

- Example: 3 redundant copies of important components

- Algorithm:
  - Try to redeploy failed component
  - Until redeployment is possible:
    1. Choose components which are no redundant copies of failed components
    2. Choose components where only one of three redundant copies already failed
    3. Choose arbitrary components
Experiment

- Scenario:
  - 36 nodes with 114 links
  - 72 components with 99 connectors
  - 5 node-specific (CPU, OS, Memory, Utilization, HDD) and 2 link-specific (Bandwidth, Loss) deployment restrictions
  - set of deployment constraints on components and connectors

- Experiment:
  - Randomly selected a node and let it fail
## Experimental Results

<table>
<thead>
<tr>
<th>Test Nr.</th>
<th>1. Solution</th>
<th>2. Solution</th>
<th>Our Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time (ms)</td>
<td>Damage</td>
<td>Time (ms)</td>
</tr>
<tr>
<td>1</td>
<td>13630</td>
<td>773</td>
<td>&gt; 1h</td>
</tr>
<tr>
<td>2</td>
<td>14890</td>
<td>97</td>
<td>56060</td>
</tr>
<tr>
<td>3</td>
<td>13790</td>
<td>4</td>
<td>14920</td>
</tr>
<tr>
<td>4</td>
<td>13660</td>
<td>34</td>
<td>16430</td>
</tr>
</tbody>
</table>
Conclusion & Future Work

- Algorithm to calculate optimal self-repair actions
- Deployment constraints solved by standard constraint solver
- Experiment showed that algorithm is nearly optimal in damage minimization and time consumption
- Not presented: pre-solving step
- Communication and monitoring framework
- Describe repair rules by graph transformation systems
Appendix
Simple Redeployment
Example
Damage Calculation

damage = 13

damage:
- all = 13
- 2 of 3 = 4
- 1 of 3 = 1
Submodel Expansion

- **Initial situation**
  - Failed components: \(a\), \(b\), \(c\)
  - Running components: \(d\), \(e\), \(f\), \(g\)

1) Submodel: \(a\), \(b\), \(c\)
   - Consider: \(d\), \(e\), \(f\), \(g\)
   - Consider later: 

Submodel not solvable

2) \(a\), \(b\), \(c\)
   - \(d\), \(e\), \(f\), \(g\)
   - Redundant copies

3) \(a\), \(b\), \(c\)
   - \(e\), \(f\), \(g\)
   - Not related
   - \(d\)

4) \(a\), \(b\), \(c\), \(e\)
   - \(f\), \(g\)
   - Not related
   - \(d\)

Submodel not solvable
Submodel Expansion (2)

Failed components | Running components
---|---
4) \( \{a, b, c, e\} \) | \( \{f, g\} \) | \( \{d\} \)

Submodel not solvable

5) \( \{a, b, c, e\} \) | \( \{f, g\} \) | \( \{d\} \)

Redundant copies

6) \( \{a, b, c, e\} \) |  | \( \{d, f, g\} \)

7) \( \{a, b, c, e, d\} \) |  | \( \{f, g\} \)

Submodel solvable
Foundations (TMR)

- Use fault tolerance techniques to ensure dependability
- Triple Modular Redundancy (TMR)
### Deployment constraints for TMR

Avoid single-point-of-failure of voter / multiplier
- Deploy voter and user to same node
  (if the user fails, the failure of the voter is no problem)

Avoid crash failures
- Deploy redundant components to distinct nodes

Heterogeneous hardware platform
- Require different CPU for components

Node 1:
- Provider
- Multiplier

Node 2:
- Voter
- User

Node 3, 4, 5:
- Component 1
- Component 2
- Component 3

Questions?
Online Redeployment
- Our Solution -

- Compute Self-Repair Action
  - Find suitable nodes to redeploy failed components

- How to find suitable nodes?

- What to do if there is no suitable node?
  2) Redeploy further (still running) components

- Goal: reduce costs
  - Redeployment should not decrease dependability (reduce damage)
  - Reduce solving time