Automatic Recovery From Runtime Failures

By Antonio Carzaniga, et al.

Presented By Ben Cheung and Vishnu Prasad
Motivations

- Amazon Store Server Crash
- Day Trading

- There are bugs everywhere, some known, some unknown.
  - “Whatever can go wrong, will go wrong” - Murphy’s Law
  - When the unknown bugs strike, will you be ready? Will you be safe?
Background - Previous Related Work

- Running copies of the system for fault tolerance
  - “The N-version approach to fault-tolerant software”, A. Acizienis
  - “System structure for software fault tolerance”, B. Randell

- Expensive to implement
- Inefficient because of correlating faults
Other works addressed issues in specific areas.

- Data Structures
- Configuration Incompatibilities
- Infinite Loops

- Too Direct, not general enough of a technique.
Research Questions

- Is there a way to correct or avoid runtime errors on the fly?
- Is this possible to do without incurring a large overhead time?
- Is it possible to do this generally?
Is there a way to correct runtime errors on the fly?

- Libraries are redundant
- Exploit these redundancies to find workarounds
- Replace error-causing code with workarounds
What is a Workaround?

- Semantically equivalent code
- Different implementation
- Product of redundancy
- Identified manually
Simplified Example

// Two semantically identically methods that might exist in a library

getA(int first, int second){
    int index = (first + second) % 10;
    return array[index];
}

getB(int first, int second){
    int index = ((first % 10) + (second % 10)) % 10;
    return array[index];
}

List.getA(2147483646, 10); // fails
List.getB(2147483646, 10); // replacement succeeds
Preprocessing Step

1. Identify Roll-back Areas (RBA)
   a. Library Calls
   b. Each RBA will be a checkpoint for roll back.

2. Prepare workarounds for each RBA
1. Checkpointing at RBAs
2. When error is thrown:
   a. RBA replaced with an unused workaround
   b. Rollback to checkpoint, continue execution
3. 
   a. No more errors: program continues or finishes
   b. No more workarounds to try: program ends unsuccessfully
Overhead Cost?

- Checkpointing costs
- Rollback, replacement costs
- Re-doing execution
Evaluation - Setup and Problem

- ARMOR System
- 2 Libraries: JodaTime, Guava
- Used on 4 Applications:
  - Fb2pdf
  - Carrot2
  - Caliper
  - Closure
Armor Pre-processing

- 63 Rewriting rules for Guava
- 100 Rewriting rules for JodaTime

<table>
<thead>
<tr>
<th>Application</th>
<th>Caliper</th>
<th>Carrot2</th>
<th>Closure</th>
<th>Fb2pdf</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total RBAs</td>
<td>130</td>
<td>139</td>
<td>2099</td>
<td>17</td>
</tr>
<tr>
<td>RBAs with variants</td>
<td>60</td>
<td>106</td>
<td>687</td>
<td>17</td>
</tr>
</tbody>
</table>
Effectiveness

- 19%-48% Effective
- Avoiding Runtime Errors is possible

<table>
<thead>
<tr>
<th>Execution</th>
<th>Caliper</th>
<th>Carrot2</th>
<th>Closure</th>
<th>Fb2pdf</th>
</tr>
</thead>
<tbody>
<tr>
<td>success</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>non-equivalent</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>equivalent</td>
<td>210</td>
<td>120</td>
<td>177</td>
<td>1805</td>
</tr>
<tr>
<td>detected</td>
<td>0</td>
<td>8</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>not detected</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>loop</td>
<td>12</td>
<td>9</td>
<td>15</td>
<td>47</td>
</tr>
<tr>
<td>detected</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>not detected</td>
<td>87</td>
<td>47</td>
<td>149</td>
<td>347</td>
</tr>
<tr>
<td>error</td>
<td>87</td>
<td>50</td>
<td>149</td>
<td>347</td>
</tr>
<tr>
<td>Mutants where ARMOR is successful</td>
<td>(28%)</td>
<td>(48%)</td>
<td>(47%)</td>
<td>(19%)</td>
</tr>
<tr>
<td>Mutants run with ARMOR</td>
<td>87</td>
<td>50</td>
<td>149</td>
<td>347</td>
</tr>
</tbody>
</table>
## Runtime Overhead

- Overhead ranges from 1%-194%
- A 194% overhead to avoid runtime errors may be worth the tradeoff

<table>
<thead>
<tr>
<th>Time (seconds)</th>
<th>Caliper</th>
<th>Carrot2</th>
<th>Closure</th>
<th>Fb2pdf</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original total running time</td>
<td>30.13</td>
<td>2.43</td>
<td>5.40</td>
<td>2.26</td>
</tr>
<tr>
<td>Exception-handling only (no checkpoints)</td>
<td>(1%) 30.41</td>
<td>(69%) 4.15</td>
<td>(95%) 10.53</td>
<td>(68%) 3.79</td>
</tr>
<tr>
<td>Snapshot-based checkpoints</td>
<td>(5%) 31.78</td>
<td>(117%) 5.32</td>
<td>&gt;1h</td>
<td>(121%) 4.99</td>
</tr>
<tr>
<td>Change-log-based checkpoints</td>
<td>(2%) 30.87</td>
<td>(94%) 4.75</td>
<td>(194%) 15.90</td>
<td>(114%) 4.70</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Memory (MB)</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Original total memory allocated</td>
<td>1.40</td>
<td>8.87</td>
<td>30.56</td>
<td>17.90</td>
</tr>
<tr>
<td>Snapshot-based checkpoints</td>
<td>12.30</td>
<td>23.78</td>
<td>—</td>
<td>90.94</td>
</tr>
<tr>
<td>Change-log-based checkpoints</td>
<td>10.18</td>
<td>11.37</td>
<td>120.58</td>
<td>25.93</td>
</tr>
</tbody>
</table>

| Number of recorded checkpoints (approx.)| 30     | 2,350   | 1,255,000| 4      |
|Values saved in change-log-based checkpoints (approx.)| 26,000 | 270,000 | 1,880,000| 9,000 |
How Is It Better?

- Less costly than Replicated server methods.
  - No copies
  - Retries with different variations of the same method.

- More General and extensible
  - Other work directed at Data Structures, infinite loops, configuration incompatibilities etc.
  - Generic because it finds workarounds inherent in the libraries
  - Can work for most libraries with redundancy
Contributions

1. ARMOR

2. Technique using workarounds and rollbacks
1. Pre-processing needs to be done for each library individually: is this feasible?
Discussion Questions

2. What downsides can you foresee in this research?
Discussion Questions

3. Exactly how redundant is the typical library?
Discussion Questions

4. ARMOR uses runtime exceptions to detect errors: how does it ignore exceptions which the developer catches and handles themselves?
Discussion Questions

5. Would it be feasible to make the code replacements permanent instead of dynamically-inserted for the purpose of making the system more error resistant?