Effective and Efficient Localization of Multiple Faults using Value Replacement

Dennis Jeffrey
jeffreyd@cs.ucr.edu

Neelam Gupta
guptajneelam@gmail.com

Rajiv Gupta
gupta@cs.ucr.edu

Presented by Dennis Jeffrey

Department of Computer Science and Engineering
The University of California, Riverside

25th IEEE International Conference on Software Maintenance
Edmonton, Alberta, Canada
September 24, 2009
Value Replacement: Overview

- **Dynamic state alteration technique** to locate faulty program statements [Jeffrey et. al., ISSTA 2008]

**INPUT:**
Faulty program and test suite (1+ failing runs)

**TASK:**
(1) Perform value replacements in failing runs
(2) Rank program statements according to collected information

**OUTPUT:**
Ranked list of program statements
Alter State by Replacing Values

Passing Execution

Correct Output
Alter State by Replacing Values

Failing Execution

ERROR

Incorrect Output
Failing Execution: Altered State

Statement: S
Instance: I
Original Set of Values: ORIG
Alternate Set of Values: ALT

If correct: Stmt S, Instance I: ORIG $\rightarrow$ ALT

Effective and Efficient Localization of Multiple Faults using Value Replacement
Value Replacement: Details

- **Replace values** at different statement instances in failing runs to search for IVMPs
  - Statements with an IVMP in more failing runs are more likely to be faulty
  - Alternate value sets taken from profiling info

- **Rank** program statements in decreasing order of **suspicousness**
  - **Suspicousness**: the # of failing runs in which the given statement is associated with an IVMP
Value Replacement: Example

1: read (x, y);
2: a := x + y;
3: if (x < y)
   4:   write (a);
   else
5:   write (a + 1);

<table>
<thead>
<tr>
<th>Test Case (x, y)</th>
<th>Actual Output</th>
<th>Expected Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>[A] (1, 1)</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>[B] (0, 1)</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td>[C] (-1, 0)</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>[D] (0, 0)</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Test Case [A] IVMPs:
stmt 1, inst 1: ( {x=1, y=1} → {x=0, y=1} )
stmt 1, inst 1: ( {x=1, y=1} → {x=0, y=0} )
stmt 2, inst 1: ( {x=1, y=1, a=2} → {x=0, y=1, a=1} )
stmt 2, inst 1: ( {x=1, y=1, a=2} → {x=0, y=0, a=0} )
stmt 5, inst 1: ( {a=2, output=3} → {a=0, output=1} )

Test Case [B] IVMPs:
stmt 1, inst 1: ( {x=0, y=1} → {x=-1, y=0} )
stmt 2, inst 1: ( {x=0, y=1, a=1} → {x=-1, y=0, a=-1} )
stmt 4, inst 1: ( {a=1, output=1} → {a=-1, output=-1} )

stmts with IVMPs: {1, 2, 5}

stmts with IVMPs: {1, 2, 4}

Most likely to be faulty: {1, 2} → {4, 5} → {3}
Least likely to be faulty:

Effective and Efficient Localization of Multiple Faults using Value Replacement
Value Replacement: Results

- Highly Effective
  - Precisely locates 39 / 129 errors (30.2%)
  - Most effective previously known: 5 / 129 (3.9%)

- Limitations
  - Assumes multiple failing runs are caused by the same error
  - Can require significant computation time to search for IVMPs
Handling Multiple Errors: Goals

- **Effectively** locate multiple simultaneous errors
  - Iteratively compute a ranked list of statements to find and fix one error at a time
  - Three variations of this technique
    - MIN
    - FULL
    - PARTIAL

- **Efficiently** search for IVMPs
  - Improve efficiency without impacting effectiveness
  - Two motivating observations
    - Redundancy
    - Independence
Multiple-Error Techniques

**Single Error**
- Faulty Program and Test Suite
  - Value Replacement
    - Ranked List of Program Statements
      - Developer Find/Fix Error
        - Done

**Multiple Errors (MIN)**
- Faulty Program and Test Suite
  - Value Replacement
    - Ranked List of Program Statements
      - Developer Find/Fix Error
      - Failing Run Remains?
        - Yes
          - Developer Find/Fix Error
        - No
          - Done
Multiple-Error Techniques

**Multiple Errors (FULL)**

1. Faulty Program and Test Suite
   - Value Replacement
     - Ranked List of Program Statements
     - Developer Find/Fix Error
       - Failing Run Remains? (Yes/No)
         - Done

**Multiple Errors (PARTIAL)**

1. Faulty Program and Test Suite
   - Partial Value Replacement
     - Ranked List of Program Statements
     - Developer Find/Fix Error
       - Failing Run Remains? (Yes/No)
         - Done
PARTIAL Technique

Step 1: Initialize ranked lists and locate first error
- For each statement $s$, compute a ranked list by considering only failing runs exercising $s$
- Report ranked list with highest suspiciousness value at the front of the list

Step 2: Iteratively revise ranked lists and locate each remaining error
- For each remaining failing run that exercises the statement just fixed, recompute IVMPs
- Update any affected ranked lists
- Report ranked list with the most different elements at the front of the list, compared to previously-selected lists
PARTIAL Technique: Example

Program (2 faulty statements)

Failing Run | Execution Trace | Statements with IVMPs
---|---|---
A | (1, 2, 3, 5) | {2, 5}
B | (1, 2, 3, 5) | {1, 2}
C | (1, 2, 4, 5) | {2, 4, 5}

Computed Ranked Lists: (statement suspiciousness)

1. 2, 5, 1, 4, 3 [based on runs A, B, C]
2. 2, 5, 1, 4, 3 [based on runs A, B, C]
3. 2, 1, 5, 3, 4 [based on runs A, B]
4. 2, 4, 5, 1, 3 [based on run C]
5. 2, 5, 1, 4, 3 [based on runs A, B, C]

Report list 1, 2, or 5 (assume 1) ➔ Fix faulty statement 2

Effective and Efficient Localization of Multiple Faults using Value Replacement
PARTIAL Technique: Example

Program (1 faulty statement)

1 → 2 → 3 → 4 → 5

<table>
<thead>
<tr>
<th>Failing Run</th>
<th>Execution Trace</th>
<th>Statements with IVMPs</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>(1, 2, 4, 5)</td>
<td>{4}</td>
</tr>
</tbody>
</table>

Computed Ranked Lists: \(\text{statement}_{\text{suspiciousness}}\)

1. 2, 1, 4, 5, 3, 0 [based on runs A, B, C] (C updated)
2. 2, 1, 5, 3, 0, 4 [based on runs A, B] (no updates)
3. 4, 1, 0, 2, 3, 0, 5 [based on run C] (C updated)
4. 2, 1, 4, 5, 3, 0 [based on runs A, B, C] (C updated)

Report list 4
→ Fix faulty statement 4
→ Done
Improving Efficiency of IVMP Search

Original Execution

- stmt instance 1
- stmt instance 2
- stmt instance 3

(assume 2 alternate value sets at each stmt instance)

Regular Value Replacement Executions

- stmt instance 1
- stmt instance 2
- stmt instance 3

(value replacements are independent of each other)
(assumes portions of original execution are duplicated multiple times)

Efficiency Improvements:

1. Fork child process to do each value replacement in original failing execution
2. Perform value replacements in parallel
Improving Efficiency of IVMP Search

With Redundant Execution Removed

With Parallelization

(no duplication of any portion of original execution)

(total time required to perform all value replacements is reduced)
### Original Siemens Benchmark Programs

<table>
<thead>
<tr>
<th>Program</th>
<th>LOC</th>
<th># Faulty Ver.</th>
<th>Test Case Pool Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>tcas</td>
<td>138</td>
<td>41</td>
<td>1608</td>
</tr>
<tr>
<td>totinfo</td>
<td>346</td>
<td>23</td>
<td>1052</td>
</tr>
<tr>
<td>sched</td>
<td>299</td>
<td>9</td>
<td>2650</td>
</tr>
<tr>
<td>sched2</td>
<td>297</td>
<td>9</td>
<td>4130</td>
</tr>
<tr>
<td>ptok</td>
<td>402</td>
<td>7</td>
<td>4130</td>
</tr>
<tr>
<td>ptok2</td>
<td>483</td>
<td>9</td>
<td>4115</td>
</tr>
<tr>
<td>replace</td>
<td>516</td>
<td>31</td>
<td>5542</td>
</tr>
</tbody>
</table>
### Experimental Benchmark Programs

<table>
<thead>
<tr>
<th>Program</th>
<th># 5-Error Faulty Versions</th>
<th>Average Suite Size (# Failing Runs / # Passing Runs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>tcas</td>
<td>20</td>
<td>11 (5 / 6)</td>
</tr>
<tr>
<td>totinfo</td>
<td>20</td>
<td>22 (10 / 12)</td>
</tr>
<tr>
<td>sched</td>
<td>20</td>
<td>29 (10 / 19)</td>
</tr>
<tr>
<td>sched2</td>
<td>20</td>
<td>30 (9 / 21)</td>
</tr>
<tr>
<td>ptok</td>
<td>2</td>
<td>32 (8 / 24)</td>
</tr>
<tr>
<td>ptok2</td>
<td>11</td>
<td>29 (5 / 24)</td>
</tr>
<tr>
<td>replace</td>
<td>20</td>
<td>38 (9 / 29)</td>
</tr>
</tbody>
</table>

- Each faulty program contains 5 seeded errors, each in a different stmt.
- Each faulty program is associated with a stmt-coverage adequate test suite such that at least one failing run exercises each error.
Techniques Compared

- (MIN) Only compute ranked list once
- (FULL) Fully recompute ranked list each time
- (PARTIAL) Compute IVMPs for subset of failing runs and revise ranked lists each time
- (ISOLATED) Locate each error in isolation
Metric for Comparison

- **Score** for each ranked statement list

\[
\frac{\text{size of list} - \text{rank of the faulty stmt}}{\text{size of list}} \times 100\%
\]

- Represents percentage of statements that need not be examined before error is located
- Higher score is better

---

Effective and Efficient Localization of Multiple Faults using Value Replacement
Effectiveness Results

Effectiveness Comparison of Value Replacement Techniques

![Bar chart showing effectiveness comparison of value replacement techniques]

- **Avg. Score per Ranked List (%)**
- **Effectiveness Comparison**: Isolated, Full, Partial, Min

---

Effective and Efficient Localization of Multiple Faults using Value Replacement 21 / 26
Efficiency Results

Time to Search for IVMPs for Each Faulty Program

Multiple errors can be located in minutes.

Previously, single errors could be located in hours.

Effective and Efficient Localization of Multiple Faults using Value Replacement 22 / 26
Prior Techniques for Locating Errors

- Program slicing
  - Pruning Dynamic Slices with Confidence [Zhang et. al. PLDI 2006]
  - Failure-Inducing Chops [Gupta et. al. ASE 2005]

- Invariant-based techniques
  - Daikon [Ernst et. al. IEEE TSE Feb. 2001]
  - AccMon [Zhou et. al. HPCA 2007]
Prior Techniques for Locating Errors

- Statistical techniques
  - **Cooperative Bug Isolation** [Ben Liblit doctoral dissertation, 2005]
  - **SOBER** [Liu et. al. FSE 2005]
  - **Tarantula** [Jones et. al. ICSE 2002]

- State-alteration techniques

---

Effective and Efficient Localization of Multiple Faults using Value Replacement
Prior State Alteration Techniques

  - Search in space for values relevant to a failure
  - Search in time for failure cause transitions

- **Predicate Switching** [Zhang et. al. ICSE 2006]
  - Alter predicate outcome to correct failing output

- **Execution Suppression** [Jeffrey et. al. ICSM 2008]
  - Suppress effects of memory corruption during execution to locate root causes of memory errors
Conclusions

- 3 technique variants to locate multiple errors
  - Minimal computation (more efficient, less effective)
  - Full recomputation (more effective, less efficient)
  - Partial recomputation (more balanced effectiveness/efficiency)

- 2 techniques to improve efficiency of IVMP search
  - Remove redundant program execution
  - Parallelize the search

Multiple simultaneous errors can be effectively located in minutes in the benchmark programs.

Previously, single errors could be effectively located in hours using the same benchmarks.