PerfBlower: A Novel Performance Testing Framework based on Virtual Amplification

Lu Fang, UCI
Liang Dou, UCI
Harry Xu, UCI
Outline

• Motivation
• Challenges
• Our Solution
  • Virtual Amplification
  • SDL
  • Mirror Chain
• Evaluation
• Conclusion
Motivation

- Performance problems
  - Can lead to severe problems
  - Difficult to find and fix

- Existing solutions
  - Most are postmortem debugging techniques
Challenges

• Challenge 1
  • The lack of effective test oracles

• Challenge 2
  • The lack of a general specification

• Challenge 3
  • The lack of effective debugging support
Our Solutions

• PerfBlower
  • A novel general performance testing framework
  • General Idea: Amplify performance problems
  • Virtual Amplification
    • Provide test oracles (for challenge 1)
  • SDL (short for Symptom Description Language)
    • Describe memory performance problems (for challenge 2)
• Mirror Chain
  • Provide useful debugging information (for challenge 3)
Contribution 1: Virtual Amplification

- Amplification
  - Add space penalties to suspicious objects
  - Make the symptoms more obvious
  - Deamplification

- Virtual
  - Counter per object

- Virtual Space Overhead
  - \((P+S)/S\)
    - P is the sum of penalty counters of all the live objects
    - S is the size of the live heap

- Test oracle
Contribution 2: SDL

- Symptom Description Language
  - Performance problems manifested by memory inefficiency
- A simple, event-based language
  - Describe symptoms
  - Specify the corresponding actions
- An SDL Program consists of:
  - **Context**
    ```
    Context ArrayContext {
      sequence = "Main.main,*,
                   JbbTransaction.start,*";
      type = "Object[]";
    }
    ```
  - History
  - Partition
  - TObject
  - Event
Contribution 2: SDL

- Symptom Description Language
  - Performance problems manifested by memory inefficiency
- A simple, event-based language
  - Describe symptoms
  - Specify the corresponding actions
- An SDL Program consists of:
  - Context
  - History
  - Partition
  - TObject
  - Event

```c
History UseHistory {
    type = "boolean";
    size = UP;  // User Parameter
}
```
Contribution 2: SDL

- Symptom Description Language
  - Performance problems manifested by memory inefficiency
- A simple, event-based language
  - Describe symptoms
  - Specify the corresponding actions
- An SDL Program consists of:
  - Context
  - History
  - **Partition**
  - Tobject
  - Event

```cpp
Partition P {
    kind = all;
    history = UseHistory;
}
```
Contribution 2: SDL

- Symptom Description Language
  - Performance problems manifested by memory inefficiency
- A simple, event-based language
  - Describe symptoms/counter-evidences
  - Specify the corresponding actions
- An SDL Program consists of:
  - Context
  - History
  - Partition
  - TObject
  - Event

```cpp
TObject MyObj{  
  include = TrackingContext;  
  partition = P;  
}
Partition P {  
  kind = all;  
  history = UseHistory;  
}
```
Contribution 2: SDL

- Symptom Description Language
  - Performance problems manifested by memory inefficiency
- A simple, event-based language
  - Describe symptoms/counter-evidences
  - Specify the corresponding actions
- An SDL Program consists of:
  - Context
  - History
  - Partition
  - TObject
  - Event

```cpp
Event on_rw(Object o, Field f, Word w1, Word w2) {
    deamplify(o);
}
```
Contribution 3: Mirror chain

- Reference path
  - Very useful for debugging
  - Difficult to record
- Mirror chain
  - An efficient way to record the reference path
Evaluations

- We have implemented 4 amplifiers
  - Memory leak amplifier
  - Over-populated container (OPC) amplifier
  - Under-utilized container (UUC) amplifier
  - Never-used return object (NUR) amplifier
- Totally we have found 14 performance problems
  - 11 unknown problems
  - 3 known problems
VSOs reported by Memory Leak Amplifier

DaCapo Benchmarks

Programs with known leaks
VSOs reported by Memory Leak Amplifier

VSO

contain memory leaks
Comparison with SLEIGH [M. D. Bond et al. ASPLOS 2006]

<table>
<thead>
<tr>
<th>Bench</th>
<th>PerfBlower</th>
<th>SLEIGH</th>
</tr>
</thead>
<tbody>
<tr>
<td>antlr</td>
<td>no leak</td>
<td>no leak</td>
</tr>
<tr>
<td>bloat</td>
<td>no leak</td>
<td>no leak</td>
</tr>
<tr>
<td>eclipse</td>
<td>leaks$^+$</td>
<td>no leak</td>
</tr>
<tr>
<td>fop</td>
<td>no leak</td>
<td>no leak</td>
</tr>
<tr>
<td>hsqldb</td>
<td>leaks$^+$</td>
<td>leaks$^+$</td>
</tr>
<tr>
<td>jython</td>
<td>leaks$^+$</td>
<td>false positive</td>
</tr>
<tr>
<td>lindex</td>
<td>no leak</td>
<td>fail to run</td>
</tr>
<tr>
<td>lusearch</td>
<td>no leak</td>
<td>no leak</td>
</tr>
<tr>
<td>pmd</td>
<td>no leak</td>
<td>leaks$^+$</td>
</tr>
<tr>
<td>xalan</td>
<td>leaks$^+$</td>
<td>fail to run</td>
</tr>
<tr>
<td>mysql</td>
<td>leaks$^+$</td>
<td>leaks$^+$</td>
</tr>
<tr>
<td>mckoi</td>
<td>leaks$^+$</td>
<td>leaks$^+$</td>
</tr>
<tr>
<td>jbb</td>
<td>leaks$^+$</td>
<td>leaks$^+$</td>
</tr>
<tr>
<td>true leaks</td>
<td>4 (unknown) + 3 (known)</td>
<td>2 + 3</td>
</tr>
<tr>
<td>false positives</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>useful information</td>
<td>7</td>
<td>2</td>
</tr>
</tbody>
</table>

1. Find more problems
2. Report is more precise
3. Provide more useful information
VSOs reported by Under Utilized Container Amplifier

<table>
<thead>
<tr>
<th>VSO</th>
<th>antlr</th>
<th>fop</th>
<th>luindex</th>
<th>lusearch</th>
<th>pmd</th>
<th>bloat</th>
<th>eclipse</th>
<th>hsqldb</th>
<th>jython</th>
<th>xalan</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>2.6</td>
<td>3.3</td>
<td>1.7</td>
<td>1.5</td>
<td>1.4</td>
<td>5.9</td>
<td>8.7</td>
<td>16.6</td>
<td>48.3</td>
<td>117.7</td>
</tr>
</tbody>
</table>

Contains Under Utilized Container
VSOs reported by Over Populated Container Amplifier

VSO

<table>
<thead>
<tr>
<th>Library</th>
<th>VSO</th>
</tr>
</thead>
<tbody>
<tr>
<td>antlr</td>
<td>1.1</td>
</tr>
<tr>
<td>bloat</td>
<td>1.1</td>
</tr>
<tr>
<td>eclipse</td>
<td>3.5</td>
</tr>
<tr>
<td>fop</td>
<td>1.1</td>
</tr>
<tr>
<td>luindex</td>
<td>1.1</td>
</tr>
<tr>
<td>lusearch</td>
<td>1.2</td>
</tr>
<tr>
<td>pmd</td>
<td>1.1</td>
</tr>
<tr>
<td>xalan</td>
<td>3.4</td>
</tr>
<tr>
<td>hsqldb</td>
<td>29.2</td>
</tr>
<tr>
<td>jython</td>
<td>16.3</td>
</tr>
</tbody>
</table>

Contain Over Populated Container
Performance Improvement

- Fix the memory leaks
  - Xalan
    - 23.7% space reduction.
  - Jython
    - 6.1% space reduction
    - 7.4% warm up time reduction

- Fix the OPC
  - Hsqldb
    - 5.3% space reduction

- Fix the UUC
  - Bloat
    - 4.8% space reduction
  - Hsqldb
    - 8.1% space reduction
  - Jython
    - 3.3% space reduction
Conclusions

- Propose a novel general framework for performance testing
- Develop compiler and runtime system support
- Successfully amplify four different types of performance problems
  - Help developers find and fix performance problems even in the development environment
Thanks!

Q&A
Overview
Example 1: Penalty Counter

After time $t$, object $A$ becomes suspicious.
SDL Program for Memory Leak in Jbb

```
Context ArrayContext {
    sequence = "Main.main,*, JbbTransaction.start,*";
    type = "Object[]";
}

Context TrackingContext {
    sequence = "Main.main,*, JbbTransaction.start,*";
    path = ArrayContext;
    type = "String";
}

History UseHistory {
    type = "boolean";
    size = UP; // User Parameter
}

Partition P {
    kind = all;
    history = UseHistory;
}
```

```
TObject MyObj{
    include = TrackingContext;
    partition = P;
    instance boolean useFlag = false; // Instance Field
}

Event on_rw(Object o, Field f, Word w1, Word w2){
    o.useFlag = true;
    deamplify(o);
}

Event on_reachedOnce(Object o){
    UseHistory h = getHistory(o);
    h.update(o.useFlag);
    if(h.isFull() && !h.contains(true)){
        amplify(o);
    }
}
```
Support Events

- alloc
- read
- write
- rw
- call
- reached
- reachedOnce.