HeapMD: Identifying Heap-based Bugs using Anomaly Detection

Trishul M. Chilimbi
Microsoft Research
Redmond, WA
trishulc@microsoft.com

Vinod Ganapathy
University of Wisconsin
Madison, WI
vg@cs.wisc.edu

ASPLOS XII, October 2006
San Jose, California
A motivating example

```
pNewAsset = Initialize(pAssetParams);
...
if (pAssetList->next != NULL) {
    pNewAsset->next = pAssetList->next
    pAssetList->next = pNewAsset
}
```
A motivating example

```
pNewAsset = Initialize(pAssetParams);
...
if (pAssetList->next != NULL) {
    pNewAsset->next = pAssetList->next
    pAssetList->next = pNewAsset
}
```
Noteworthy points

• Violation of implicit data-structure invariant
  – Only extremal nodes must have indegree=1
• Malformed, but pointer-correct structure
  – Bug does not necessarily result in a crash
Key challenges

• Programmers rarely write down invariants for heap data structures
• Bugs may not be immediately apparent

Can we infer invariants for the heap and use them for bug detection?
Presenting HeapMD

• A tool to monitor the health of the heap

**Highlights of results**

HeapMD found 40 bugs (31 new) in 5 large commercial applications
Talk outline

• Motivation
• Stability of the heap
• Application to bug-finding
• Related work and conclusion
Heap data

If this were true in practice, building large software would be untenable

1. Invisible
   - No tangible representation

2. Can be arbitrarily modified
   - Akin to self-modifying code

3. Can have arbitrary structure
   - Akin to programming with gotos
Heap data

In practice, the heap has a simple, stable structure

1. Invisible
   - No tangible representation
2. Can be arbitrarily modified
   - Often only a small fraction is modified
3. Can have arbitrary structure
   - In practice, structure is simple
Stability of pointer-valued locations

Strong evidence that a large fraction of the heap is not modified
two values during their lifetime
Simple structure of the heap

• Most data structures in large programs have low in- and out-degrees
  – Linked lists
  – Trees
  – Hash tables

Can we quantify the simplicity and stability of the heap?
Simple metrics suffice

- % nodes with indegree = 0 (root nodes)
- % nodes with outdegree = 0 (leaf nodes)
- % nodes with indegree = 1
- % nodes with indegree = 2
- % nodes with outdegree = 1
- % nodes with outdegree = 2
- % nodes with indegree = outdegree
Gathering metrics: Setup

• Instrument program (x86) to track metrics

• Execute instrumented program on a set of inputs

• Gather at metric-computation points
  – Function entry points
  – Frequency is tunable: currently 1/100,000
% indegree = 2
% outdegree = 0 (leaves)
% indegree = outdegree
Algorithm to find stable metrics

Program

Calibration inputs

Compute metrics

Metric reports

Determine stability

Can potentially find bugs exercised by calibration inputs

Stable metrics and their ranges

Find range of metric

Is stable for 40% inputs?

Check for range violation

Other 60%

Metric unstable (discard)
Stable metrics exist: SPEC

<table>
<thead>
<tr>
<th>SPEC b/m</th>
<th># Inputs</th>
<th># Stable</th>
</tr>
</thead>
<tbody>
<tr>
<td>twolf</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>vpr</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>vortex</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>gzip</td>
<td>100</td>
<td>2</td>
</tr>
<tr>
<td>parser</td>
<td>100</td>
<td>3</td>
</tr>
<tr>
<td>gcc</td>
<td>100</td>
<td>2</td>
</tr>
</tbody>
</table>

% of vertices with Indegree = Outdegree
Range = [14.2%, 17.2%]
Stable metrics exist: Commercial

<table>
<thead>
<tr>
<th>Benchmark</th>
<th># Inputs</th>
<th># Stable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multimedia</td>
<td>50</td>
<td>2</td>
</tr>
<tr>
<td>Interactive web-app.</td>
<td>50</td>
<td>2</td>
</tr>
<tr>
<td>PC-game (simulation)</td>
<td>50</td>
<td>2</td>
</tr>
<tr>
<td>PC-game (action)</td>
<td>50</td>
<td>1</td>
</tr>
<tr>
<td>Productivity</td>
<td>50</td>
<td>2</td>
</tr>
</tbody>
</table>
Extending the observation

Several degree-based metrics of the heap-graph remain stable as the heap evolves

In fact, we observed that ...

Stable heap metrics exist even across different development versions of a program
## Metric stability across versions

<table>
<thead>
<tr>
<th>Benchmark</th>
<th># Versions</th>
<th># Inputs</th>
<th># Stable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multimedia</td>
<td>5</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>Interactive web-app.</td>
<td>5</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>PC-game (simulation)</td>
<td>5</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>PC-game (action)</td>
<td>5</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>Productivity</td>
<td>5</td>
<td>10</td>
<td>2</td>
</tr>
</tbody>
</table>
Talk outline

• Motivation
• Stability of the heap
• Application to bug-finding
• Related work and conclusion
Architecture of HeapMD

Calibration

Goal
Identify a subset of observed metrics as stable metrics

Bug-finding

Goal
Check the metrics identified by training for range violation
Finding bugs using HeapMD

Key intuition
Range violation → Likely bug
Recall our motivating example

```c
pNewAsset = Initialize(pAssetParams);
...
if (pAssetList->next != NULL) {
    pNewAsset->next = pAssetList->next;
    pAssetList->next = pNewAsset;
}
```

Violated invariant

Only extremal nodes have indegree=1

[Diagram showing the violation of the invariant with an arrow pointing to a node that is not extremal, labeled with an X.]

Range violation for this example

Indegree = 1 for PC Game (Action)

Log the call stack for diagnostics
## Summary of bugs found

<table>
<thead>
<tr>
<th>Benchmark</th>
<th># Bugs</th>
<th># New bugs</th>
<th># Invariant violations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multimedia</td>
<td>8</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>Interactive web-app</td>
<td>10</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>PC-game (simulation)</td>
<td>9</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>PC-game (action)</td>
<td>8</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>Productivity</td>
<td>5</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>40</strong></td>
<td><strong>31</strong></td>
<td><strong>17</strong></td>
</tr>
</tbody>
</table>
Kinds of bugs found (1)

- Erroneous insertion into linked list

% vertices with indegree = 0, 1
% vertices with outdegree = 0, 1
% vertices with indegree = outdegree
Kinds of bugs found (2)

- Shared data structure manipulation errors

% vertices with outdegree = 0
% vertices with indegree = 2
% vertices with indegree = outdegree

Delete
Kinds of bugs found (3)

- Data structure invariants

\[
\begin{align*}
\% \text{ vertices with outdegree } &= 1, 2 \\
\% \text{ vertices with indegree } &= 1, 2
\end{align*}
\]
Kinds of bugs found (3)

• Data structure invariants

See the paper for several more examples

% vertices with indegree = outdegree
% vertices with indegree = 1
## Comparison with SWAT

[Chilimbi and Hauswirth, ASPLOS’04]

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>SWAT</th>
<th>HeapMD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Leaks</td>
<td>#FP</td>
</tr>
<tr>
<td>Multimedia</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interactive web-app.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PC-Game (simulation)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Characteristics of bugs found

• **Systemic bugs**: Repeated often enough to affect heap-graph metric

• HeapMD cannot find “one-off” bugs
  – Temporary data structure invariant violations
False positives and negatives

• In our experiments: 0 false positives
  – Metrics are computed for the whole heap rather than per data structure
  – Anomaly detector only looks for range violations, not stability

• **Comes at a cost: False negatives**
  – Bugs with no effect on heap-graph metrics
  – Bugs that affect heap-graph metrics, but do not violate calibrated range
Talk outline

• Motivation
• Stability of the heap
• Application to bug-finding
• Related work and conclusion
Related work

• Tools for special classes of bugs
  – Purify, Valgrind, SWAT, …
  – Not built for detecting invariant violations

• Tools to find invariants
  – Daikon [Ernst], DIDUCE [Hangal and Lam, ICSE 2002]
  – HeapMD complements these in the types of invariants found

• Shape analysis algorithms and tools
  – Can find invariant violations given a correctness specification
Points to take home

Stable heap-graph metrics exist. Their ranges serve as invariants

Range violation $\rightarrow$ Bug likely exercised

Can find non-crashing bugs e.g., invariant violations
Questions?

HeapMD: Identifying Heap-based Bugs using Anomaly Detection

Trishul M. Chilimbi  
Microsoft Research  
Redmond, WA  
trishulc@microsoft.com

Vinod Ganapathy  
University of Wisconsin  
Madison, WI  
vg@cs.wisc.edu