Blended Program Analysis for Improving Reliability of Real-world Applications

Dr. Barbara G. Ryder
J. Byron Maupin Professor of Engineering
Virginia Tech

Collaborators: Bruno Dufour (Rutgers), Gary Sevitsky (IBM Research), Marc Fisher II (VT), Ben Wiedermann (VT), Shiyi Wei (VT), S. Basu (ug-Lafayette), Luke Marrs (ug-VT); Funded by IBM Open Collaborative Research Program and NSF 08-0811518
Outline

- What is program analysis?
- Challenge of modern software applications
- What is blended program analysis?
- Examples of blended analysis
  - Performance diagnosis for framework-based applications (Java)
  - Data integrity & program understanding for webpage codes (JavaScript)
- Summary
What is Program Analysis?

- Historically,
  - **Static program analysis** was used for code optimization during compilation
    - Gave a safe approximation of all possible program behaviors, without running the program
    - Allows checking of specific program properties
  - **Dynamic program analysis** was used for tracking program behavior at runtime
    - Gathered information about program on one execution
What is Program Analysis?

- **Static Analysis**
  - Input: code
  - Output: model of semantics of program
  - When: At compile time
  - Cost: High
  - Goal: code optimization, property validation, program understanding, test case generation...

- **Dynamic Analysis**
  - Input: trace of execution(s)
  - Output: an observed property
  - When? At runtime
  - Cost: Low
  - Goal: Debugging, uncovering code dependences, test coverage...
Challenge: Tools for Modern SW Applications

- Framework-based software systems (Java)
  - Transactional, complex, many libraries/components
    - E.g., personal financial managers, e-commerce applications, information records managers
  - Performance problems difficult to isolate
  - Need understanding of business logic as well as run-time behavior
Dynamic Language Constructs

- **Java**
  - Reflection - values from run-time environment influence program behavior (e.g., in dynamic class loading)
    
    ```java
    //read name of class to be dynamically loaded
    //from command line*
    Class a = Class.forName(args[0]); ...
    Method mainMethod = findMain(a);
    mainMethod.invoke( ... );
    ```

*http://media.techtarget.com/tss/static/articles/content/dm_classForName/DynLoad.pdf*
Challenge: Tools for Modern SW Applications

- Websites (JavaScript)
  - Constructed from combination of static and dynamically loaded/generated code
  - Code can be constructed at runtime and then executed
  - Program understanding (for maintenance), testing, and validating data integrity are hard
Dynamic language constructs

- **JavaScript**
  - Ability to construct a URL at runtime and then load that webpage
  - Functions with variable numbers of arguments
  - Ability to write code at runtime and execute it with `eval` statement

// `xmlhttp` is an instance of `XMLHttpRequest`
`eval(xmlhttp.response());`
What is Blended Program Analysis?

A way to integrate dynamic and static analyses to obtain scalability and precision for specific problems

- Use of a dynamic program representation
  - Reflects call structure of execution(s)
- Use of light-weight dynamic information
  - To prune (some) infeasible paths
  - To tie static analysis information to actual runtime objects
  - To deal with dynamic program constructs
Outline

- Blended analysis for performance diagnosis in framework-intensive applications (Java)
Framework-based Applications

- Application is an iceberg
  - Bulk of the code in libraries and frameworks
  - Genre not commonly addressed by research community
  - E.g., financial planning services, e-commerce sites, online reservation systems, Tomcat-based systems software

- Programs are not just large, but are more complex in interactions between frameworks
- Performance problems span multiple layers
Framework-based Applications

- **Software characteristics**
  - Not amenable to static analyses
    - Not scalable -- too complex
  - Not amenable to dynamic analysis
    - Too intrusive into execution for production codes
  - Application’s main function often is data transformation

- **Goal:** design analyses for performance diagnosis of these systems
Goal: Find Object Churn

- Identify execution contexts with excessive use of temporaries
  - Based on total number of instances
  - Not the same as finding often-executed allocation sites
  - Need to identify temporary objects and to approximate "object lifetime"

- Elimination strategies
  - Optimize the frequent use of frameworks and libraries together
  - Introduce caching for temporary data structures
  - Code specialization
Current Practice: Jinsight Trace of `HoldingDataBean_Ser.serialize()`

Tens of thousands of calls
How to find churn locality?
Blended Analysis - Scalability

Looking at the entire trace

2 orders of magnitude!

Approximating contexts that use temporaries

Identifying contexts that truly use temporaries

getHoldings()
Blended Analysis Paradigm

- Java Application
  - Profile
    - Dynamic Calling Structure
      - Models of methods
        - Static Analysis
          - Reflection Specification + Templates
        - Loaded Classes
Pruning Code in Methods

Entry

\[
x = \text{new } B()
\]

\[
y = D.m()
\]

\[
z = C.m()
\]

\[
w = \text{new } A()
\]

Allocated types: \{B\}

Observed targets: \{D.m\}

Exit
void bar() {
    a = new A();
    a.x = new B();
}

void foo(F f) {
    c = baz();
    f.w = c.z;
}

void zag() {
    F f = new F();
    foo(f);
    G.global = f;
}
Invocation tree for hasConnectAccess

Optimized trace; only shows allocating and capturing methods 1000 instances created but 880 not escaping

Each invoke of hasConnectaccess has a unique SecurityDescriptor for an ID instance; could cache w/I ID;

SecurityServer
hasConnectAccess
180 captured

SecurityDescriptor
getEffectiveAccess
60 captured, 40 new

SecurityDescriptor
getAttribute
20 captured, 20 new

SecurityDescriptor
loadFromBytes
100 captured, 40 new

SecurityContext
isSidPresent
80 captured

GCDObjec
getAttribute
80 captured

LittleEndianUtil
readInt
40 captured, 40 new

LittleEndianUtil
readInt
80 captured, 80 new

LittleEndianUtil
readIntFromBytes
40 captured

LittleEndianUtil
readIntFromBytes
80 captured

SecurityContext
isSidPresent
80 captured

AccessControlList
deserialize
80 new

PermissionSource
getInstanceFromInt
40 captured, 40 new

AccessControlEntry
deserialize
120 new

AccessControlEntry
deserialize
120 new

AceAccess
getInstanceFromInt
40 captured, 40 new

LittleEndianUtil
readShort
40 captured, 20 new

LittleEndianUtil
readShort
80 captured, 40 new

LittleEndianUtil
readShort
80 captured, 40 new

LittleEndianUtil
readShort
80 captured, 40 new

APSEC Keynote, Dec 2012, BG Ryder
Metrics

Designed new metrics for blended escape analysis

- Measure effectiveness of pruning
  - Scalability of analysis – % of blocks in methods pruned
<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Analysis time (h:m:s)</th>
<th>Speedup</th>
<th>%Pruned</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>No pruning</td>
<td>Pruned</td>
<td></td>
</tr>
<tr>
<td>Direct-Std</td>
<td>00:00:18</td>
<td>00:00:17</td>
<td>1.1</td>
</tr>
<tr>
<td>Direct-WS</td>
<td>01:34:01</td>
<td>00:04:41</td>
<td>20.0</td>
</tr>
<tr>
<td>EJB-Std</td>
<td>00:04:24</td>
<td>00:01:46</td>
<td>2.5</td>
</tr>
<tr>
<td>EJB-WS</td>
<td>N/A</td>
<td>29:23:16</td>
<td>N/A</td>
</tr>
<tr>
<td>Eclipse</td>
<td>24:37:12</td>
<td>06:37:22</td>
<td>3.7</td>
</tr>
<tr>
<td>Jazz</td>
<td>02:49:55</td>
<td>00:39:06</td>
<td>4.3</td>
</tr>
<tr>
<td>IBM ApplIn</td>
<td>00:04:35</td>
<td>00:02:05</td>
<td>2.2</td>
</tr>
</tbody>
</table>
Metrics

- Measure usage of temporaries
  - Disposition - categorizes instances as globally: escaping, captured, mixed
Disposition of Instances

Most instances exhibit only 1 behavior

~50% instances captured
Metrics

- **Measure usage of temporaries**
  - *Concentration* - measures locality of temporary usage
Concentration of Captured Instances

% of captured instances

- Direct/Std: 29.0%
- Direct/WS: 54.3%
- EJB/Std: 42.0%
- EJB/WS: 51.2%
- Eclipse: 88.4%
- Jazz: 62.2%
- IBM App: 66.8%
Outline

- Blended analysis for capturing effects of dynamically generated or dynamically loaded code (JavaScript)
JavaScript – Website Glue Code

- JavaScript is *lingua franca* of client-side applications
  - 98 out of 100 most popular websites use JavaScript (Guarnieri et al, ISSTA11)
  - Use of dynamic features is evident in websites (Richards et al, ECOOP11, PLDI10; Zorn et al, WebApps10)
Blended Analysis Paradigm for JavaScript

- Application
  - Profile
    - Dynamic Calling Structure
      - Gather dynamically generated/loaded code
      - Pruned models of methods
        - Static Analysis
How to Profile a Website for Analysis?

- Run several executions of the website, to explore its behaviors
- Each execution is comprised by a set of page traces
- Gather all page traces for the same webpage together and consider them a single JavaScript program for analysis
JavaScript Analysis Framework

Execution collector → Trace selector → Solution integrator

Static analysis

Static analyzer → Solution integrator

Dynamic analysis

JS Program Solution

APSEC Keynote, Dec 2012, BG Ryder
Tainted Input Analysis

- Integrity violation – allowing user input to reach a sensitive operation
  - Tainted Source: user has control of its value
  - Sensitive operation: can affect behavior of website or browser
  - Source-sink pair reported if there is a possible dataflow between them (ignoring sanitizers)
- Hyp: blended tainted input analysis will report fewer false alarms and more true positives than pure static tainted input analysis
Benchmarks 1

<table>
<thead>
<tr>
<th>Website</th>
<th>Page count</th>
<th>Trace count</th>
</tr>
</thead>
<tbody>
<tr>
<td>bing.com</td>
<td>19</td>
<td>30</td>
</tr>
<tr>
<td>twitter.com</td>
<td>14</td>
<td>30</td>
</tr>
<tr>
<td>linkedin.com</td>
<td>12</td>
<td>30</td>
</tr>
<tr>
<td>qq.com</td>
<td>18</td>
<td>30</td>
</tr>
<tr>
<td>wordpress.com</td>
<td>23</td>
<td>30</td>
</tr>
<tr>
<td>sina.com.cn</td>
<td>16</td>
<td>30</td>
</tr>
<tr>
<td>163.com</td>
<td>22</td>
<td>30</td>
</tr>
<tr>
<td>cnn.com</td>
<td>18</td>
<td>30</td>
</tr>
<tr>
<td>msn.com</td>
<td>18</td>
<td>30</td>
</tr>
<tr>
<td>conduit.com</td>
<td>10</td>
<td>16</td>
</tr>
<tr>
<td>imdb.com</td>
<td>10</td>
<td>18</td>
</tr>
<tr>
<td>myspace.com</td>
<td>10</td>
<td>24</td>
</tr>
<tr>
<td>sohu.com</td>
<td>10</td>
<td>19</td>
</tr>
<tr>
<td>xing.com</td>
<td>10</td>
<td>18</td>
</tr>
<tr>
<td>xunlei.com</td>
<td>10</td>
<td>22</td>
</tr>
<tr>
<td>zedo.com</td>
<td>10</td>
<td>16</td>
</tr>
<tr>
<td>washingtonpost.com</td>
<td>10</td>
<td>27</td>
</tr>
<tr>
<td>pconline.com.cn</td>
<td>10</td>
<td>21</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td><strong>14</strong></td>
<td><strong>25</strong></td>
</tr>
</tbody>
</table>
Blended vs Pure Static Analysis

- Pure static solution
- Blended solution
- Part of static solution not found by blended
- Part of solution from dynamically loaded or generated code
- Shaded area is part of solution found by pure static and blended
## Tainted Input Results

<table>
<thead>
<tr>
<th>Website</th>
<th>Pure Static</th>
<th>Pure Static</th>
<th>Blended</th>
<th>Blended</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>true soln</td>
<td>false alarm</td>
<td>true soln</td>
<td>false alarm</td>
</tr>
<tr>
<td>live.com</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>youtube.com</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>myspace.com</td>
<td>1</td>
<td>1</td>
<td>false negatives</td>
<td></td>
</tr>
<tr>
<td>sohu.com</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>xunlei.com</td>
<td>3</td>
<td>false positives</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>msn.com</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bing.com</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>totals</td>
<td>6</td>
<td>2</td>
<td>9</td>
<td></td>
</tr>
</tbody>
</table>

*false positives*

*false negatives*
Statement-level Side-effects (ST-MOD) For Program Understanding

- Want to know the number of objects whose \( f \) field value may be changed at statement: \( x.f = \)
  - Helpful in navigating unfamiliar code
- How solve?
  - Find which objects \( o \) that \( x \) can point to
  - Find which objects \( o.f \) can point to
### Benchmarks2

<table>
<thead>
<tr>
<th>Website</th>
<th>Page count</th>
<th>Trace count</th>
<th>eval page</th>
<th>variadic function</th>
</tr>
</thead>
<tbody>
<tr>
<td>google.com</td>
<td>203</td>
<td>2104</td>
<td>52</td>
<td>177</td>
</tr>
<tr>
<td>facebook.com</td>
<td>138</td>
<td>1098</td>
<td>23</td>
<td>65</td>
</tr>
<tr>
<td>youtube.com</td>
<td>122</td>
<td>579</td>
<td>19</td>
<td>29</td>
</tr>
<tr>
<td>yahoo.com</td>
<td>52</td>
<td>265</td>
<td>21</td>
<td>13</td>
</tr>
<tr>
<td>baidu.com</td>
<td>49</td>
<td>147</td>
<td>6</td>
<td>16</td>
</tr>
<tr>
<td>wikipedia.org</td>
<td>67</td>
<td>130</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>live.com</td>
<td>54</td>
<td>226</td>
<td>10</td>
<td>44</td>
</tr>
<tr>
<td>blogger.com</td>
<td>24</td>
<td>146</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td><strong>totals</strong></td>
<td><strong>709</strong></td>
<td><strong>4695</strong></td>
<td><strong>137</strong></td>
<td><strong>354</strong></td>
</tr>
</tbody>
</table>
## Comparison: Pure Static to Blended Solutions

(points-to)

<table>
<thead>
<tr>
<th>Website</th>
<th>% Coverage of pure static solution</th>
<th>% Additional results</th>
</tr>
</thead>
<tbody>
<tr>
<td>google.com</td>
<td>89.7</td>
<td>5.9</td>
</tr>
<tr>
<td>facebook.com</td>
<td>85.3</td>
<td>7.5</td>
</tr>
<tr>
<td>youtube.com</td>
<td>89.1</td>
<td>9.9</td>
</tr>
<tr>
<td>yahoo.com</td>
<td>78</td>
<td>9.8</td>
</tr>
<tr>
<td>baidu.com</td>
<td>93</td>
<td>6.7</td>
</tr>
<tr>
<td>wikipedia.org</td>
<td>92.1</td>
<td>none</td>
</tr>
<tr>
<td>live.com</td>
<td>81.8</td>
<td>7.5</td>
</tr>
<tr>
<td>blogger.com</td>
<td>83.8</td>
<td>1.4</td>
</tr>
<tr>
<td><strong>mean</strong></td>
<td><strong>86.6</strong></td>
<td><strong>7.0</strong></td>
</tr>
</tbody>
</table>

APSEC Keynote, Dec 2012, BG Ryder
<table>
<thead>
<tr>
<th>Websites</th>
<th>Pure Static Average number of objects in static code</th>
<th>Blended Average number of objects in static code</th>
<th>Average number of objects in dynamic code</th>
</tr>
</thead>
<tbody>
<tr>
<td>google.com</td>
<td>5.8</td>
<td>2.4</td>
<td>2.1</td>
</tr>
<tr>
<td>facebook.com</td>
<td>7.7</td>
<td>4.1</td>
<td>3.6</td>
</tr>
<tr>
<td>youtube.com</td>
<td>5.9</td>
<td>3.5</td>
<td>2.4</td>
</tr>
<tr>
<td>yahoo.com</td>
<td>5.2</td>
<td>2.5</td>
<td>2.8</td>
</tr>
<tr>
<td>baidu.com</td>
<td>2.6</td>
<td>1.4</td>
<td>1.8</td>
</tr>
<tr>
<td>live.com</td>
<td>2.9</td>
<td>1.6</td>
<td>2.2</td>
</tr>
<tr>
<td>blogger.com</td>
<td>4.5</td>
<td>2.8</td>
<td>2.3</td>
</tr>
<tr>
<td>average</td>
<td>4.9</td>
<td>2.6</td>
<td>2.5</td>
</tr>
</tbody>
</table>
Summary

- Modern SW applications require new approaches to program analysis - the engine behind SW tools
- New blended analysis paradigm seems promising for handling the more dynamic programming constructs
  - Performance diagnosis of framework-based apps
  - Understanding of website codes for enhancement
- More experimentation and investigation needed to select best analysis for specific use
Challenges

- Expect more combinations of static and dynamic analyses for web & mobile apps
- Challenge: analyzing 3rd party apps (executables) in combo with known codes
- Challenge: analyzing event-driven codes
- Challenge: increased use of explicit concurrency will require new tools and supporting analyses
Thank You

Questions?