Unit Test Virtualization: Optimizing Testing Time

Jonathan Bell and Gail Kaiser
Computer Science Department, Columbia University

The Problem: Isolating Test Cases

Developers can accidentally create code that makes testing difficult:

```java
/**
 * If true, cookie values are allowed to contain an equals character
 * without being quoted.
 */
public static final boolean ALLOW_EQUALS_IN_VALUE =
    Boolean.valueOf(System.getProperty("org.apache.tomcat.
        util.http.ServerCookie.ALLOW_EQUALS_IN_VALUE","false"))
    .booleanValue();
```

Code sample from Apache Tomcat that demonstrates the sort of code that can create unexpected test case dependencies: ALLOW_EQUALS_IN_VALUE can be set only once: on subsequence executions within the same process, its value will not change, even if the system property does. This sort of dependency is non-trivial to detect (in fact, NP-complete).

This fix is very commonly used in large Java projects, and is very slow. We mined the top 1,000 Java projects on ohloh. We looked at those using ant or maven to run automated tests to see how many isolate test cases in separate processes. For 20 of these, we calculated the overhead of isolating each test (shown in bottom table).

<table>
<thead>
<tr>
<th>Number of tests in project</th>
<th>Number of projects creating a new process for each test</th>
<th>Number of lines of code in project</th>
<th>Number of projects creating a new process for each test</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-10</td>
<td>24/71 (34%)</td>
<td>0-10k</td>
<td>7/42 (17%)</td>
</tr>
<tr>
<td>10-100</td>
<td>81/235 (34%)</td>
<td>10k-100k</td>
<td>60/200 (30%)</td>
</tr>
<tr>
<td>100-1000</td>
<td>97/239 (41%)</td>
<td>100k-1m</td>
<td>114/267 (43%)</td>
</tr>
<tr>
<td>&gt;=1000</td>
<td>38/47 (81%)</td>
<td>&gt;1m</td>
<td>58/82 (71%)</td>
</tr>
<tr>
<td>(Overall)</td>
<td>240/591 (41%)</td>
<td>(Overall)</td>
<td>240/591 (41%)</td>
</tr>
</tbody>
</table>

Efficiently reset Java applications to their starting state

Assuming that classes are not reused between test executions (by the test runner), only possible leakage is through static fields. The graph below shows how such a leakage could occur.

```
VMVM is much faster than running each test in its own process
We compared VMVM's overhead to that of traditional, process-based isolation, finding it significantly reduced test execution time.

```

VMVM uses a hybrid static-dynamic analysis
VMVM efficiently resets these static fields on-demand using a two-phase static/dynamic byte code analysis. Statically, VMVM identifies classes that may possibly need to be reset and inserts guards. At runtime, these guards are checked.

Applications to non-Java languages
VMVM targets JVM, but is tightly integrated with JUnit. We are currently integrating VMVM with the Scala compiler’s partest. The Scala compiler test suite contains over 3,500 test cases, each executed in their own process.

VMVM is on GitHub: http://github.com/Programming-Systems-Lab/vmvm