Probabilistic Calling Context

Michael D. Bond       Kathryn S. McKinley
University of Texas at Austin

Why Context Sensitivity?

- Static program location not enough

```
at com.mckoi.db.jdbcserver.JDBCInterface.executeQuery():213
```
Why Context Sensitivity?

- Static program location not enough

at com.mckoi.db.jdbcserver.JDBCInterface.executeQuery():213
at com.mckoi.db.jdbc.MConnection.executeQuery():348
at com.mckoi.db.jdbc.MStatement.executeQuery():110
at com.mckoi.db.jdbc.MStatement.executeQuery():127
at Test.main():48

- Motivated by
  - Complex programs
  - Small methods
  - Virtual dispatch
Why Context Sensitivity?

- **Static program location not enough**

  ```
  at com.mckoi.db.jdbcserver.JDBCInterface.executeQuery():213
  at com.mckoi.db.jdbc.MConnection.executeQuery():348
  at com.mckoi.db.jdbc.MStatement.executeQuery():110
  at com.mckoi.db.jdbc.MStatement.executeQuery():127
  at Test.main():48
  ```

- **Motivated by**
  - Complex programs
  - Small methods
  - Virtual dispatch

Context Is Nontrivial

<table>
<thead>
<tr>
<th>Program</th>
<th>API calls</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Call sites</td>
<td>Distinct contexts</td>
<td></td>
</tr>
<tr>
<td>antlr</td>
<td>4,184</td>
<td>128,627</td>
<td></td>
</tr>
<tr>
<td>bloat</td>
<td>3,306</td>
<td>600,947</td>
<td></td>
</tr>
<tr>
<td>chart</td>
<td>2,335</td>
<td>202,603</td>
<td></td>
</tr>
<tr>
<td>eclipse</td>
<td>9,611</td>
<td>226,020</td>
<td></td>
</tr>
<tr>
<td>fop</td>
<td>2,225</td>
<td>37,710</td>
<td></td>
</tr>
<tr>
<td>hsqldb</td>
<td>947</td>
<td>16,050</td>
<td></td>
</tr>
<tr>
<td>jython</td>
<td>1,830</td>
<td>628,048</td>
<td></td>
</tr>
<tr>
<td>luindex</td>
<td>654</td>
<td>102,556</td>
<td></td>
</tr>
<tr>
<td>lusearch</td>
<td>507</td>
<td>905</td>
<td></td>
</tr>
<tr>
<td>pmd</td>
<td>1,890</td>
<td>847,108</td>
<td></td>
</tr>
<tr>
<td>xalan</td>
<td>1,530</td>
<td>17,905</td>
<td></td>
</tr>
</tbody>
</table>
Example: Residual Testing

Does behavior occur at production time that did not occur at testing time?

class SimpleWindow {
    close() {
        ... 
    }
}

class EditorWindow {
    close() {
        ... 
    }
}

Example: Residual Testing

Does behavior occur at production time that did not occur at testing time?

inputHandler() {
    ...
    case CLICK_EXIT:
        w.checkUnsaved();
        w.close();
        ...
    }

autoUpdate() {
    ...
    for all windows w
        w.close();
    ...
}
Example: Residual Testing

Does behavior occur at production time that did not occur at testing time?

```
inputHandler() {
    ...
    case CLICK_EXIT:
        w.checkUnsaved();
        w.close();
    ...
}
```

```
autoUpdate() {
    ...
    for all windows w
        w.close();
    ...
}
```

```
class EditorWindow {
    close() {
        ...
    }
}
```

```
class SimpleWindow {
    close() {
        ...
    }
}
```

```
Bug!
```

New behavior indicates bugs

Context sensitivity helps find new behavior
Two-Phase Dynamic Analyses

Training → Production

Behavior observed → New or anomalous behavior detected

Residual testing
What behavior occurs at production time that did not occur at testing time? [Vaswani et al. '07]

Anomaly-based bug detection
What new behavior occurs during a buggy program run? [Hangal & Lam '02]

Anomaly-based intrusion detection
Does a program exhibit anomalous behavior? [Inoue '05]
Probabilistic Calling Context

- Adds context sensitivity to dynamic analyses
- Maintains value representing context
  - Unique with high probability
  - New value → new context → walk stack
- High accuracy: <0.1% false negatives
- Low overhead: 3% overhead, 0-8% for clients

Outline

- Introduction
- Previous approaches
- Maintaining the PCC value
- Evaluation
  - Accuracy
  - Performance
Previous Approaches

- **Tracking context** [Ammons et al. ’97] [Spivey ‘04]
  - Maintain CCT position at each call/return
- **Walking the stack** [Nethercote & Seward ‘07]
- **Path profiling** [Ball & Larus ’96] [Melski & Reps ’99]
  - Call graphs large → path explosion
  - Virtual dispatch complicates instrumentation

---

Previous Approaches

- **Tracking context** [Ammons et al. ’97] [Spivey ‘04]
  - Maintain CCT position at each call/return
- **Walking the stack** [Nethercote & Seward ‘07]
- **Path profiling** [Ball & Larus ’96] [Melski & Reps ’99]
  - Call graphs large → path explosion
  - Virtual dispatch complicates instrumentation

- **Sampling** [Zhuang et al. ’06]
  - Sacrifices coverage for low overhead
Outline

- Introduction
- Previous approaches
- Maintaining the PCC value
- Evaluation
  - Accuracy
  - Performance

PCC Function

\[ f ( V , cs ) \]

- \( V \) is PCC value
- \( cs \) is call site ID
PCC Function

\[ f( V, cs ) \equiv 3V + cs \pmod{2^{32}} \]

- \( V \) is PCC value
- \( cs \) is call site ID
**PCC Function**

\[ f( V, cs ) \equiv 3V + cs \pmod{2^{32}} \]

- Motivated by MPI datatype hashing
  [Langou et al. '05] [Gropp '00]
- Cheap to compute
- Desirable properties:
  - Non-commutative
  - Composition efficient to compute

**Differentiating Similar Contexts**

- \( V \leftarrow 3V + cs_1 \)
- \( B \leftarrow 3V + cs_2 \)
- \( V \leftarrow 3V + cs_2 \)
- \( A() \rightarrow B() \rightarrow ... \)
- \( B() \rightarrow A() \rightarrow ... \)
Differentiating Similar Contexts

- Non-commutative

\[ f(f(V, cs_1), cs_2) \neq f(f(V, cs_2), cs_1) \]

Efficiency at Inlined Calls
Efficiency at Inlined Calls

- Composition efficient to compute
  \[ f^n(V, cs_i) = 3^n V + \sum_i 3^i cs_i \]

Outline

- Introduction
- Previous approaches
- Maintaining the PCC value
- Evaluation
  - Methodology
  - Evaluating potential clients
  - Accuracy
  - Performance
Methodology

- Implementation in Jikes RVM 2.4.6
  - Available on Jikes RVM Research Archive
- Deterministic calling context profiling
  - Maintains CCT node at each call & return
- Benchmarks: DaCapo, SPEC JBB2000, SPEC JVM98
- Platform: 3.6 GHz Pentium 4 w/Linux

How Clients Use PCC

Record values  \rightarrow\text{New value} \rightarrow \text{new context} \rightarrow \text{walk stack}

Training  \rightarrow  Production

Behavior observed  \rightarrow  New or anomalous behavior detected
Evaluating Potential Clients

Memory overhead: proportional to contexts
Evaluating Potential Clients

- Anomaly-based intrusion detection
  Check PCC value at system calls (Network, I/O, OS)

- Residual testing
  Check PCC value at Java API calls (calls to `java.*`)

- Upper bound
  Check PCC value at all calls

Ideal Accuracy

- PCC maps context to value
  - New PCC value → new context
  - Familiar PCC value → probably familiar context
Ideal Accuracy

- PCC maps context to value
  - New PCC value → new context
  - Familiar PCC value → probably familiar context

<table>
<thead>
<tr>
<th>Distinct contexts</th>
<th>32-bit values</th>
<th>64-bit values</th>
</tr>
</thead>
<tbody>
<tr>
<td>100,000</td>
<td>1 (0.0%)</td>
<td>0 (0.0%)</td>
</tr>
<tr>
<td>1,000,000</td>
<td>116 (0.0%)</td>
<td>0 (0.0%)</td>
</tr>
<tr>
<td>10,000,000</td>
<td>11,632 (0.1%)</td>
<td>0 (0.0%)</td>
</tr>
<tr>
<td>100,000,000</td>
<td>1,155,170 (1.2%)</td>
<td>0 (0.0%)</td>
</tr>
<tr>
<td>1,000,000,000</td>
<td>107,882,641 (10.8%)</td>
<td>0 (0.0%)</td>
</tr>
<tr>
<td>10,000,000,000</td>
<td>6,123,623,065 (61.2%)</td>
<td>3 (0.0%)</td>
</tr>
</tbody>
</table>
### Ideal Accuracy

- PCC maps context to value
  - New PCC value → new context
  - Familiar PCC value → probably familiar context

<table>
<thead>
<tr>
<th>Distinct contexts</th>
<th>Expected conflicts (false negatives)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>32-bit values</td>
</tr>
<tr>
<td>100,000</td>
<td>1 (0.0%)</td>
</tr>
<tr>
<td>1,000,000</td>
<td>116 (0.0%)</td>
</tr>
<tr>
<td>10,000,000</td>
<td>11,632 (0.1%)</td>
</tr>
<tr>
<td>100,000,000</td>
<td>1,155,170 (1.2%)</td>
</tr>
<tr>
<td>1,000,000,000</td>
<td>107,882,641 (10.8%)</td>
</tr>
<tr>
<td>10,000,000,000</td>
<td>6,123,623,065 (61.2%)</td>
</tr>
</tbody>
</table>

### Ideal Accuracy

- PCC maps context to value
  - New PCC value → new context
  - Familiar PCC value → probably familiar context

<table>
<thead>
<tr>
<th>Distinct contexts</th>
<th>Expected conflicts (false negatives)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>32-bit values</td>
</tr>
<tr>
<td>100,000</td>
<td>1 (0.0%)</td>
</tr>
<tr>
<td>1,000,000</td>
<td>116 (0.0%)</td>
</tr>
<tr>
<td>10,000,000</td>
<td>11,632 (0.1%)</td>
</tr>
<tr>
<td>100,000,000</td>
<td>1,155,170 (1.2%)</td>
</tr>
<tr>
<td>1,000,000,000</td>
<td>107,882,641 (10.8%)</td>
</tr>
<tr>
<td>10,000,000,000</td>
<td>6,123,623,065 (61.2%)</td>
</tr>
</tbody>
</table>
### PCC’s Accuracy

<table>
<thead>
<tr>
<th>Program</th>
<th>System calls</th>
<th>Java API calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>antlr</td>
<td>211,490</td>
<td>1,567</td>
</tr>
<tr>
<td>bloat</td>
<td>12</td>
<td>10</td>
</tr>
<tr>
<td>chart</td>
<td>63</td>
<td>62</td>
</tr>
<tr>
<td>eclipse</td>
<td>14,110</td>
<td>197</td>
</tr>
<tr>
<td>fop</td>
<td>18</td>
<td>17</td>
</tr>
<tr>
<td>hsqldb</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>jython</td>
<td>5,929</td>
<td>4,289</td>
</tr>
<tr>
<td>luindex</td>
<td>2,615</td>
<td>14</td>
</tr>
<tr>
<td>lusearch</td>
<td>141</td>
<td>11</td>
</tr>
<tr>
<td>pmd</td>
<td>1,045</td>
<td>25</td>
</tr>
<tr>
<td>xalan</td>
<td>137,895</td>
<td>59</td>
</tr>
</tbody>
</table>
PCC’s Accuracy

<table>
<thead>
<tr>
<th>Program</th>
<th>All calls</th>
<th>Dynamic</th>
<th>Distinct</th>
<th>Conf.</th>
</tr>
</thead>
<tbody>
<tr>
<td>antlr</td>
<td>490,363,211</td>
<td>1,006,578</td>
<td>118</td>
<td></td>
</tr>
<tr>
<td>bloat</td>
<td>6,276,446,059</td>
<td>1,980,205</td>
<td>453</td>
<td></td>
</tr>
<tr>
<td>chart</td>
<td>908,459,469</td>
<td>845,432</td>
<td>91</td>
<td></td>
</tr>
<tr>
<td>eclipse</td>
<td>1,266,810,504</td>
<td>4,815,901</td>
<td>2,652</td>
<td></td>
</tr>
<tr>
<td>fop</td>
<td>44,200,446</td>
<td>174,955</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>hsqldb</td>
<td>877,680,667</td>
<td>110,795</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>jython</td>
<td>5,326,949,158</td>
<td>3,859,545</td>
<td>1,738</td>
<td></td>
</tr>
<tr>
<td>luindex</td>
<td>740,053,104</td>
<td>374,201</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>lusearch</td>
<td>1,439,034,336</td>
<td>6,039</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>pmd</td>
<td>2,726,876,957</td>
<td>8,043,096</td>
<td>7,653</td>
<td></td>
</tr>
<tr>
<td>xalan</td>
<td>10,083,858,546</td>
<td>163,205</td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>

PCC’s Execution Time Overhead

![Chart showing the execution time overhead for PCC and DaCapo, SPEC, and All categories. The overhead values range from 0 to 60%. The data shows that the overhead is less than 3% for DaCapo and SPEC, with a peak of 7.653% for All.]
### Summary

- **PCC maintains calling context value**
  - New value indicates new behavior
- **Low overhead**
  - Maintaining PCC value adds 3%
  - Checking PCC value 0-8%
  - Memory overhead proportional to contexts
- **High accuracy**
  - Less than 0.1% false negative rate
- **PCC adds context sensitivity to clients that detect anomalous behavior**
Summary

- PCC maintains calling context value
  - New value indicates new behavior
- Low overhead
  - Maintaining PCC value adds 3%
  - Checking PCC value 0-8%
  - Memory overhead proportional to contexts
- High accuracy
  - Less than 0.1% false negative rate
- PCC adds context sensitivity to clients that detect anomalous behavior

Extra slides
Context Sensitivity Mostly Unused

- Do paths capture enough behavior?

```
C/Fortran method
```

```
Java/C# method
```