Efficient Detection of Thread Safety Violations via Coverage-guided Generation of Concurrent Unit Tests

Ankit Choudhary  Shan Lu  Michael Pradel
Thread Safety

“A class is thread-safe if it behaves correctly when accessed from multiple threads, regardless of the scheduling or interleaving of the execution of those threads by the runtime environment, and with no additional synchronization or coordination on the part of the calling code.”

- Java Concurrency in Practice
Thread Safety Bug - Example

```java
public class IntegerList {
    protected int array[] = ...;
    protected int index = 0;
    public synchronized void add(int num) {
        if(array != null) {
            if(index == array.length) {
                resize();
            }
        }
    }
    public void close() {
        array = null;
    }
}
```
public class IntegerList {
    protected int array[] = ...;
    protected int index = 0;
    public synchronized void add(int num) {
        if (array != null) {
            if (index == array.length) {
                resize();
            }
        }
    }
    public void close() {
        array = null;
    }
}
public class IntegerList {
    protected int array[] = ...;
    protected int index = 0;
    public synchronized void add(int num) {
        if(array != null) {
            if(index == array.length) {
                resize();
            }
        }
    }
    public void close() {
        array = null;
    }
}
public class IntegerList {
    protected int array[] = ..;
    protected int index = 0;
    public synchronized void add(int num) {
        if(array != null) {
            if(index == array.length) {
                resize();
            }
        }
    }
    public void close() {
        array = null;
    }
}
public class IntegerList {
    protected int array[] = ...;
    protected int index = 0;
    public synchronized void add(int num) {
        if(array != null) {
            if(index == array.length) {
                resize();
            }
        }
    }
    public void close() {
        array = null;
    }
}
Finding Concurrency Bugs

- Generation of Multithreaded Tests
- Exploration of Low-level Interleavings
Finding Concurrency Bugs

- Generation of Multithreaded Tests
- Exploration of Low-level Interleavings
Finding Concurrency Bugs

- Generation of Multithreaded Tests
- Exploration of Low-level Interleavings
- Random
- Analysis-based
Finding Concurrency Bugs

Generation of Multithreaded Tests

Random

Exploration of Low-level Interleavings

Analysis-based

+ Simple and inexpensive.
- Repeatedly generates same test.
- Does not consider locks / synchronization (use static analysis).
Finding Concurrency Bugs

**Generation of Multithreaded Tests**

- Random
  + Simple and inexpensive.
  - Repeatedly generates same test.
  - Does not consider locks / synchronization (use static analysis).

**Exploration of Low-level Interleavings**

- Analysis-based
  + Tests directed towards finding bug.
  - Costly (time).
  - Focuses on a particular type of bug (race conditions, atomicity violations, or deadlocks).
This Talk

Generation of Multithreaded Tests

- Simple and inexpensive.
- Repeatedly generates same test.
- Does not consider locks / synchronization (use static analysis). Dynamically assigns lower priority to methods with locks / synchronization.

Our Approach

- Tests directed towards finding bug not yet generated ones.
- Costly (time).
- Focuses on a particular type of bug (race conditions, atomicity violations, or deadlocks) that can lead to exception or deadlock.

Exploration of Low-level Interleavings

Random

Analysis-based
This Talk

Generation of Multithreaded Tests

- Random
- Our Approach
- Analysis-based

Exploration of Low-level Interleavings

- Simple and inexpensive.
- Tests directed towards finding bug not yet generated ones.
- Tests directed towards finding bug type of bug (race conditions, atomicity violations, or deadlocks) that can lead to exception or deadlock.
- Focuses on a particular type of bug (race conditions, atomicity violations, or deadlocks) that can lead to exception or deadlock.
- Repeatedly generates same test.
- Does not consider locks / synchronization.
- Dynamically assigns lower priority to methods with locks / synchronization.

Best of Both Worlds!
A Concurrent Test
A Concurrent Test

IntegerList il = new IntegerList();
A Concurrent Test

```java
IntegerList il = new IntegerList();
```

Thread 1

Thread 2
A Concurrent Test

IntegerList il = new IntegerList();

Thread 1
il.add(5);
il.close();

Thread 2
il.close();
il.add(3);
A Concurrent Test

IntegerList il = new IntegerList();

Thread 1
il.add(5);
il.close();

Thread 2
il.close();
il.add(3);
IntegerList il = new IntegerList();

Thread 1

il.add(5);
il.close();

Thread 2

il.add(3);
il.close();

How do we select methods to test in suffixes?
CovCon - Overview

Class Under Test

Concurrent Method Pairs

Prioritizer

Coverage Detector

Method-pair

Traces

Test Generator

Test Executor

Test Validator

Success

False Positive

Exception / Deadlock

Test

True Positive

Thread Safety Bug
Concurrent Method Pairs

- Set of all pairs of public methods in a class and its super-class.

```java
public class IntegerList {
    public synchronized void add(int num) { }
    public void close() { }
    public synchronized int getIndex(int num) { }
}
```

<table>
<thead>
<tr>
<th>Method 1</th>
<th>Method 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>add</td>
</tr>
<tr>
<td>add</td>
<td>close</td>
</tr>
<tr>
<td>close</td>
<td>close</td>
</tr>
<tr>
<td>getIndex</td>
<td>getIndex</td>
</tr>
<tr>
<td>add</td>
<td>getIndex</td>
</tr>
<tr>
<td>close</td>
<td>getIndex</td>
</tr>
</tbody>
</table>
Test Generator

- Generates test using the selected method pair.

```java
IntegerList il = new IntegerList();
il.add(5);
il.close();
```

<table>
<thead>
<tr>
<th>Method 1</th>
<th>Method 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>add</td>
</tr>
<tr>
<td>add</td>
<td>close</td>
</tr>
<tr>
<td>close</td>
<td>close</td>
</tr>
<tr>
<td>getIndex</td>
<td>getIndex</td>
</tr>
<tr>
<td>add</td>
<td>getIndex</td>
</tr>
<tr>
<td>close</td>
<td>getIndex</td>
</tr>
</tbody>
</table>
Coverage Detector

- Analyze trace files generated in Test Executor.

<table>
<thead>
<tr>
<th>Method 1</th>
<th>Method 2</th>
<th>Covered Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>add</td>
<td>0</td>
</tr>
<tr>
<td>add</td>
<td>close</td>
<td>0</td>
</tr>
<tr>
<td>close</td>
<td>close</td>
<td>0</td>
</tr>
<tr>
<td>getIndex</td>
<td>getIndex</td>
<td>0</td>
</tr>
<tr>
<td>add</td>
<td>getIndex</td>
<td>0</td>
</tr>
<tr>
<td>close</td>
<td>getIndex</td>
<td>0</td>
</tr>
</tbody>
</table>

Trace File 1 | Trace File 2
Coverage Detector

- Analyze trace files generated in Test Executor.

<table>
<thead>
<tr>
<th></th>
<th>Method 1</th>
<th>Method 2</th>
<th>Covered Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>add</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>add</td>
<td>close</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>close</td>
<td>close</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>getIndex</td>
<td>getIndex</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>add</td>
<td>getIndex</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>close</td>
<td>getIndex</td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

Trace File 1
Start:add    Time:1

Trace File 2
## Coverage Detector

- Analyze trace files generated in Test Executor.

<table>
<thead>
<tr>
<th>Method 1</th>
<th>Method 2</th>
<th>Covered Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>add</td>
<td>0</td>
</tr>
<tr>
<td>add</td>
<td>close</td>
<td>0</td>
</tr>
<tr>
<td>close</td>
<td>close</td>
<td>0</td>
</tr>
<tr>
<td>getIndex</td>
<td>getIndex</td>
<td>0</td>
</tr>
<tr>
<td>add</td>
<td>getIndex</td>
<td>0</td>
</tr>
<tr>
<td>close</td>
<td>getIndex</td>
<td>0</td>
</tr>
</tbody>
</table>

**Trace File 1**

Start: add    Time: 1

**Trace File 2**

Start: close  Time: 2
Coverage Detector

- Analyze trace files generated in Test Executor.

<table>
<thead>
<tr>
<th>Method 1</th>
<th>Method 2</th>
<th>Covered Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>add</td>
<td>0</td>
</tr>
<tr>
<td>add</td>
<td>close</td>
<td>1</td>
</tr>
<tr>
<td>close</td>
<td>close</td>
<td>0</td>
</tr>
<tr>
<td>getIndex</td>
<td>getIndex</td>
<td>0</td>
</tr>
<tr>
<td>add</td>
<td>getIndex</td>
<td>0</td>
</tr>
<tr>
<td>close</td>
<td>getIndex</td>
<td>0</td>
</tr>
</tbody>
</table>

Trace File 1
Start:add Time:1
End:add Time:3

Trace File 2
Start:close Time:2
Coverage Detector

- Analyze trace files generated in Test Executor.

<table>
<thead>
<tr>
<th>Method 1</th>
<th>Method 2</th>
<th>Covered Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>add</td>
<td>0</td>
</tr>
<tr>
<td>add</td>
<td>close</td>
<td>2</td>
</tr>
<tr>
<td>close</td>
<td>close</td>
<td>1</td>
</tr>
<tr>
<td>getIndex</td>
<td>getIndex</td>
<td>0</td>
</tr>
<tr>
<td>add</td>
<td>getIndex</td>
<td>0</td>
</tr>
<tr>
<td>close</td>
<td>getIndex</td>
<td>0</td>
</tr>
</tbody>
</table>

Trace File 1

- Start: add Time: 1
- End: add Time: 3
- Start: close Time: 4
- End: close Time: 8

Trace File 2

- Start: close Time: 2
- End: close Time: 5
- Start: add Time: 6
- End: add Time: 7
Prioritizer
Prioritizer

- **Tried Count** ($T$): Number of times a method-pair appears in concurrent suffixes
Prioritizer

- **Tried Count** ($T$): Number of times a method-pair appears in concurrent suffixes

- **Covered Count** ($C$): Number of times a method-pair is executed concurrently
**Prioritizer**

- **Tried Count ($T$):** Number of times a method-pair appears in concurrent suffixes

- **Covered Count ($C$):** Number of times a method-pair is executed concurrently

- **Coverage Score ($S$):** Lower score means higher priority
Prioritizer

- **Tried Count ($T$):** Number of times a method-pair appears in concurrent suffixes

- **Covered Count ($C$):** Number of times a method-pair is executed concurrently

- **Coverage Score ($S$):** Lower score means higher priority

$$S = \max(\text{abs}(T - C), 1) \times \max(T, 1)$$
A Few Executions Later ...

Lower Coverage Score = Higher Priority

<table>
<thead>
<tr>
<th>Method 1</th>
<th>Method 2</th>
<th>Tried Count (T)</th>
<th>Covered Count (C)</th>
<th>Coverage Score (S)</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>add</td>
<td>6</td>
<td>0</td>
<td>36</td>
</tr>
<tr>
<td>add</td>
<td>close</td>
<td>13</td>
<td>11</td>
<td>26</td>
</tr>
<tr>
<td>close</td>
<td>close</td>
<td>8</td>
<td>4</td>
<td>32</td>
</tr>
<tr>
<td>getIndex</td>
<td>getIndex</td>
<td>6</td>
<td>0</td>
<td>36</td>
</tr>
<tr>
<td>add</td>
<td>getIndex</td>
<td>6</td>
<td>0</td>
<td>36</td>
</tr>
<tr>
<td>close</td>
<td>getIndex</td>
<td>14</td>
<td>12</td>
<td>28</td>
</tr>
</tbody>
</table>

\[ S = \text{max}(\text{abs}(T - C), 1) \times \text{max}(T, 1) \]
A Few Executions Later ...

<table>
<thead>
<tr>
<th>Method 1</th>
<th>Method 2</th>
<th>Tried Count (T)</th>
<th>Covered Count (C)</th>
<th>Coverage Score (S)</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>add</td>
<td>6</td>
<td>0</td>
<td>36</td>
</tr>
<tr>
<td>add</td>
<td>close</td>
<td>13</td>
<td>11</td>
<td>26</td>
</tr>
<tr>
<td>close</td>
<td>close</td>
<td>8</td>
<td>4</td>
<td>32</td>
</tr>
<tr>
<td>getIndex</td>
<td>getIndex</td>
<td>6</td>
<td>0</td>
<td>36</td>
</tr>
<tr>
<td>add</td>
<td>getIndex</td>
<td>6</td>
<td>0</td>
<td>36</td>
</tr>
<tr>
<td>close</td>
<td>getIndex</td>
<td>14</td>
<td>12</td>
<td>28</td>
</tr>
</tbody>
</table>

\[ S = \max(\text{abs}(T - C), 1) \times \max(T, 1) \]
A Few Executions Later ...

Maybe protected by locks / synchronization

<table>
<thead>
<tr>
<th>Method 1</th>
<th>Method 2</th>
<th>Tried Count (T)</th>
<th>Covered Count (C)</th>
<th>Coverage Score (S)</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>add</td>
<td>6</td>
<td>0</td>
<td>36</td>
</tr>
<tr>
<td>add</td>
<td>close</td>
<td>13</td>
<td>11</td>
<td>26</td>
</tr>
<tr>
<td>close</td>
<td>close</td>
<td>8</td>
<td>4</td>
<td>32</td>
</tr>
<tr>
<td>getIndex</td>
<td>getIndex</td>
<td>6</td>
<td>0</td>
<td>36</td>
</tr>
<tr>
<td>add</td>
<td>getIndex</td>
<td>6</td>
<td>0</td>
<td>36</td>
</tr>
<tr>
<td>close</td>
<td>getIndex</td>
<td>14</td>
<td>12</td>
<td>28</td>
</tr>
</tbody>
</table>

\[ S = \max(\text{abs}(T - C), 1) \times \max(T, 1) \]
A Few Executions Later ...

<table>
<thead>
<tr>
<th>Method 1</th>
<th>Method 2</th>
<th>Tried Count (T)</th>
<th>Covered Count (C)</th>
<th>Coverage Score (S)</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>add</td>
<td>6</td>
<td>0</td>
<td>36</td>
</tr>
<tr>
<td>add</td>
<td>close</td>
<td>13</td>
<td>11</td>
<td>26</td>
</tr>
<tr>
<td>close</td>
<td>close</td>
<td>8</td>
<td>4</td>
<td>32</td>
</tr>
<tr>
<td>getIndex</td>
<td>getIndex</td>
<td>6</td>
<td>0</td>
<td>36</td>
</tr>
<tr>
<td>add</td>
<td>getIndex</td>
<td>6</td>
<td>0</td>
<td>36</td>
</tr>
<tr>
<td>close</td>
<td>getIndex</td>
<td>14</td>
<td>12</td>
<td>28</td>
</tr>
</tbody>
</table>

\[ S = \max(\text{abs}(T - C), 1) \times \max(T, 1) \]
A Few Executions Later ...

Select `add` and `close`

<table>
<thead>
<tr>
<th>Method 1</th>
<th>Method 2</th>
<th>Tried Count ($T$)</th>
<th>Covered Count ($C$)</th>
<th>Coverage Score ($S$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>add</td>
<td>6</td>
<td>0</td>
<td>36</td>
</tr>
<tr>
<td>add</td>
<td>close</td>
<td>13</td>
<td>11</td>
<td>26</td>
</tr>
<tr>
<td>close</td>
<td>close</td>
<td>8</td>
<td>4</td>
<td>32</td>
</tr>
<tr>
<td>getIndex</td>
<td>getIndex</td>
<td>6</td>
<td>0</td>
<td>36</td>
</tr>
<tr>
<td>add</td>
<td>getIndex</td>
<td>6</td>
<td>0</td>
<td>36</td>
</tr>
<tr>
<td>close</td>
<td>getIndex</td>
<td>14</td>
<td>12</td>
<td>28</td>
</tr>
</tbody>
</table>

$$S = \max(\text{abs}(T - C), 1) \times \max(T, 1)$$
Evaluation - Setup

• 18 thread-safe classes (StringBuffer, Vector, XStream, etc).

• Each benchmark is executed 10 times.

• Timeout: 1 hour for each execution of a benchmark.

• Approaches evaluated:
  ▪ CovCon\textsuperscript{ICSE ’17}: Coverage-based Approach (this talk).
  ▪ ConTeGe\textsuperscript{PLDI ’12}: Random-based Approach.
  ▪ Nainom\textsuperscript{OOPSLA ’14; FSE ’15; PLDI ’15}: Sequential Tests based Approach.
  ▪ AutoConTest\textsuperscript{ICSE ’16}: Coverage-based Approach.
Bug Finding Capability

<table>
<thead>
<tr>
<th>Tool</th>
<th>No. of benchmarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>CovCon (This Talk)</td>
<td>17</td>
</tr>
<tr>
<td>ConTeGe</td>
<td>15</td>
</tr>
<tr>
<td>Nainom</td>
<td>8</td>
</tr>
<tr>
<td>AutoConTest</td>
<td>4</td>
</tr>
</tbody>
</table>
Speedup: Time to Find Bug

![Bar chart showing speedup over existing approaches for different classes under test. The chart compares ConTeGe, Nainom, and AutoContest.]
Speedup: Time to Find Bug

Speedup >= 4x in 22/47 cases
Speedup: Time to Find Bug

- Speedup $\geq 4x$ in 22/47 cases
- Slowdown $\geq 4x$ in 3/47 cases
Conclusion

• Simple. Effective. Efficient.

• Inexpensive coverage analysis.

• Tests generated towards infrequently covered method pairs.

• Dynamically assigns lower priority to method pairs which are synchronized/lock protected.
## Benchmarks

<table>
<thead>
<tr>
<th>Id</th>
<th>Class</th>
<th>Code Base</th>
<th>Type of Bug</th>
<th>Methods (#)</th>
<th>CMPs (#)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>BufferedInputStream</td>
<td>JDK 1.1</td>
<td>Atomicity Violation</td>
<td>9</td>
<td>45</td>
</tr>
<tr>
<td>2</td>
<td>Logger</td>
<td>JDK 1.4.1</td>
<td>Atomicity Violation</td>
<td>44</td>
<td>990</td>
</tr>
<tr>
<td>3</td>
<td>SynchronizedMap</td>
<td>JDK 1.4.2</td>
<td>Deadlock</td>
<td>15</td>
<td>120</td>
</tr>
<tr>
<td>4</td>
<td>ConcurrentHashMap</td>
<td>JDK 1.6.0</td>
<td>Atomicity Violation</td>
<td>22</td>
<td>253</td>
</tr>
<tr>
<td>5</td>
<td>StringBuffer</td>
<td>JDK 1.6.0</td>
<td>Atomicity Violation</td>
<td>52</td>
<td>1378</td>
</tr>
<tr>
<td>6</td>
<td>TimeSeries</td>
<td>JFreeChart 0.9.8</td>
<td>Race Condition</td>
<td>41</td>
<td>861</td>
</tr>
<tr>
<td>7</td>
<td>XYSeries</td>
<td>JFreeChart 0.9.8</td>
<td>Race Condition</td>
<td>25</td>
<td>325</td>
</tr>
<tr>
<td>8</td>
<td>NumberAxis</td>
<td>JFreeChart 0.9.12</td>
<td>Atomicity Violation</td>
<td>110</td>
<td>6105</td>
</tr>
<tr>
<td>9</td>
<td>PeriodAxis</td>
<td>JFreeChart 1.0.1</td>
<td>Race Condition</td>
<td>125</td>
<td>7875</td>
</tr>
<tr>
<td>10</td>
<td>XYPlot</td>
<td>JFreeChart 1.0.9</td>
<td>Race Condition</td>
<td>217</td>
<td>23653</td>
</tr>
<tr>
<td>11</td>
<td>Day</td>
<td>JFreeChart 1.0.13</td>
<td>Race Condition</td>
<td>26</td>
<td>351</td>
</tr>
<tr>
<td>12</td>
<td>PerUserPoolDataSource</td>
<td>CommonsDBCP 1.4</td>
<td>Race Condition</td>
<td>65</td>
<td>2145</td>
</tr>
<tr>
<td>13</td>
<td>SharedPoolDataSource</td>
<td>CommonsDBCP 1.4</td>
<td>Race Condition</td>
<td>51</td>
<td>1326</td>
</tr>
<tr>
<td>14</td>
<td>XStream</td>
<td>XStream 1.4.1</td>
<td>Race Condition</td>
<td>66</td>
<td>2211</td>
</tr>
<tr>
<td>15</td>
<td>Vector</td>
<td>JDK 1.1.7</td>
<td>Atomicity Violation</td>
<td>24</td>
<td>300</td>
</tr>
<tr>
<td>16</td>
<td>Vector</td>
<td>JDK 1.4.2</td>
<td>Atomicity Violation</td>
<td>45</td>
<td>1035</td>
</tr>
<tr>
<td>17</td>
<td>IntRange</td>
<td>Apache Commons 2.4</td>
<td>Atomicity Violation</td>
<td>26</td>
<td>351</td>
</tr>
<tr>
<td>18</td>
<td>AsMap</td>
<td>Google Commons 1.0</td>
<td>Atomicity Violation</td>
<td>15</td>
<td>120</td>
</tr>
</tbody>
</table>
Coverage Measurement Cost

Test Validator
Coverage Detector + Prioritizer
Test Executor
Test Generator

Percentage of overall time

Classes under test
Coverage-driven