jPredictor
A Predictive Runtime Analysis Tool for Java

Traian Florin Șerbănuță
(joint work with Feng Chen and Grigore Roșu)
University of Illinois at Urbana-Champaign
Challenge

- Concurrent programs are hard to analyze
  - Model checking: number of interleavings is prohibitively large
  - Testing: interleavings depend on environment

- What can we do about it?

  Combine dynamic and static methods:
  Find bad behaviors near correct executions.
Our Solution

- **Sliced Causality**
  - General purpose technique to predict (bad) behaviors from correct runs
  - **Sound**: predictions are right

- **jPredictor**
  - Tool implementing Sliced Causality
  - **Better** than tools specialized for detecting dataraces or atomicity violations
Property: “authenticate before access”

Main Thread:
\[ s_1: \text{resource.authenticate}(); \]
\[ s_2: \text{flag.value} = \text{true}; \]
\[ \ldots \]

Task Thread:
\[ \ldots \]
\[ \ldots \]
\[ s_3: \textbf{while} (! \text{flag.value}) \]
\[ \quad \text{Thread.yield}(); \]
\[ s_4: \text{resource.access}(); \]
\[ \ldots \]

Observed execution: \[ \ldots s_1 \ s_2 \ s_3 \ s_4 \ldots \]
Predicting Concurrency Errors

*Property*: “authenticate before access”

Main Thread:

\[ s_1: \text{resource.authenticate}(); \]
\[ s_2: \text{flag.value} = \text{true}; \]
\[ ... \]

Task Thread:

\[ ... \]
\[ ... \]
\[ s_3: \text{if} (! \text{flag.value}) \]
\[ \quad \text{Thread.yield}(); \]
\[ s_4: \text{resource.access}(); \]
\[ ... \]

Observed execution: ... \[ s_1 \, s_2 \, s_3 \, s_4 ... \]

- **Buggy**: \( s_4 \) can be executed before \( s_1 \)
- **Low possibility to hit the error in testing**
Predicting Concurrency Errors

Property: “authenticate before access”

Main Thread:

\( s_1: \text{resource.authenticate}() \);
\( s_2: \text{flag.value} = \text{true}; \)
... 

Task Thread:

... 
... 
\( s_3: \text{if}(! \text{flag.value}) \)
\quad \text{Thread.yield}();
\quad s_4: \text{resource.access}();

Can we predict the error even when the above execution is observed? Yes! But not in a traditional way.

- Buggy: \( s_4 \) can be executed before \( s_1 \)
- Low possibility to hit the error in testing
Predictive Runtime Analysis

Search space
Predictive Runtime Analysis

Search space

Observed execution
Predictive Runtime Analysis

Search space

Observed execution

Causal model
Predictive Runtime Analysis

- Search space
- Observed execution
- Causal model
- Inferred executions
- Bug
Predictive Runtime Analysis

- Search space
- Observed execution
- Causal model
- Inferred executions
- Bug

More relaxed causal model yields more inferred executions
Traditional Predictive Runtime Analysis: Happens-Before

- Originally for distributed systems [Lamport-78]
  - Applied to shared-memory systems by many authors
- Causal model = non-permutable pairs of events
  - = \{intra-thread total orders\} \cup \{causal dependencies\}
  - Causal dependency: if two events access same location and one writes it, then their execution order matters
- Inferred executions = extending the causal model
Happens-Before Works ... If Lucky

**Property:** “authenticate before access”

**Main Thread:**

\[ s_1: \text{resource.authenticate()} \]
\[ s_2: \text{flag.value = true;} \]

**Task Thread:**

\[ s_3: \text{if (! flag.value)} \]
\[ \quad \text{Thread.yield();} \]
\[ s_4: \text{resource.access();} \]

Observed execution: \( s_3 \ s_1 \ s_2 \ s_4 \)
Happens-Before Works ... If Lucky

**Property**: “authenticate before access”

Main Thread:

\[s_1: \text{resource.authenticate}()\]
\[s_2: \text{flag.value} = \text{true};\]

Task Thread:

\[s_3: \text{if} \ (! \text{flag.value})\]
\[\text{Thread.yield}();\]
\[s_4: \text{resource.access}();\]

Observed execution: \(s_3 \ s_1 \ s_2 \ s_4\)

Causal dependency: \(s_3 < s_2\)

Bad execution inferred: \(s_3 \ s_4 \ s_1 \ s_2\). Bug detected!

Chances of observing this execution are very low
Happens-Before: Limitations

**Property**: “authenticate before access”

**Main Thread**: 

- $s_1$: resource.authenticate()
- $s_2$: flag.value = true;

**Task Thread**: 

- $s_3$: if (! flag.value) Thread.yield();
- $s_4$: resource.access();

Observed execution: $s_1 \ s_2 \ s_3 \ s_4$

Causal dependency: $s_2 < s_3$. No bug found …

Too constrained: access will be performed regardless of the flag
Our Technique: Sliced Causality

- Relax the Happens-Before causal model
  - Yields no false alarms: formally proved in [chen-rosu-07]
- How? Focus on the property!
- Use static information about the program
- Remove events and causalities irrelevant to property
  - Smaller and more relaxed causal model
  - (exponentially) more inferred executions
  - better predictive capability
Static Information: Control Scope
[chen-rosu-06]

- **S₂** is in the control scope of **S₁** if its execution depends on a choice at **S₁**
  
  ```
  s₀: i=0;
  s₁: while (!flag) {
      s₂:  ...
      s₃:  i++
  }
  s₄: ...
  
  s₁: if (flag) {
      s₂:  ...
      s₃:  ...
  } else {
      s₂:  ...
      s₃:  ...
  }
  ```

- Extends to other control statements
  - break/continue, return, exceptions
Static Information: Control Scope
[chen-rosu-06]

- $S_2$ is in the control scope of $S_1$ if its execution depends on a choice at $S_1$

  - $s_1$: if (flag) {
    - $s_2$: ...
    - } else {
    - $s_3$: ...
    - }
  - $s_4$: ...

  - $s_0$: i=0;
  - $s_1$: while (i<3) {
    - $s_2$: ...
    - $s_3$: i++
    - }
  - $s_4$: ...

- Extends to other control statements
  - break/continue, return, exceptions
Static Information: Control Scope
[chen-rosu-06]

- **S₂** is in the control scope of **S₁** if its execution depends on a choice at **S₁**

  ```
  s₁: if (flag) {
    s₂: ...
    } else {
    s₃: ...
    }

  s₄: ...
  ```

- Extends to other control statements
  - break/continue, return, exceptions

```diff
  s₀: i=0;
  s₁: while (i<3) {
    s₂: ...
    s₃: i++
    }
  s₄: ...

  s₁: while (!flag) {
    s₂: ...
    }
  s₃: ...
```
Static Information: Control Scope
[chen-rosu-06]

- **S_2** is in the control scope of **S_1** if its execution depends on a choice at **S_1**

\[s_1: \text{if (flag) } \{
\]
\[s_2: \text{...}
\]
\[\} \text{ else } \{
\]
\[s_3: \text{...}
\]
\[\}
\]
\[s_4: \text{...}
\]

- Extends to other control statements
  - break/continue, return, exceptions
Sliced Causality Works!

*Property*: “authenticate before access”

**Main Thread:**
- $s_1$: `resource.authenticate()`
- $s_2$: `flag.value = true;`

**Task Thread:**
- $s_3$: if (! `flag.value`)  
  `Thread.yield();`
- $s_4$: `resource.access();`

Observed execution: $s_1 \ s_2 \ s_3 \ s_4$

Only $s_1$ and $s_4$ directly relevant to the property
Sliced Causality Works!

Property: “authenticate before access”

Main Thread:

s₁: resource.authenticate()
s₂: flag.value = true;

Task Thread:

s₃: if (! flag.value)
    Thread.yield();
    s₄: resource.access();

Observed execution: s₁ s₂ s₃ s₄

Only s₁ and s₄ directly relevant to the property

Execution of s₄ not dependent of s₃; ignore the causal dependency s₂ < s₃

Sliced causality: s₁ <> s₄; s₄ s₁ is a potential execution. Bug detected!
No False Alarms 😊

Property: “authenticate before access”

Main Thread:

s_1: resource.authenticate()

s_2: flag.value = true;

Task Thread:

s_3: while (! flag.value)
    Thread.yield();

s_4: resource.access();

Observed execution: s_1 s_2 s_3 s_4
Property: “authenticate before access”

Main Thread:
\[s_1: \text{resource.authenticate()}\]
\[s_2: \text{flag.value} = \text{true};\]

Task Thread:
\[s_3: \textbf{while} (! \text{flag.value})\]
\[
\text{Thread.yield();} \\
\text{s_4: resource.access();}
\]

Observed execution: \(s_1 \ s_2 \ s_3 \ s_4\)
Execution of \(s_4\) depends on \text{flag.value} being \text{true} at \(s_3\)
causal dependency \(s_2 < s_3\) matters
Sliced causality: \(s_1 < s_2 < s_3 < s_4\), no false alarm!
jPredictor: Black-Box View

Original Program

Static Analyzer

Instrumented Program

JVM

Recorded Trace

Preprocessor

Complete Trace

Trace Slicer

Sliced Trace

Vector Clock Calculator

Causal Model

Property Checker

Predicted Violations: Counter-Examples
jPredictor: Filling the box

Original Program

Property

Predicted Violations: Counter-Examples
jPredictor: Filling the box

Original Program

Static Analyzer

Property

Predicted Violations: Counter-Examples
jPredictor: Filling the box

Original Program

Static Analyzer

Instrumented Program

Structural Information

Property

Predicted Violations: Counter-Examples
jPredictor: Filling the box

Original Program

Static Analyzer

Instrumented Program

JVM

Recorded Trace

Preprocessor

Complete Trace

Property

Structural Information

Predicted Violations: Counter-Examples
jPredictor: Filling the box

Original Program

Static Analyzer

Instrumented Program

JVM

Recorded Trace

Preprocessor

Complete Trace

Trace Slicer

Sliced Trace

Property

Predicted Violations: Counter-Examples
jPredictor: Filling the box

Original Program

Static Analyzer

Instrumented Program

JVM

Recorded Trace

Preprocessor

Complete Trace

Trace Slicer

Sliced Trace

Vector Clock Calculator

Causal Model

Property

Predicted Violations: Counter-Examples
jPredictor: Filling the box

Original Program

Static Analyzer

Instrumented Program

JVM

Recorded Trace

Preprocessor

Complete Trace

Trace Slicer

Sliced Trace

Vector Clock Calculator

Causal Model

Property Checker

Property

Predicted Violations: Counter-Examples
Evaluation

- Evaluated on many real life applications
  - Benchmark from previous work
    - Banking, tsp, hedc…
  - Open source programs
    - Apache FTP server, Apache Common library, Tomcat…
  - Industry programs
    - Java Collection Library, IBM WebCrawler…

- Focused on dataraces and atomicity violations
  - Sliced causality is general purpose
    - Replacement for Happened-Before Causality
Results: Datarace Detection

- Found almost all previously reported bugs in just one or few runs
- Found new bugs, no false alarms

<table>
<thead>
<tr>
<th>Benchmarks</th>
<th>Time (seconds)</th>
<th>Races</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Preprocessing</td>
<td>Analysis</td>
</tr>
<tr>
<td>Elevator</td>
<td>10.3</td>
<td>0.15</td>
</tr>
<tr>
<td>Tsp</td>
<td>30.6</td>
<td>0.23</td>
</tr>
<tr>
<td>Sor</td>
<td>2.4</td>
<td>0.01</td>
</tr>
<tr>
<td>Hedc</td>
<td>20.16</td>
<td>0.01</td>
</tr>
<tr>
<td>StringBuffer</td>
<td>0.38</td>
<td>0.01</td>
</tr>
<tr>
<td>Vector</td>
<td>0.44</td>
<td>0.01</td>
</tr>
<tr>
<td>IBM WebCrawler</td>
<td>0.94</td>
<td>0.01</td>
</tr>
<tr>
<td>StaticBucketMap</td>
<td>151.4</td>
<td>0.03</td>
</tr>
<tr>
<td>Pool 1.2</td>
<td>2.02</td>
<td>0.01</td>
</tr>
<tr>
<td>Pool 1.3</td>
<td>1.15</td>
<td>0.01</td>
</tr>
<tr>
<td>Apache FTP Serv.</td>
<td>4.9</td>
<td>0.02</td>
</tr>
</tbody>
</table>

red = new bugs
Results: Datarace Detection

- Found almost all previously reported bugs in just one or few runs
- Found new bugs, no false alarms

<table>
<thead>
<tr>
<th>Benchmarks</th>
<th>Time (seconds)</th>
<th>Races</th>
<th>Hybrid Dynamic [O'Callahan-Choi-03]</th>
<th>Static Analysis [Naik-Aiken-Whaley-06]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Preprocessing</td>
<td>Analysis</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Elevator</td>
<td>10.3</td>
<td>0.15</td>
<td>0</td>
<td>0 (0)</td>
</tr>
<tr>
<td>Tsp</td>
<td>30.6</td>
<td>0.23</td>
<td>1</td>
<td>1 (2) 1 (12)</td>
</tr>
<tr>
<td>Sor</td>
<td>2.4</td>
<td>0.01</td>
<td>0</td>
<td>0 (0)</td>
</tr>
<tr>
<td>Hedc</td>
<td>20.16</td>
<td>0.01</td>
<td>4</td>
<td>4 (0) 4 (13)</td>
</tr>
<tr>
<td>StringBuffer</td>
<td>0.38</td>
<td>0.01</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Vector</td>
<td>0.44</td>
<td>0.01</td>
<td>1</td>
<td>1 (0)</td>
</tr>
<tr>
<td>IBM WebCrawler</td>
<td>0.94</td>
<td>0.01</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>StaticBucketMap</td>
<td>151.4</td>
<td>0.03</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Pool 1.2</td>
<td>2.02</td>
<td>0.01</td>
<td>18</td>
<td>17 (0)</td>
</tr>
<tr>
<td>Pool 1.3</td>
<td>1.15</td>
<td>0.01</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Apache FTP Serv.</td>
<td>4.9</td>
<td>0.02</td>
<td>11</td>
<td>12 (23)</td>
</tr>
</tbody>
</table>

red = new bugs races found(false alarms)
## Results: Datarace Detection

- Found almost all previously reported bugs in just **one or few runs**
- Found **new bugs, no false alarms**

<table>
<thead>
<tr>
<th>Benchmarks</th>
<th>Time (seconds)</th>
<th>Races</th>
<th>Hybrid Dynamic [O'Callahan-Choi-03]</th>
<th>Static Analysis [Naik-Aiken-Whaley-06]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Preprocessing</td>
<td>Analysis</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Elevator</td>
<td>10.3</td>
<td>0.15</td>
<td>0</td>
<td>0 (0)</td>
</tr>
<tr>
<td>Tsp</td>
<td>30.6</td>
<td>0.23</td>
<td>1</td>
<td>1 (2)</td>
</tr>
<tr>
<td>Sor</td>
<td>2.4</td>
<td>0.01</td>
<td>0</td>
<td>0 (0)</td>
</tr>
<tr>
<td>Hedc</td>
<td>20.16</td>
<td>0.01</td>
<td>4</td>
<td>4 (0)</td>
</tr>
<tr>
<td>StringBuffer</td>
<td>0.38</td>
<td>0.01</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Vector</td>
<td>0.44</td>
<td>0.01</td>
<td>1</td>
<td>1 (0)</td>
</tr>
<tr>
<td>IBM WebCrawler</td>
<td>0.94</td>
<td>0.01</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>StaticBucketMap</td>
<td>151.4</td>
<td>0.03</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Pool 1.2</td>
<td>2.02</td>
<td>0.01</td>
<td>18</td>
<td>17 (0)</td>
</tr>
<tr>
<td>Pool 1.3</td>
<td>1.15</td>
<td>0.01</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Apache FTP Serv.</td>
<td>4.9</td>
<td>0.02</td>
<td>11</td>
<td>12 (23)</td>
</tr>
</tbody>
</table>

**red** = new bugs  
**races found (false alarms)**
## Results: Datarace Detection

- Found almost all previously reported bugs in just **one or few runs**
- Found **new bugs, no false alarms**

<table>
<thead>
<tr>
<th>Benchmarks</th>
<th>Time (seconds)</th>
<th>Races</th>
<th>Hybrid Dynamic</th>
<th>Static Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Preprocessing</td>
<td>Analysis</td>
<td>[O'Callahan-Choi-03]</td>
<td>[Naik-Aiken-Whaley-06]</td>
</tr>
<tr>
<td>Elevator</td>
<td>10.3</td>
<td>0.15</td>
<td>0</td>
<td>0 (0)</td>
</tr>
<tr>
<td>Tsp</td>
<td>30.6</td>
<td>0.23</td>
<td>1</td>
<td>1 (2)</td>
</tr>
<tr>
<td>Sor</td>
<td>2.4</td>
<td>0.01</td>
<td>0</td>
<td>0 (0)</td>
</tr>
<tr>
<td>Hedc</td>
<td>20.16</td>
<td>0.01</td>
<td>4</td>
<td>4 (0)</td>
</tr>
<tr>
<td>StringBuffer</td>
<td>0.38</td>
<td>0.01</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Vector</td>
<td>0.44</td>
<td>0.01</td>
<td>1</td>
<td>1 (0)</td>
</tr>
<tr>
<td>IBM WebCrawler</td>
<td>0.94</td>
<td>0.01</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>StaticBucketMap</td>
<td>151.4</td>
<td>0.03</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Pool 1.2</td>
<td>2.02</td>
<td>0.01</td>
<td>18</td>
<td>17 (0)</td>
</tr>
<tr>
<td>Pool 1.3</td>
<td>1.15</td>
<td>0.01</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Apache FTP Serv.</td>
<td>4.9</td>
<td>0.02</td>
<td>11</td>
<td>12 (23)</td>
</tr>
</tbody>
</table>

**red** = new bugs  
**races** = found(false alarms)
Results: Atomicity Violations

- Check atomicity of synchronized methods
- Based on atomicity conditions from [Vaziri-Tip-Dolby-06]
- Found all known bugs plus new bugs, no false alarms

<table>
<thead>
<tr>
<th>Benchmarks</th>
<th>Analysis Time* (seconds)</th>
<th>Violations</th>
<th>Atomizer [flanagan-freund-04]</th>
<th>Other [wang-stoller-06]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elevator</td>
<td>4.2</td>
<td>0</td>
<td>0 (2)</td>
<td>0 (2)</td>
</tr>
<tr>
<td>Sor</td>
<td>3.6</td>
<td>1</td>
<td>0 (0)</td>
<td>0 (0)</td>
</tr>
<tr>
<td>Tsp</td>
<td>1.16</td>
<td>0</td>
<td>0 (7)</td>
<td>0 (2)</td>
</tr>
<tr>
<td>Hedc</td>
<td>0.22</td>
<td>1</td>
<td>1 (4)</td>
<td>0 (0)</td>
</tr>
<tr>
<td>StringBuffer</td>
<td>0.07</td>
<td>1</td>
<td></td>
<td>0 (1)</td>
</tr>
<tr>
<td>Vector</td>
<td>0.27</td>
<td>4</td>
<td></td>
<td>4 (4)</td>
</tr>
<tr>
<td>StaticBucketMap</td>
<td>22.03</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pool 1.2</td>
<td>5.5</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pool 1.3</td>
<td>0.83</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* We are reusing the slices generated for race detection
Conclusions and Future Work

- Sliced Causality: **sound** technique for predicting concurrency errors
- jPredictor: data-races **and** atomicity violations
  - As good, or better than current specialized tools

- Online jPredictor: predict bugs during runtime
  - Embedding in JVM
- Combine with test-generation techniques
Thank you!

http://fsl.cs.uiuc.edu/jPredictor

Predicted Violations: Counter-Examples