Effectiveness and Challenges in Generating Concurrent Tests for Thread-Safe Classes

Valerio Terragni*

Mauro Pezzè*

* USI Università della Svizzera italiana, Switzerland

Università di Milano Bicocca, Italy

5 September, Montpellier, France
"A class that encapsulates synchronizations that ensure a correct behavior when the same instance of the class is accessed from multiple threads"
Achieving Optimal Synchronization is Challenging

Performance

Correctness
Thread-Safe Classes are Buggy

Exception when using SharedPoolDataSource concurrently

#278 Axis classes are not Thread Safe

Status: closed-fixed
Priority: 9
Updated: 2003-11-07

Race Condition in class java.util.logging.Logger

Details
Type: Bug
Priority: P4
Affects Version/s: 1.4.0, 1.4.1, 7
Component/s: core-libs
Subcomponent: java.util.logging
Resolved In Build: b16
CPU: generic, x86, sparc
OS: generic, solaris_7, windows_2000
Verification: Not verified

Thread safety bug: ENV clobbered #1709

Closed: balexand opened this issue on Nov 22, 2014 · 10 comments
CPU: generic,x86
Example of a Thread-Safety Violation

```java
public void log(LogRecord r) {
    synchronized (this) {
        if (filter != null) {
            if (!filter.isLoggable(r)) {
                return;
            }
        }
    }
}
```

```java
public void setFilter(Filter f) {
    this.filter = f;
}
```

**Null Pointer exception**
Set of method call sequences that exercise the public interface of a class from multiple threads.

Logger `sout = Logger.getLoggerAnonymousLogger();`
Filter `filter0 = new Filter();`
`sout.setFilter(filter0);

Thread 1
```
sout.info("");```
Thread 2
```
sout.setFilter(null);
```
Concurrent Test Generation

State of the Art

2012
Ballerina@ICSE
ConTeGe@PLDI

2013
ConSuite@ICST

2014
Narada@PLDI
Intruder@FSE

2015

2016
AutoConTest@ICSE
Minion@OOPSLA

2017
CovCon@ICSE
Concurrent Test Generation

ARE WE THERE YET?
1. A survey on existing concurrent test generators
2. A large-scale experimental evaluation of 6 generators
3. Analysis of their limitations
4. Guidelines for future research in this area
## State of the Art

### Concurrent Test Generation

<table>
<thead>
<tr>
<th>Tool name</th>
<th>Venue</th>
<th>Year</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ballerina</td>
<td>ICSE</td>
<td>2012</td>
<td>Random-based</td>
</tr>
<tr>
<td>ConTeGe</td>
<td>PLDI</td>
<td>2012</td>
<td></td>
</tr>
<tr>
<td>ConSuite</td>
<td>ICST</td>
<td>2013</td>
<td></td>
</tr>
<tr>
<td>AutoConTest</td>
<td>ICSE</td>
<td>2016</td>
<td>Coverage-based</td>
</tr>
<tr>
<td>CovCon</td>
<td>ICSE</td>
<td>2017</td>
<td></td>
</tr>
<tr>
<td>Omen</td>
<td>OOPSLA</td>
<td>2014</td>
<td></td>
</tr>
<tr>
<td>Narada</td>
<td>PLDI</td>
<td>2015</td>
<td>Sequential-test-based</td>
</tr>
<tr>
<td>Intruder</td>
<td>FSE</td>
<td>2015</td>
<td></td>
</tr>
<tr>
<td>Minion</td>
<td>OOPSLA</td>
<td>2016</td>
<td></td>
</tr>
</tbody>
</table>
Random-Based

<table>
<thead>
<tr>
<th>Tool name</th>
<th>Venue</th>
<th>Year</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ballerina</td>
<td>ICSE</td>
<td>2012</td>
<td><strong>Random-based</strong></td>
</tr>
<tr>
<td>ConTeGe</td>
<td>PLDI</td>
<td>2012</td>
<td></td>
</tr>
</tbody>
</table>

- Many randomly generated tests are needed
- Many redundant tests are generated
- Low computational analysis
### Coverage-Based

<table>
<thead>
<tr>
<th>Tool name</th>
<th>Venue</th>
<th>Year</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConSuite</td>
<td>ICST</td>
<td>2013</td>
<td></td>
</tr>
<tr>
<td>AutoConTest</td>
<td>ICSE</td>
<td>2016</td>
<td><strong>Coverage-based</strong></td>
</tr>
<tr>
<td>CovCon</td>
<td>ICSE</td>
<td>2017</td>
<td></td>
</tr>
</tbody>
</table>

- **Limited number of redundant tests**
- **Difficult trade-off between:**
  - A precise computation of coverage targets
  - Low analysis computational cost
## Sequential-Test Based

### Concurrent Test Generation

<table>
<thead>
<tr>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUT</td>
</tr>
<tr>
<td>Auxiliary Classes</td>
</tr>
</tbody>
</table>

#### Step 1
- **Call Sequence Generator**
- **Method Call Sequences**: S1, S2, S3...

#### Step 2
- **Call Sequences Assembler**

### Concurrent Test Generation

<table>
<thead>
<tr>
<th>Tool name</th>
<th>Venue</th>
<th>Year</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Omen</td>
<td>OOPSLA</td>
<td>2014</td>
<td>Generate only those tests that reveal the considered type of bug</td>
</tr>
<tr>
<td>Narada</td>
<td>PLDI</td>
<td>2015</td>
<td>Require a seeded sequential test suites in input</td>
</tr>
<tr>
<td>Intruder</td>
<td>FSE</td>
<td>2015</td>
<td></td>
</tr>
<tr>
<td>Minion</td>
<td>OOPSLA</td>
<td>2016</td>
<td><strong>Sequential-test-based</strong></td>
</tr>
</tbody>
</table>

**Sequential-test-based**
## Interleaving Exploration & Thread-Safety Oracle

![Diagram of Interleaving Exploration and Thread-Safety Oracle]

<table>
<thead>
<tr>
<th>Tool name</th>
<th>Venue</th>
<th>Year</th>
<th>Random</th>
<th>Selective</th>
<th>Exhaustive</th>
<th>Implicit</th>
<th>Internal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ballerina</td>
<td>ICSE</td>
<td>2012</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>ConTeGe</td>
<td>PLDI</td>
<td>2012</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>ConSuite</td>
<td>ICST</td>
<td>2013</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AutoConTest</td>
<td>ICSE</td>
<td>2016</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CovCon</td>
<td>ICSE</td>
<td>2017</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Omen</td>
<td>OOPSLA</td>
<td>2014</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Narada</td>
<td>PLDI</td>
<td>2015</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Intruder</td>
<td>FSE</td>
<td>2015</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Minion</td>
<td>OOPSLA</td>
<td>2016</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Contributions (Outline)

1. A survey on existing concurrent test generators
2. A large-scale experimental evaluation of 6 generators
3. Analysis of their limitations
4. Guidelines for future research in this area
# JaConTeBe: A Benchmark Suite of Real-World Java Concurrency Bugs

**ASE 2015**

Ziyi Lin*, Darko Marinov†, Hao Zhong‡, Yuting Chen‡, and Jianjun Zhao‡

<table>
<thead>
<tr>
<th>Code base (label)</th>
<th># of subjects (bugs)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache DBCP (dbcp)</td>
<td>4</td>
<td>Database connection pool</td>
</tr>
<tr>
<td>Apache Derby (derby)</td>
<td>5</td>
<td>Relational database</td>
</tr>
<tr>
<td>Apache Groovy (groovy)</td>
<td>6</td>
<td>Dynamic language for JVM</td>
</tr>
<tr>
<td>OpenJDK (jdk)</td>
<td>20</td>
<td>Java Development Kit</td>
</tr>
<tr>
<td>Apache Log4J (log4j)</td>
<td>5</td>
<td>Logging library</td>
</tr>
<tr>
<td>Apache Lucene (lucene)</td>
<td>2</td>
<td>Search library</td>
</tr>
<tr>
<td>Apache Pool (pool)</td>
<td>5</td>
<td>Object-pooling API</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>47</strong></td>
<td></td>
</tr>
</tbody>
</table>
Evaluation Setup

JaConTeBe 47 subjects

Class Under Test (CUT) Auxiliary classes

Buggy code base
Bug report
Manually-written test

Time budget **one hour** per subject

10 runs per subject
<table>
<thead>
<tr>
<th>Fault Type</th>
<th>Failure Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConTeGe</td>
<td>ConTeGeJPF</td>
</tr>
<tr>
<td>AutoConTest</td>
<td>CovCon</td>
</tr>
<tr>
<td>CovConJPF</td>
<td>Intruder</td>
</tr>
<tr>
<td>Fault Type</td>
<td>Failure Type</td>
</tr>
<tr>
<td>-------------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>inconsistent</td>
<td>endless loop</td>
</tr>
<tr>
<td>synchronization</td>
<td></td>
</tr>
<tr>
<td></td>
<td>logic</td>
</tr>
<tr>
<td>race</td>
<td></td>
</tr>
<tr>
<td>atomicity</td>
<td></td>
</tr>
<tr>
<td>violations</td>
<td></td>
</tr>
<tr>
<td></td>
<td>endless loop</td>
</tr>
<tr>
<td></td>
<td>exception</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>logic</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>endless hang</td>
</tr>
<tr>
<td>resource</td>
<td></td>
</tr>
<tr>
<td>deadlock</td>
<td></td>
</tr>
<tr>
<td>wait-notify</td>
<td></td>
</tr>
<tr>
<td>deadlock</td>
<td></td>
</tr>
<tr>
<td>Fault Type</td>
<td>Failure Type</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>inconsistent synchronization</td>
<td>endless loop</td>
</tr>
<tr>
<td></td>
<td>logic</td>
</tr>
<tr>
<td></td>
<td>endless loop</td>
</tr>
<tr>
<td>race</td>
<td>exception</td>
</tr>
<tr>
<td>atomicity</td>
<td></td>
</tr>
<tr>
<td>violations</td>
<td></td>
</tr>
<tr>
<td>logic</td>
<td></td>
</tr>
<tr>
<td>resource</td>
<td>endless hang</td>
</tr>
<tr>
<td>deadlock</td>
<td></td>
</tr>
<tr>
<td>wait-notify</td>
<td>endless hang</td>
</tr>
<tr>
<td>deadlock</td>
<td></td>
</tr>
</tbody>
</table>

- Automated concurrent test generators find (8 out of 47) 17% of the faults
- None of them alone finds more than (6 out of 47) 13% of the faults
Contributions (Outline)

1. A survey on existing concurrent test generators
2. A large-scale experimental evaluation of 6 generators
3. Analysis of their limitations
4. Guidelines for future research in this area
Analysis of the Tools Limitations

JaConTeBe 47 subjects

Buggy code base

Bug report

Manually-written test

Manual Investigation

Concurrent Test Generation

Step 1
Call Sequence Generator

Step 2
Method Call Sequences S1, S2, S3...

Call Sequences Assembler

Concurrent Test

S1

Thread 1

Thread 2

S2

S3

Interleaving Exploration

Step 3
Interleaving Explorer

Step 4
Thread Interleavings

Thread-safety Oracle

Output

Thread-safety Violations
Common Issue 1: Invalid Assumptions

40% faults violate at least one of the following assumptions

• Two threads only
• One shared object under test
• No static invocations

manually-written test : derby5

```java
... 
storePage.setExclusive(baseContainerHandle);
baseContainerHandle.addObserver(storePage);

Thread 1

Thread 2

baseContainerHandle.close();
storePage.releaseExclusive();
```

Assumption violated: 2 shared objects under test
Common Issue 2: Environmental Dependencies

25% of the faults require environmental dependencies (DB, files ...)

manually-written test : jdk6_2

String dirA = projectBase + "\base\a";
String dirB = projectBase + "\base\b";

File fileA = new File(dirA);
fileA.mkdirs();

File fileB = new File(dirB);
fileB.mkdirs();
Common Issue 3: Inadequacy for Wait-Notify

19% of the faults require the execution of `wait()-notify()`

(Step 1) **Feedback-directed approach**
Sequential (single-thread) execution of call sequences
**Discards** sequences that throw exceptions or **never terminates (time-out)**

```java
ClassA sout = new ClassA();
sout.m1();
sout.m2();
sout.m3();
```

```java
public void m3() {
    ...
    lock.wait();  // Time Out!
    ...
}
```
1. A survey on existing concurrent test generators

2. A large-scale experimental evaluation of 6 generators

3. Analysis of their limitations

4. Guidelines for future research in this area
Adaptive Configuration

Automatically identify a proper configuration for a given class under test
Search Space Reduction

Reduce the search space by identifying methods whose concurrent interactions cannot lead to concurrency failures.
Handling Wait and Notify

Revise the 4 steps framework to handle wait/notify synchronization primitives
Revise the 4 steps framework to handle wait/notify synchronization primitives.
Conclusion

Concurrent Test Generation

State of the Art

ConSuite@ICST  Narada@PLDI  CovCon@ICSE
2012  2013  2014  2015  2016  2017
Intruder@FSE

Ballerina@ICSE  Omen@FSE  AutoConTest@ICSE
ConTeGe@PLDI  Minion@OOPSLA

Subjects

JaConTeBe: A Benchmark Suite of Real-World Java Concurrency Bugs

ASE 2015
Ziyi Lin, Darío Martínez, Hao Zhong, Yuting Chen, and Jianjun Zhao

<table>
<thead>
<tr>
<th>Code base (label)</th>
<th># of subjects (bugs)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache DBCP (dbcp)</td>
<td>4</td>
<td>Database connection pool</td>
</tr>
<tr>
<td>Apache Derby (derby)</td>
<td>5</td>
<td>Relational database</td>
</tr>
<tr>
<td>Apache Groovy (groovy)</td>
<td>6</td>
<td>Dynamic language for JVM</td>
</tr>
<tr>
<td>OpenJDK (jdk)</td>
<td>20</td>
<td>Java Development Kit</td>
</tr>
<tr>
<td>Apache Log4J (log4j)</td>
<td>5</td>
<td>Logging library</td>
</tr>
<tr>
<td>Apache Lucene (lucene)</td>
<td>2</td>
<td>Search library</td>
</tr>
<tr>
<td>Apache Pool (pool)</td>
<td>5</td>
<td>Object-pooling API</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>47</strong></td>
<td></td>
</tr>
</tbody>
</table>

Analysis of the Tools Limitations

JaConTeBe 47 subjects

- Automated concurrent test generators find (8 out of 47) 17% of the faults
- None of them alone finds more than (6 out of 47) 13% of the faults

Buggy code base
Bug report
Manually-written test

Manual Investigation

Concurrent Test Generation

Step 1
Call Sequence Generator
Input
Call Sequence
Method
Argument
Thread
Sequence
Step 2
Interleaving Exploration
Thread
Chains
Step 3
Concurrent Test
Input
S1,S2,S3
Step 4
Output
Thread-safety Violations
Artifact is available!

http://star.inf.usi.ch/star/software/contest2018

- Runnable scripts
- Experimental data
Contributions (Outline)

1. A survey on existing concurrent test generators
2. A large-scale experimental evaluation of 6 generators
3. Analysis of their limitations
4. Guidelines for future research in this area