SpeedGun: 
Performance Regression Testing of Concurrent Classes

Michael Pradel¹, Markus Huggler², and Thomas R. Gross²

¹ University of California, Berkeley  
² ETH Zurich
Motivation

Writing concurrent software is difficult

**Correctness:**
Synchronize concurrent accesses to shared data

**Performance:**
Avoid unnecessary synchronization
Motivation

Writing concurrent software is difficult

Correctness:
Synchronize concurrent accesses to shared data
Data races
Atomicity violations
Thread safety
Schedule exploration

Performance:
Avoid unnecessary synchronization
Real-World Example

History of Groovy’s ExpandoMetaClass

Correctness
- Bug 2166: Missing synchronization

Performance
- Bug 3557: Too much synchronization
Real-World Example

History of Groovy’s ExpandoMetaClass

Correctness

Bug 2166: Missing synchronization

Performance

Bug 3557: Too much synchronization

Bug 3657: Too much synchronization

...
Real-World Example

History of Groovy’s `ExpandoMetaClass`

Correctness

Bug 2166: Missing synchronization

Performance

Bug 3557: Too much synchronization

Bug 2166: Missing synchronization

Bug 3557: Too much synchronization
Real-World Example (2)

class ExpandoMetaClass {
    private boolean initialized;
    synchronized void initialize() {
        if (!this.initialized) {
            this.initialized = true;
        }
    }
    boolean isInitialized() {
        return this.initialized;
    }
}

Before

bug 2166
Real-World Example (2)

```java
class ExpandoMetaClass {
    private boolean initialized;
    synchronized void initialize() {
        if (!isInitialized()) {
            setInitialized(true);
        }
    }
    synchronized boolean isInitialized() {
        return this.initialized;
    }
    synchronized void setInitialized(boolean b) {
        this.initialized = b;
    }
}
```

Fix for bug 2166
class ExpandoMetaClass {
    private volatile boolean initialized;
    synchronized void initialize() {
        if (!isInitialized()) {
            setInitialized(true);
        }
    }

    boolean isInitialized() {
        return this.initialized;
    }

    void setInitialized(boolean b) {
        this.initialized = b;
    }
}

Fix for bug 3557
**SpeedGun: Overview**

Automated performance regression testing for thread-safe classes

Version 1

Generation of concurrent performance tests

Test execution and performance measurement

Test Oracle

Regression, improvement, or none

Version 2
Challenges

Measuring performance ain’t easy
Challenges

Measuring performance ain’t easy of concurrent software at all
Challenges

Measuring performance ain’t easy at all
- Measurement accuracy
  → Minimum measurable timespan
Challenges

Measuring performance ain’t easy at all

- Measurement accuracy
  - Minimum measurable timespan
- Thread scheduling
  - Repeated execution
Challenges

Measuring performance ain’t easy at all of concurrent software

- Measurement accuracy
  - Minimum measurable timespan
- Thread scheduling
  - Repeated execution
- Just-in-time compilation
  - Warm up + steady state
Challenges

Measuring performance ain’t easy at all

- Measurement accuracy
  - Minimum measurable timespan
- Thread scheduling
  - Repeated execution
- Just-in-time compilation
  - Warm up + steady state
- Garbage collection
  - Invoke before measurements
Concurrent Tests: Example

Sequential prefix + concurrent suffixes

```
ExpandoMetaClassInit v0 = new ExpandoMetaClassInit();
ExpandoMetaClass v1 = v0.unInitializedExpandoMetaClass();
Class v2 = v1.getJavaClass();
ExpandoMetaClass x = new ExpandoMetaClass(v2, true);
x.getExpandoMethods();

Thread 1       ...
String v4 = x.toString();
x.respondsTo(v4, v4, null);
x.isModified();
...

Thread N
x.initialize();
x.getClassNode();
x.getProperties();
...
```
Feedback-directed random generation of concurrent tests [PLDI’12]

Here: Long tests with many suffixes

- Exceed minimum measurable timespan
- High degree of concurrency
Test Generation: Scalability

Challenge: Scaling feedback-directed test generation to large tests

Goal: \( \text{L calls} \)
Test Generation: Scalability

Challenge: Scaling feedback-directed test generation to large tests

Goal: \( L \) calls
Test Generation: Scalability

Challenge: Scaling feedback-directed test generation to large tests

Goal: $L$ calls
Test Generation: Scalability

Challenge: Scaling feedback-directed test generation to large tests

Goal: \( L \) calls
Test Generation: Scalability

Challenge: Scaling feedback-directed test generation to large tests

Goal: Execute at least $1 + \ldots + L = \frac{L^2 + L}{2}$ calls
Test Generation: Scalability

Approach: Generate smaller sequences and repeat them

Repeat \( \sqrt{L} \) calls \( \sqrt{L} \) times

\[ = \]

Goal: \( L \) calls
Test Generation: Scalability

Approach: Generate smaller sequences and repeat them

Repeat \( \sqrt{L} \) calls \( \sqrt{L} \) times = Goal: \( L \) calls

Execute at least \( 1 + \ldots + \sqrt{L} = \frac{L + \sqrt{L}}{2} \) calls
Test Execution (Single)

How to measure test execution time?

Prefix
Thread setup
Suffixes
Thread cleanup
Test Execution (Single)

How to measure test execution time?

Prefix
Thread setup

Suffixes =
Measured time

Thread cleanup
Test Execution (Repeated)

Warm up

Steady state

Steady state

Set $\mathcal{M}$ of measurements

$\text{time}$

$\text{= test execution}$
Test Execution (Repeated)

Warm up \[\cdots\] Steady state \[\cdots\] Steady state

Set $\mathcal{M}$ of measurements

Add measurements until variance is within fixed bounds: \[\sigma(\mathcal{M}) \leq \overline{\mathcal{M}} \cdot \beta\]

- Standard deviation
- Mean
- Default: 0.01
Length of Tests

How long should tests be?
Length of Tests

How long should tests be?

Steady state

Average execution time of test $T$: $t_T$
Repetitions: $r_T$

Constraints:

- $t_T > \text{minimum measurable timespan}$
- $r_T > \text{minimum number of repetitions}$
Approach: Binary search

Repeat until constraints fulfilled:

- Generate a test $T$
- Execute and measure $t_T$ and $r_T$ for both versions
Length of Tests (2)

Approach: **Binary search**

Repeat until constraints fulfilled:

- Generate a test $T$
- Execute and measure $t_T$ and $r_T$ for both versions

$L_{\text{min}}$  \hspace{2cm} \uparrow \hspace{2cm} \text{1. Not enough repetitions} \hspace{2cm} L_{\text{max}}$
Length of Tests (2)

Approach: Binary search

Repeat until constraints fulfilled:

- Generate a test $T$
- Execute and measure $t_T$ and $r_T$ for both versions

1. Not enough repetitions
2. Too short to measure

$L_{min}$ $L_{max}$
Approach: **Binary search**

Repeat until constraints fulfilled:

- Generate a test $T$
- Execute and measure $t_T$ and $r_T$ for both versions

---

$L_{min}$  
Too short to measure

1. Not enough repetitions  
3. Okay

$L_{max}$
Test Oracle

Does one version outperform the other?

1) Decide winner of each test

.. mean and confidence interval
Test Oracle

Does one version outperform the other?

1) Decide winner of each test

Statistically significant difference?

.. mean and confidence interval
Test Oracle

Does one version outperform the other?

1) Decide winner of each test

Execution times

Version 1 Version 2

.. mean and confidence interval

Statistically significant difference?
Difference above threshold?
Test Oracle

Does one version outperform the other?

2) Decide overall winner
Test Oracle

Does one version outperform the other?

2) Decide overall winner

- **Group tests by winner:**
  \[ \mathcal{T}_{V_1}, \mathcal{T}_{V_2}, \mathcal{T}_{None} \]

- **Report regression if**
  \[ |\mathcal{T}_{V_1}| > |\mathcal{T}_{V_2}| \quad \text{and} \quad |\mathcal{T}_{V_1}| > |\mathcal{T}_{None}| \]

- **Report improvement if**
  \[ |\mathcal{T}_{V_2}| > |\mathcal{T}_{V_1}| \quad \text{and} \quad |\mathcal{T}_{V_2}| > |\mathcal{T}_{None}| \]
SpeedGun: Overview

Automated performance regression testing for thread-safe classes

Version 1

Generation of concurrent performance tests

Version 2

Test execution and performance measurement

Test Oracle

Regression, improvement, or none
Evaluation

Does SpeedGun identify performance regressions and improvements?

Setup:
- 5 classes from 4 projects
- Full version history of 3 classes
- 113 pairs of classes

Baseline:
- Comments from developers
- Manual inspection
Results

113 pairs of classes

- No warning (96)
  - Expected difference is reported (11)
  - Expected difference is not reported (4)
  - Unexpected difference is reported (2)
Results

113 pairs of classes

No warning (96)

Expected difference is reported (11)

Expected difference is not reported (4)

Unexpected difference is reported (2)
Results

113 pairs of classes

No warning (96)

Expected difference is reported (11)

Expected difference is not reported (4)

Unexpected difference is reported (2)
Results

113 pairs of classes

No warning (96)

Expected difference is reported (11)

Expected difference is not reported (4)

Unexpected difference is reported (2)
Results

113 pairs of classes

No warning (96)

- Expected difference is reported (11)
- Expected difference is not reported (4)

Unexpected difference is reported (2)
# Examples

<table>
<thead>
<tr>
<th>Program</th>
<th>Change</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Groovy</td>
<td>Synchronize methods</td>
<td>0.92x</td>
</tr>
<tr>
<td>Groovy</td>
<td>Volatile instead of synchronized</td>
<td>1.50x</td>
</tr>
<tr>
<td>Collections</td>
<td>Fix correctness bug by adding synchronization</td>
<td>0.64x</td>
</tr>
<tr>
<td>Pool</td>
<td>Finer-grained locking to avoid deadlocks</td>
<td>1.52x</td>
</tr>
</tbody>
</table>

Intel Core i7 CPU, 8 threads
Related Work

Performance analysis and profiling
- Jovic2011
- Xu2012
- Grechanik2012
- Nistor2012
- Han2012
- Foo2010
- Yilmaz2005
- Chen2007

Test generation
- Visser2004
- Csallner2004
- Sen2005
- Godefroid2005
- Pacheco2007
- Ciupa2008
- Pradel2012
- Jin2010

Regression testing
- McCamant2003
- Burnim2009
- Zhang2011
- SpeedGun
- Jin2010
Conclusion

SpeedGun: Automated performance regression testing for thread-safe classes

- Generation of concurrent performance tests
- Systematically avoid pitfalls of measuring concurrent performance

A step towards **reliable and efficient** concurrent software
SpeedGun: Performance Regression Testing of Concurrent Classes

Michael Pradel\textsuperscript{1}, Markus Huggler\textsuperscript{2}, and Thomas R.Gross\textsuperscript{2}

\textsuperscript{1} University of California, Berkeley
\textsuperscript{2} ETH Zurich

I’m looking for students to join my group at TU Darmstadt!