Performance Regression Testing of Concurrent Classes

BY MICHAEL PRADEL, MARKUS HUGGLER, THOMAS R. GROSS

Goal

Automate for 2 versions of a concurrent class:

- Performance measurements
  - Reliable (no false positives or true negatives)
  - Meaningful (representative of real-world usage)

- Measurement evaluation

Algorithm overview: SpeedGun

1. Performance test generation

2. Performance measurements

3. Measurement evaluation

Test generation

Given a class to evaluate, generate…

- Sequential initialization (called **prefix**)
- Concurrent usage (called **suffixes**)

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


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

```
x.initialize();
x.getClassNode();
x.getProperties();
...
```

Test generation (suffixes)

Which methods to test?

- Common interface of classes under test
  - ⇒ Enables performance comparison
- Focus on methods with altered implementation
  - ⇒ Only performance differences are interesting
- Other methods potentially required, can’t be ignored

Test generation (suffixes)

Test length?

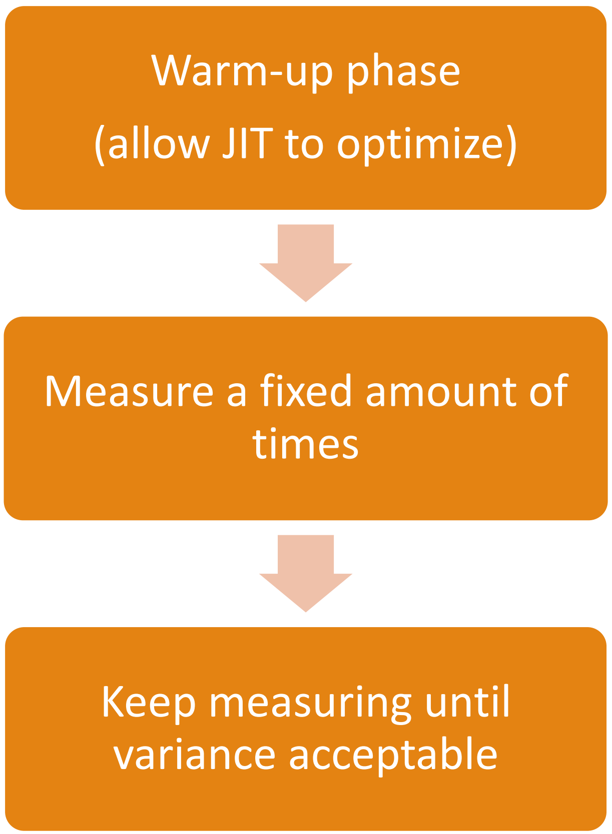
- OS-provided timers have limited accuracy
  - ⇒ No accurate measurement of short tests possible
- Too long tests have disadvantages, too
  - Expensive to generate: **O**(n^2), n := *length of call sequence*
  - Little additional measurement value

Solutions:

⇒ Binary search for n via trial and error

⇒ Call sequence of length $\sqrt{n}$ repeated $\sqrt{n}$ times

Performance measurements



Algorithm 2 Gather execution times of a test.

Input: Test T; Number of repetitions r_w and r_s for the warm-up phase and the steady-state phase, respectively

Output: Set $\mathcal{M}$ of execution times or *inconclusive*

```
runGarbageCollection()
repeat(T, r_w)                                   ▷ Warm-up phase
M ← ∅                                            ▷ Start of steady-state phase
repeat
    M ← M ∪ repeatAndMeasure(T, r_s)
until m_min measurements done
while σ(M) > M̄ · σ_stop do
    M ← M ∪ repeatAndMeasure(T, r_s)
    if |M| = m_max then
        if σ(M) ≤ M̄ · σ_acceptable then
            return M
        else
            return inconclusive
        end if
    end if
end while
return M                                         ▷ End of steady-state phase
```

Performance measurements

How to measure execution time of a test?

- Only **suffixes** are relevant
- May or may not measure **suffixes** individually, depending on use case

Algorithm 3 $repeatAndMeasure(T, r)$

Input: Test T; Number of repetitions r
Output: Execution time t

$t \leftarrow 0$
repeat
  Execute prefix of T
  Setup threads for suffixes of T
  $start \leftarrow currentTime()$ ▷ Start measurement
  for each thread **do**
    Execute a suffix of T
  end for
  $t \leftarrow t + currentTime() - start$ ▷ Stop measurement
  Clean up threads
until r repetitions done
return t

Measurement evaluation

Given execution times for a particular test…

- Compute mean and confidence interval
- Report performance difference if…
  - Confidence intervals don’t overlap
  - Difference between performances bigger than threshold

Measurement evaluation

Given evaluations of all tests concerning a class…

- Report performance difference if the majority of the tests show a performance difference in one direction

Real-world experiment setup

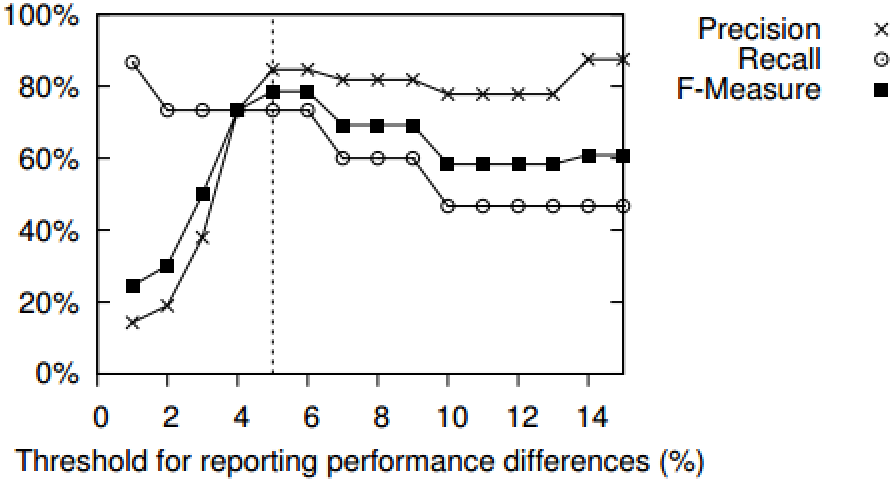
- Manual analysis of changes to Java code bases based on commit messages
- Comparison of **SpeedGun**'s results with manual analysis

Experimental results

ID	Code base	Class	Revision	Description	Performance: Baseline	SpeedGun: 8 thr.	SpeedGun: 64 thr.
(1)	Pool	GenericObjectPool	774007	Finer-grained locking to avoid deadlocks described in Issue 125	↗	↗ 1.52	↗ 1.57
(2)	Pool	GenericObjectPool	603449	Replace synchronized methods with volatile fields to address Issue 113	↗	↗ 1.30	↗ 1.38
(3)	Pool	GenericObjectPool	602773	Fix of a performance problem (Issue 93) by introducing more fine-grained locking	↗	↗ 2.09	↗ 2.20
(4)	Collections	StaticBucketMap	1076039	Fix of a correctness bug (Issue 334) by adding synchronization	↘	↘ 0.64	↘ 0.61
(5)	JodaTime	DateTime	v2.1	Newer version is reported to decrease performance over v1.5.2 due to additional synchronization (Issue 153)	↘	↘ 0.91	↘ 0.91
(6)	Groovy	ExpandoMetaClass	d3da3a44	Add synchronized blocks to fix correctness problem	↘	↘ 0.48	↘ 0.52
(7)	Groovy	ExpandoMetaClass	1c947d6b	Replace synchronized collections with project-internal concurrent collections	↗	↗ 1.08	↗ 1.29
(8)	Groovy	ExpandoMetaClass	2b09801e	Add synchronized block to fix correctness problem	↘	→	→
(9)	Groovy	ExpandoMetaClass	feff5190	Synchronize methods to fix correctness bug (Issue 2166)	↘	↘ 0.92	↘ 0.96
(10)	Groovy	ExpandoMetaClass	83629dc1	Patch to improve (sequential) performance	→	↗ 1.39	↗ 1.38
(11)	Groovy	ExpandoMetaClass	77822d4c	Replace project-internal concurrent collections with java.util.concurrent collections	↗	→	→
(12)	Groovy	ExpandoMetaClass	6e349cd9	Large patch without any obvious effects on performance	→	↘ 0.88	↘ 0.91
(13)	Groovy	ExpandoMetaClass	26fc2100	Replace synchronized methods with volatile field to fix performance bug (Issue 3557)	↗	↗ 1.50	↗ 1.42
(14)	Groovy	ExpandoMetaClass	d92c12ab	Replace volatile fields with synchronized methods to fix correctness problem	↘	↘ 0.95	↘ 0.95
(15)	Groovy	ExpandoMetaClass	48269129	Replace synchronized method with volatile fields to address performance problem (Issue 4182)	↗	↗ 1.03	→
(16)	Groovy	ExpandoMetaClass	cdc39843	Supposed performance improvement by replacing synchronized method with explicit locks	↗	→	→
(17)	Groovy	ExpandoMetaClass	d38da33c	Replace volatile field with synchronized method to fix correctness bug	↘	→	→

Experimental results

- **SpeedGun** coincides with majority of manual analysis
- Identified where expected performance improvements did not happen and vice versa
- Quality versus quantity of reports controlled by threshold



Conclusion

Results:

- Goal met! No obvious issues found in real-world experiment.

Limitations:

- Running time of several hours per class
- Automatic test generation may be too artificial for real world

Questions?

Thank you!