Lightweight Java Profiling with Partial Safepoints and Incremental Stack Tracing

> Peter Hofer
David Gnedt
Hanspeter Mössenböck

2 February 2015
Profiling

Where does my code spend its time?
Sampling with Safepoints

![Diagram showing sampling with safepoints]

- T1
- T2
- T3
- T4

- Safepointing
- In safepoint
- Resuming

Take stack traces
Partial Safepoints

Sample first $k$ threads that enter

… out of a set of $n$ threads of interest

optional: include waiting threads
Partial Safepoints and Self-Sampling

Each thread walks its own stack.
Redundant Stack Tracing Effort
Incremental Stack Tracing

Solution: decode only changed frames.
Implementation in HotSpot JVM (OpenJDK)

Challenges:

Frame layouts
  interpreter frames, compiled frames

Inlining
  multiple methods in one stack frame

Exceptions
  trace while unwinding the stack

Deoptimization and on-stack replacement
  frame is transformed, patching is lost, ...
Overhead Comparison

DaCapo and scalabench, $k = 4$ threads on quad-core CPU
Overhead Comparison

DaCapo and scalabench benchmarks

- **Safepoints**
- **Partial Incremental**
- **No Sampling**

1ms sampling interval
Accuracy

Compare to profile from instrumentation?

→ Stability and comparison to profile with safepoints

Method

Collect profiles of multiple executions of a workload

Merge into a single “average profile”

Analyze:

  compare individual profiles to avg profile

  compare avg. profile to avg. profile with safepoints
Stability

![Stability Graph]

- **Overlap**
- **Safepoints**
- **Partial Incremental**

<table>
<thead>
<tr>
<th>Project</th>
<th>Safepoints</th>
<th>Partial Incremental</th>
</tr>
</thead>
<tbody>
<tr>
<td>avorla</td>
<td></td>
<td></td>
</tr>
<tr>
<td>h2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>jython</td>
<td></td>
<td></td>
</tr>
<tr>
<td>luindex</td>
<td></td>
<td></td>
</tr>
<tr>
<td>lusearch</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pmd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sunflow</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tomcat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tradesoap</td>
<td></td>
<td></td>
</tr>
<tr>
<td>reactors</td>
<td></td>
<td></td>
</tr>
<tr>
<td>factorie</td>
<td></td>
<td></td>
</tr>
<tr>
<td>kiama</td>
<td></td>
<td></td>
</tr>
<tr>
<td>scalac</td>
<td></td>
<td></td>
</tr>
<tr>
<td>scaladoc</td>
<td></td>
<td></td>
</tr>
<tr>
<td>scalapi</td>
<td></td>
<td></td>
</tr>
<tr>
<td>scalatest</td>
<td></td>
<td></td>
</tr>
<tr>
<td>scalasp</td>
<td></td>
<td></td>
</tr>
<tr>
<td>specs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tnt</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Stability: hot methods

![Bar chart showing hot-edge coverage for different projects and methods. The chart compares safepoints and partial incremental methods.](image-url)
Comparison

![Bar chart showing overlap percentages for various projects including avroa, h2, jython, luindex, lusearch, pmid, sunflow, tomcat, tradesoap, kalan, actors, apparat, factorie, kiama, scalac, scaladoc, scalap, scalaijorm, scala-xp, specs, tnt.](chart.png)
Comparison: hot methods
Conclusion

Techniques

- Partial Safepoints
- Self-Sampling
- Incremental Stack Tracing

Low overhead

- without hardware or operating system support

Short and predictable pause times

Accuracy unaffected
Questions