Analysis of Overhead in Dynamic Java Performance Monitoring

Vojtěch Horký, Jaroslav Kotrč, Peter Libič and Petr Tůma

Charles University in Prague
Context: Dynamic Monitoring of Production Systems
Dynamic Monitoring of Production Systems

Measurement probes are active only when needed, measuring everything all the time might not be practical.
Measurement probes are active only when needed, measuring everything all the time might not be practical.
Dynamic Monitoring of Production Systems

Measurement probes are active only when needed, measuring everything all the time might not be practical.

We are interested in performance of this function.
Dynamic Monitoring of Production Systems

Measurement probes are active only when needed, measuring everything all the time might not be practical.
Dynamic Monitoring of Production Systems

Measurement probes are active only when needed, measuring everything all the time might not be practical.

Code is dynamically instrumented when measuring.
Dynamic Monitoring of Production Systems

Measurement probes are active only when needed, measuring everything all the time might not be practical.

Once enough data is collected, probes are removed.
Issues of Dynamic Monitoring

In managed environments, code is compiled at run-time; probe insertion (removal) causes recompilation. Monitored application can thus behave differently.
Issues of Dynamic Monitoring

In managed environments, code is compiled at run-time; probe insertion (removal) causes recompilation. Monitored application can thus behave differently.

Interesting Questions

How do the code manipulations affect the application?

What is the overhead of such probe?

Is the observed performance representative?

Is there zero overhead once the probe is removed?
Experiment Setup
Experiment Coordination
<table>
<thead>
<tr>
<th>Two Measurement Infrastructures</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Self-measurement</strong></td>
</tr>
<tr>
<td><strong>Dynamic monitoring</strong></td>
</tr>
</tbody>
</table>
## Two Measurement Infrastructures

<table>
<thead>
<tr>
<th>Performance</th>
<th>Self-measurement</th>
<th>Dynamic monitoring</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Baseline</td>
<td>“Observed”</td>
</tr>
</tbody>
</table>
## Two Measurement Infrastructures

<table>
<thead>
<tr>
<th></th>
<th>Self-measurement</th>
<th>Dynamic monitoring</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Performance</strong></td>
<td>Baseline</td>
<td>“Observed”</td>
</tr>
<tr>
<td><strong>Location</strong></td>
<td>Method entry and exit points (both)</td>
<td></td>
</tr>
</tbody>
</table>
## Two Measurement Infrastructures

<table>
<thead>
<tr>
<th></th>
<th>Self-measurement</th>
<th>Dynamic monitoring</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Performance</strong></td>
<td>Baseline</td>
<td>“Observed”</td>
</tr>
<tr>
<td><strong>Location</strong></td>
<td>Method entry and exit points (both)</td>
<td></td>
</tr>
<tr>
<td><strong>Instrumentation</strong></td>
<td>Static</td>
<td>Dynamic (run-time)</td>
</tr>
</tbody>
</table>
## Two Measurement Infrastructures

<table>
<thead>
<tr>
<th></th>
<th>Self-measurement</th>
<th>Dynamic monitoring</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Performance</strong></td>
<td>Baseline</td>
<td>“Observed”</td>
</tr>
<tr>
<td><strong>Location</strong></td>
<td>Method entry and exit points (both)</td>
<td>Dynamic (run-time)</td>
</tr>
<tr>
<td><strong>Instrumentation</strong></td>
<td>Static</td>
<td>On demand</td>
</tr>
<tr>
<td><strong>Data collection</strong></td>
<td>Continuous</td>
<td></td>
</tr>
</tbody>
</table>

Department of Distributed and Dependable Systems
## Two Measurement Infrastructures

<table>
<thead>
<tr>
<th></th>
<th>Self-measurement</th>
<th>Dynamic monitoring</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Performance</strong></td>
<td>Baseline</td>
<td>“Observed”</td>
</tr>
<tr>
<td><strong>Location</strong></td>
<td>Method entry and exit points (both)</td>
<td></td>
</tr>
<tr>
<td><strong>Instrumentation</strong></td>
<td>Static</td>
<td>Dynamic (run-time)</td>
</tr>
<tr>
<td><strong>Data collection</strong></td>
<td>Continuous</td>
<td>On demand</td>
</tr>
<tr>
<td><strong>Implementation</strong></td>
<td>Native method (in C)</td>
<td>Pure Java</td>
</tr>
</tbody>
</table>
## Two Measurement Infrastructures

<table>
<thead>
<tr>
<th></th>
<th>Self-measurement</th>
<th>Dynamic monitoring</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Performance</strong></td>
<td>Baseline</td>
<td>“Observed”</td>
</tr>
<tr>
<td><strong>Location</strong></td>
<td>Method entry and exit points (both)</td>
<td></td>
</tr>
<tr>
<td><strong>Instrumentation</strong></td>
<td>Static</td>
<td>Dynamic (run-time)</td>
</tr>
<tr>
<td><strong>Data collection</strong></td>
<td>Continuous</td>
<td>On demand</td>
</tr>
<tr>
<td><strong>Implementation</strong></td>
<td>Native method (in C)</td>
<td>Pure Java</td>
</tr>
</tbody>
</table>

![Diagram showing dynamic and static probes]

- **Dynamic probe**: measurement at run-time
- **Static probe**: measurement at compile-time
- **Function of interest**: specific aspect to measure
Experiment Process
Experiment Process

run for some time
Experiment Process

pick random method

run for some time
Experiment Process

run for some time

pick random method

insert dynamic probe

Dynamic monitoring
Experiment Process

- Run for some time
- Pick random method
- Insert dynamic probe
- Dynamic monitoring
- Run for some time
Experiment Process

1. **run for some time**
2. **pick random method**
3. **insert dynamic probe**
4. **Dynamic monitoring**
   - **run for some time**
   - **dump from static probes**
   - **dump from dynamic probe**
Experiment Process

- **run for some time**
  - **pick random method**
  - **insert dynamic probe**
    - **run for some time**
      - **dump from static probes**
      - **dump from dynamic probe**
        - What is the **observed** performance?
Experiment Process

- run for some time
  - pick random method
  - insert dynamic probe
- How fast it runs **with** dynamic monitoring?
  - dump from static probes
  - dump from dynamic probe
- run for some time
- What is the **observed** performance?
Experiment Process

- **run for some time**
- **pick random method**
- **insert dynamic probe**
- **dump from static probes**
- **remove dynamic probe**

**Dynamic monitoring**
- **run for some time**
- **How fast it runs with dynamic monitoring?**
- **dump from dynamic probe**
- **What is the observed performance?**
Experiment Process

1. run for some time
2. pick random method
3. insert dynamic probe
4. dump from static probes
5. remove dynamic probe
6. run for some time
7. dynamic monitoring
   - How fast it runs with dynamic monitoring?
   - dump from dynamic probe
   - What is the observed performance?
Experiment Process

- **pick random method**
- **run for some time**
- **dump from static probes**
- **run for some time**

**Dynamic monitoring**

- **insert dynamic probe**
- **run for some time**

- **How fast it runs with dynamic monitoring?**

- **dump from static probes**
- **remove dynamic probe**

- **dump from dynamic probe**

- **What is the observed performance?**
Experiment Process

1. Pick random method
2. Run for some time
   - Dump from static probes
   - How fast it runs without dynamic monitoring?
3. Insert dynamic probe
4. Run for some time
   - Dump from dynamic probe
   - What is the observed performance?
5. Dump from static probes
6. Remove dynamic probe
   - How fast it runs with dynamic monitoring?
Experiment Process

- pick random method
- run for some time
- insert dynamic probe
- dump from static probes
- dump from dynamic probe
- run for some time
- remove dynamic probe
- dump from static probes
- What is the observed performance?

How fast it runs without dynamic monitoring?

How fast it runs with dynamic monitoring?
Platform and Application Details
Platform and Application Details

- Hardware: 32 CPUs, 2 NUMA nodes, 48G RAM.
Platform and Application Details

- Hardware: 32 CPUs, 2 NUMA nodes, 48G RAM.
- SPECjbb2015 augmented with static probes.
  - Fixed request rate 4000 reqs/s.
    (Close to maximum with static probes on our hardware.)
- Over 1200 monitored methods.
  - Business code of the benchmark.
  - Practically all methods called frequently enough.
  - About one minute of dynamic monitoring per method.
Platform and Application Details

- Hardware: 32 CPUs, 2 NUMA nodes, 48G RAM.

- SPECjbb2015 augmented with static probes.
  - Fixed request rate 4000 reqs/s.
    (Close to maximum with static probes on our hardware.)

- Over 1200 monitored methods.
  - Business code of the benchmark.
  - Practically all methods called frequently enough.
  - About one minute of dynamic monitoring per method.

- Several TBs of raw data per week of run-time.
Results
Overall Overhead of Dynamic Monitoring
Overall Overhead of Dynamic Monitoring

Record CPU utilization with dynamic monitoring... and without it.
Overall Overhead of Dynamic Monitoring

- Without dynamic monitoring
- With dynamic monitoring

CPU utilization [%]

Frequency
Overall Overhead of Dynamic Monitoring

Measuring one method (even a hot one) at a time brings no significant overhead.
Time Needed for Just-in-time Recompilation
Time Needed for Just-in-time Recompilation

- Dynamic monitoring
  - run for some time
  - dump from dynamic probe

- Record Just-in-time compiler events here ...
  - ... and here.
  - run for some time

- pick random method
  - insert dynamic probe
  - dump from static probes

- run for some time
  - dump from static probes
  - remove dynamic probe
  - dump from dynamic probe
Time Needed for Just-in-time Recompilation

Recompilation duration [s]
(waited for a minute without JIT activity)

Frequency

Instrumentation
(probe inserted)

Deinstrumentation
(probe removed)
JIT compiler typically needs at least 30 s to finish recompilations after probe insertion (removal).
Accuracy of Collected Data
Accuracy of Collected Data

Ratio between observed and baseline performance.
Accuracy of Collected Data

Ratio between times reported by static and dynamic probes

Method execution time (static probe) [s]
Accuracy of Collected Data

Ratio between times reported by static and dynamic probes

Method execution time (static probe) [µs]

0 10 20 30 40

1.0 2.0 3.0 4.0
Accuracy of Collected Data

Ratio between times reported by static and dynamic probes

Interpretation of numbers from dynamic monitoring:

<table>
<thead>
<tr>
<th>Observed</th>
<th>Actual</th>
</tr>
</thead>
<tbody>
<tr>
<td>50 µs</td>
<td>45 µs – 50 µs</td>
</tr>
<tr>
<td>20 µs</td>
<td>10 µs – 20 µs</td>
</tr>
<tr>
<td>2 µs</td>
<td>½ µs – 2 µs</td>
</tr>
</tbody>
</table>
Impact of Dynamic Monitoring
Impact of Dynamic Monitoring

- Dynamic monitoring
  - run for some time
  - insert dynamic probe
  - dump from static probes
  - dump from dynamic probe
  - run for some time
  - remove dynamic probe

Ratio of baseline performance with and without dynamic monitoring

- pick random method
  - run for some time
  - dump from static probes
  - run for some time
Impact of Dynamic Monitoring

Static probes: ratio of mean execution times during and after dynamic instrumentation

Faster when being monitored

Slower when being monitored

⇒⇒
Impact of Dynamic Monitoring

Static probes: ratio of mean execution times during and after dynamic instrumentation

Faster when being monitored

Slower when being monitored
Impact of Dynamic Monitoring

Dynamic monitoring can observe shorter times (as if the probes speeded-up the application).

Static probes: ratio of mean execution times during and after dynamic instrumentation
Conclusion
Analysis of Overhead in Dynamic Java Performance Monitoring

We evaluated how dynamic monitoring affects a running application and what is the accuracy of the obtained data.
Analysis of Overhead in Dynamic Java Performance Monitoring

We evaluated how dynamic monitoring affects a running application and what is the accuracy of the obtained data.

Rules of thumb coming from our experiment . . .
- Measuring one method at a time does not change CPU utilization.
- At least 30 s are needed for (JIT) recompilation.
- If the reported time is 30 µs . . .
  . . . the actual duration is between 20 µs and 40 µs
  (durations of at least 100 µs are more “trustworthy”, though).
Analysis of Overhead in Dynamic Java Performance Monitoring

We evaluated how dynamic monitoring affects a running application and what is the accuracy of the obtained data.

Rules of thumb coming from our experiment . . .

- Measuring one method at a time does not change CPU utilization.
- At least 30 s are needed for (JIT) recompilation.
- If the reported time is 30 µs . . .
  
  ...the actual duration is between 20 µs and 40 µs
  (durations of at least 100 µs are more “trustworthy”, though).

http://d3s.mff.cuni.cz/resources/icpe2016
Analysis of Overhead in Dynamic Java Performance Monitoring

We evaluated how dynamic monitoring affects a running application and what is the accuracy of the obtained data.

Rules of thumb coming from our experiment . . .

– Measuring one method at a time does not change CPU utilization.
– At least 30 s are needed for (JIT) recompilation.
– If the reported time is 30 µs . . .
  . . . the actual duration is between 20 µs and 40 µs
  (durations of at least 100 µs are more “trustworthy”, though).

http://d3s.mff.cuni.cz/resources/icpe2016

Thank You!