Kernels and Tracing

Lecture 2, Part 2: The Probe Effect
Prof. Robert N. M. Watson
2022-2023
The **Probe Effect**

• The **probe effect** is the unintended alteration of system behaviour that arises from measurement
  • Software instrumentation is **active**: execution is changed
• DTrace minimises probe effect when unused...
  • ... but has a very significant impact when it is used
  • Disproportionate effect on probed events
• Potential perturbations:
  • Speed relative to other cores (e.g., lock hold times)
  • Speed relative to external events (e.g., timer ticks)
  • Microarchitectural effects (e.g., cache, branch predictor)
Probe effect example: dd(1) execution time

• Simple (naïve) microbenchmark – dd(1)
  • dd copies blocks from input to output
  • Copy 10M buffer from /dev/zero to /dev/null
  • (“Do nothing .. But do it slowly”)
  • Execution time measured with /usr/bin/time
  • Workload chosen to illustrate high overhead

# dd if=/dev/zero of=/dev/null bs=10m count=1 status=none

• Simultaneously, run various DTrace scripts
  • Compare resulting execution times using ministat
  • Difference is probe effect (+/- measurement error)
Probe effect 1: memory allocation

- Using the `dtmalloc` provider, count kernel memory allocations:

```c
dtmalloc:::
{ @count = count(); }
```

<table>
<thead>
<tr>
<th></th>
<th>Min</th>
<th>Max</th>
<th>Median</th>
<th>Avg</th>
<th>Stddev</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>0.2</td>
<td>0.22</td>
<td>0.21</td>
<td>0.20818182</td>
<td>0.0060302269</td>
</tr>
<tr>
<td>+</td>
<td>0.2</td>
<td>0.22</td>
<td>0.21</td>
<td>0.21272727</td>
<td>0.0064666979</td>
</tr>
</tbody>
</table>

No difference proven at 95.0% confidence

- No statistically significant overhead at 95% confidence level
Probe effect 2: locking

- Using the `lockstat` provider, track kernel lock acquire, release:

  ```
  lockstat:::
  { @count = count(); }
  ```

  ```
  x no-dtrace
  + lockstat-count
  +---------------------------------------------------+
  |   x   |   +   |
  |   x   |   +   |
  |   x   |   +   |
  |  x  x  |   +   |
  |  x  x  |   +   |
  |  x  x  x |   +   |
  |   |_A_1|   + |
  +---------------------------------------------------+
  ```

<table>
<thead>
<tr>
<th>N</th>
<th>Min</th>
<th>Max</th>
<th>Median</th>
<th>Avg</th>
<th>Stddev</th>
</tr>
</thead>
<tbody>
<tr>
<td>x 11</td>
<td>0.2</td>
<td>0.22</td>
<td>0.21</td>
<td>0.20818182</td>
<td>0.0060302269</td>
</tr>
<tr>
<td>+ 11</td>
<td>0.42</td>
<td>0.44</td>
<td>0.44</td>
<td>0.43454545</td>
<td>0.0068755165</td>
</tr>
</tbody>
</table>

  Difference at 95.0% confidence
  0.226364 +/- 0.00575196
  108.734% +/- 2.76295%
  (Student's t, pooled s = 0.0064667)

- **109% overhead** – 170K locking operations vs. 6 `malloc()` calls!
Probe effect 3: limiting to dd(1)?

- Limit the action to processes with the name dd:

```c
lockstat::: /execname == "dd"/
{ @count = count(); }
```

- Well, crumbs. Now **168% overhead**!
Probe effect 4: stack traces

- Gather more locking information in action – capture call stacks:

  ```
  lockstat::: { @stacks[stack()] = count(); }
  lockstat::: /execname == "dd"/ { @stacks[stack()] = count(); }
  ```

  x no-dtrace
  + lockstat-stack
  * lockstat-stack-dd

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>Min</td>
<td>Max</td>
<td>Median</td>
</tr>
<tr>
<td>---</td>
<td>-----</td>
<td>-----</td>
<td>--------</td>
</tr>
<tr>
<td>x</td>
<td>11</td>
<td>0.2</td>
<td>0.22</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+</td>
<td>11</td>
<td>1.38</td>
<td>1.57</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>*</td>
<td>11</td>
<td>1.5</td>
<td>1.55</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
What does this mean for us?

• Always think about the potential role of the probe effect when instrumenting a workload
  • E.g., avoid benchmarking while running DTrace ...
  • ... unless measuring or accounting for the probe effect

• Traced applications may behave (very) differently
  • E.g., more timer ticks will fire, affecting thread inverleaving
  • E.g., I/O will “seem faster” relative to computation, as latter may slow down due to probe effect

• Performance overheads may be disproportionate
  • E.g., if you instrument one way of doing things, but not another, and workloads have a different functional footprint

• Consider ways to decide if an analysis is representative
  • E.g., are the performance inflection points consistent even if absolute performance is lower?