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

```bash
# 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(); }
  ```

\[
\begin{array}{|c|c|c|c|}
\hline
N & Min & Max & Median \\
\hline
x & 11 & 0.2 & 0.22 & 0.21 \\
\hline
+ & 11 & 0.2 & 0.22 & 0.21 \\
\hline
\end{array}
\]

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:

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

<table>
<thead>
<tr>
<th>x no-dtrace</th>
<th>+ lockstat-count</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>+</td>
</tr>
<tr>
<td>x</td>
<td>+</td>
</tr>
<tr>
<td>x</td>
<td>+</td>
</tr>
<tr>
<td>x</td>
<td>+</td>
</tr>
<tr>
<td>x x</td>
<td>+</td>
</tr>
<tr>
<td>x x</td>
<td>+</td>
</tr>
<tr>
<td>x x x</td>
<td>+</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<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:

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

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

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

```plaintext
lockstat::: { @stacks[stack()] = count(); }
lockstat::: /execname == "dd"/ { @stacks[stack()] = 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>1.38</td>
<td>1.57</td>
<td>1.44</td>
<td>1.4618182</td>
<td>0.058449668</td>
</tr>
<tr>
<td>*</td>
<td>1.25364 +/- 0.0369572</td>
<td>602.183% +/- 17.7524%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>*</td>
<td>1.30455 +/- 0.00970671</td>
<td>626.638% +/- 4.66261%</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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