From the trenches: Lessons learned from the JBoss performance team

Ståle W. Pedersen
Senior Principal Software Engineer
29 June 2016
Agenda

Lessons of what we've learned….

- Methodology
- Tools
- Profiling
- Benchmarking
- Road to EAP7 from a performance perspective
Methodology

Performance optimization Methodology
Methodology

What

• What prevents my application from running faster?
  – Monitoring
Methodology

What -> Where

• What prevents my application from running faster?
  – Monitoring

• Where does it hide?
  – Profiling
Methodology

What -> Where -> How

• What prevents my application from running faster?
  – Monitoring

• Where does it hide?
  – Profiling

• How can we improve performance?
  – tuning/optimizing
Methodology

Top down approach

• System Level
  – Disks, CPU, Memory, Network, ....
Methodology

Top down approach

• System Level
  – Disks, CPU, Memory, Network, ....
• JVM Level
  – Heap, GC, JIT, Classloading, ....
Methodology

Top down approach

• System Level
  – Disks, CPU, Memory, Network, ....
• JVM Level
  – Heap, GC, JIT, Classloading, ....
• Application Level
  – APIs, Algorithms, Threading, ....
Tools
Tools

Tool types

- Observability, safe, depending on overhead
- Benchmarking, load testing. Difficult to do in a production environment
- Tuning, changes could hurt performance, perhaps not now or at the current load, but later
System summary, list processes or threads

- Can miss processes that do not last long enough to be caught
- Will most often consume a fair bit of CPU
- Default summary CPU usage, use “1” to get a better view of cores
dmesg

Print kernel ring buffer

- Display system messages
- An error tool that can give info why an application is slow/dies
- Eg: OOM issues, filehandles, TCP errors, ++
free

Memory usage

[benchuser@benchserver2 ~]$ free -m

```
Mem: 255879  69845  183100  9  2933  185504
Swap: 2047 0 2047
```

• Give a quick overview over memory usage
• Virtual page, and block devices I/O cache
vmstat

Virtual memory stat

<table>
<thead>
<tr>
<th>procs</th>
<th>memory</th>
<th>swap</th>
<th>io</th>
<th>system</th>
<th>cpu</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>b</td>
<td>swpd</td>
<td>free</td>
<td>buff</td>
<td>cache</td>
</tr>
<tr>
<td>33</td>
<td>0</td>
<td>0</td>
<td>187492384</td>
<td>1223708</td>
<td>1783148</td>
</tr>
<tr>
<td>66</td>
<td>0</td>
<td>0</td>
<td>187493088</td>
<td>1223716</td>
<td>1782948</td>
</tr>
<tr>
<td>52</td>
<td>0</td>
<td>0</td>
<td>187492304</td>
<td>1223724</td>
<td>1782916</td>
</tr>
<tr>
<td>29</td>
<td>0</td>
<td>0</td>
<td>187491952</td>
<td>1223724</td>
<td>1782956</td>
</tr>
<tr>
<td>37</td>
<td>0</td>
<td>0</td>
<td>187491344</td>
<td>1223724</td>
<td>1783152</td>
</tr>
<tr>
<td>54</td>
<td>0</td>
<td>0</td>
<td>187494144</td>
<td>1223724</td>
<td>1782992</td>
</tr>
<tr>
<td>38</td>
<td>0</td>
<td>0</td>
<td>187493856</td>
<td>1223732</td>
<td>1783016</td>
</tr>
</tbody>
</table>

- **r** - number of processes running (waiting to run), should be < num cpus
- **b** - number of processes sleeping
- **free** – idle memory, should be high
- **si, so** – memory swapped to/form disk, should be 0
- **cpu**, percentages of total CPU time
  - user time, system time, idle, waiting for I/O and stolen time
  - system time used for I/O
mpstat

Processors statistics

<table>
<thead>
<tr>
<th>Time</th>
<th>CPU</th>
<th>%usr</th>
<th>%nice</th>
<th>%sys</th>
<th>%iowait</th>
<th>%irq</th>
<th>%soft</th>
<th>%steal</th>
<th>%guest</th>
<th>%gnice</th>
<th>%idle</th>
</tr>
</thead>
<tbody>
<tr>
<td>11:52:54 PM</td>
<td>CPU</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11:52:55 PM</td>
<td>all</td>
<td>66.10</td>
<td>0.00</td>
<td>7.45</td>
<td>0.00</td>
<td>4.17</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>22.28</td>
</tr>
<tr>
<td>11:52:55 PM</td>
<td>0</td>
<td>75.00</td>
<td>0.00</td>
<td>9.00</td>
<td>0.00</td>
<td>12.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>4.00</td>
</tr>
<tr>
<td>11:52:55 PM</td>
<td>1</td>
<td>77.78</td>
<td>0.00</td>
<td>6.06</td>
<td>0.00</td>
<td>12.12</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>4.04</td>
</tr>
<tr>
<td>11:52:55 PM</td>
<td>2</td>
<td>79.00</td>
<td>0.00</td>
<td>5.00</td>
<td>0.00</td>
<td>13.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>3.00</td>
</tr>
<tr>
<td>11:52:55 PM</td>
<td>3</td>
<td>78.00</td>
<td>0.00</td>
<td>6.00</td>
<td>0.00</td>
<td>10.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>6.00</td>
</tr>
<tr>
<td>11:52:55 PM</td>
<td>4</td>
<td>79.17</td>
<td>0.00</td>
<td>5.21</td>
<td>0.00</td>
<td>9.38</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>6.25</td>
</tr>
<tr>
<td>11:52:55 PM</td>
<td>5</td>
<td>77.00</td>
<td>0.00</td>
<td>7.00</td>
<td>0.00</td>
<td>12.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>4.00</td>
</tr>
<tr>
<td>11:52:55 PM</td>
<td>6</td>
<td>79.80</td>
<td>0.00</td>
<td>7.07</td>
<td>0.00</td>
<td>8.08</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>5.05</td>
</tr>
<tr>
<td>11:52:55 PM</td>
<td>7</td>
<td>77.55</td>
<td>0.00</td>
<td>6.12</td>
<td>0.00</td>
<td>13.27</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>3.06</td>
</tr>
<tr>
<td>11:52:55 PM</td>
<td>8</td>
<td>58.76</td>
<td>0.00</td>
<td>8.25</td>
<td>0.00</td>
<td>1.03</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>31.96</td>
</tr>
<tr>
<td>11:52:55 PM</td>
<td>9</td>
<td>59.00</td>
<td>0.00</td>
<td>8.00</td>
<td>0.00</td>
<td>2.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>31.00</td>
</tr>
</tbody>
</table>

• Gives a good overview of CPU time
• Comparable to top (with 1 pressed to show all CPU's), but use less CPU
sar

System activity information

<table>
<thead>
<tr>
<th>Time</th>
<th>Interface</th>
<th>rxpck/s</th>
<th>txpck/s</th>
<th>rxkB/s</th>
<th>txkB/s</th>
<th>rxcmp/s</th>
<th>txcmp/s</th>
<th>rxmcst/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>11:53:55 PM</td>
<td>IFACE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11:53:56 PM</td>
<td>eno1</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>11:53:56 PM</td>
<td>eno2</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>11:53:56 PM</td>
<td>lo</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>11:53:56 PM</td>
<td>ens1f0</td>
<td>124928.00</td>
<td>115218.00</td>
<td>114429.66</td>
<td>383212.22</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>11:53:56 PM</td>
<td>ens4f0</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>11:53:56 PM</td>
<td>ens5f0</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>11:53:56 PM</td>
<td>ens1f1</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>11:53:56 PM</td>
<td>ens4f1</td>
<td>54326.00</td>
<td>54785.00</td>
<td>15473.99</td>
<td>12272.61</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>11:53:56 PM</td>
<td>ens5f1</td>
<td>3.00</td>
<td>1.00</td>
<td>0.19</td>
<td>0.19</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

- Check network throughput
- rxpck/s, txpck/s – total number of packets received/transmitted per second
- %ifutil – utilization percentage of the network interface
- Use EDEV for statistics on failures the network devices are reporting
System activity information

- Reports TCP network traffic and errors
- active/s – number of local initiated TCP connections per second
- passive/s – number of remote initiated TCP connections per second
- retrans/s – number of segments retransmitted per second
## iostat

Monitor system I/O devices

<table>
<thead>
<tr>
<th>Device</th>
<th>tps</th>
<th>kB_read/s</th>
<th>kB_wrtn/s</th>
<th>kB_read</th>
<th>kB_wrtn</th>
</tr>
</thead>
<tbody>
<tr>
<td>sdb</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>sdc</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>sda</td>
<td>2.00</td>
<td>0.00</td>
<td>24.00</td>
<td>0</td>
<td>24</td>
</tr>
<tr>
<td>fiob</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>dm-0</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>dm-1</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>dm-2</td>
<td>6.00</td>
<td>0.00</td>
<td>24.00</td>
<td>0</td>
<td>24</td>
</tr>
<tr>
<td>fioa</td>
<td>1910.00</td>
<td>0.00</td>
<td>4404.00</td>
<td>0</td>
<td>4404</td>
</tr>
</tbody>
</table>

avg-cpu: %user  %nice  %system  %iowait  %steal  %idle
65.72  0.00  11.76  0.00  0.00  22.52
Java Tools

Observing Java applications
Java Tools

Observing Java applications

• Not as many tools out of the box
Java Tools

Observing Java applications

• Not as many tools out of the box
  – but they are powerful

• jstack, jstat, jcmd, jps, jmap, jconsole, jvisualvm
jstack

Prints Java thread stack traces for a Java process

```
"httpWorker Accept" #91 prio=5 os_prio=0 tid=0x00002b69a81f0000 nid=0x3b02 runnable [0x00002b6a7110b000]
  java.lang.Thread.State: RUNNABLE
    at sun.nio.ch.EPollArrayWrapper.epollWait(Native Method)
    at sun.nio.ch.SelectorImpl.lockAndDoSelect(SelectorImpl.java:86)
    - locked <0x00000004c25dc400> (a sun.nio.ch.Util$2)
    - locked <0x000000004c25dc3f8> (a java.util.Collections$UnmodifiableSet)
    - locked <0x000000004c25dc020> (a sun.nio.ch.EPollSelectorImpl)
    at sun.nio.ch.SelectorImpl.select(SelectorImpl.java:97)
    at sun.nio.ch.SelectorImpl.select(SelectorImpl.java:101)
    at org.xnio.nio.WorkerThread.run(WorkerThread.java:509)
```

• Useful to quickly look at the current state of the threads
jmap

Prints Java shared object memory maps or heap memory

<table>
<thead>
<tr>
<th>Timestamp</th>
<th>S0</th>
<th>S1</th>
<th>E</th>
<th>O</th>
<th>M</th>
<th>CCS</th>
<th>YGC</th>
<th>YGCT</th>
<th>FGC</th>
<th>FGCT</th>
<th>GCT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1105.8</td>
<td>90.96</td>
<td>0.00</td>
<td>90.63</td>
<td>40.72</td>
<td>93.13</td>
<td>85.62</td>
<td>530</td>
<td>19.312</td>
<td>0</td>
<td>0.000</td>
<td>19.312</td>
</tr>
<tr>
<td>1106.8</td>
<td>0.00</td>
<td>77.27</td>
<td>40.72</td>
<td>40.76</td>
<td>93.13</td>
<td>85.62</td>
<td>531</td>
<td>19.345</td>
<td>0</td>
<td>0.000</td>
<td>19.345</td>
</tr>
<tr>
<td>1107.8</td>
<td>0.00</td>
<td>77.27</td>
<td>88.27</td>
<td>40.76</td>
<td>93.13</td>
<td>85.62</td>
<td>531</td>
<td>19.345</td>
<td>0</td>
<td>0.000</td>
<td>19.345</td>
</tr>
<tr>
<td>1108.8</td>
<td>83.28</td>
<td>0.00</td>
<td>37.50</td>
<td>40.80</td>
<td>93.13</td>
<td>85.62</td>
<td>532</td>
<td>19.383</td>
<td>0</td>
<td>0.000</td>
<td>19.383</td>
</tr>
<tr>
<td>1109.8</td>
<td>83.28</td>
<td>0.00</td>
<td>85.29</td>
<td>40.80</td>
<td>93.13</td>
<td>85.62</td>
<td>532</td>
<td>19.383</td>
<td>0</td>
<td>0.000</td>
<td>19.383</td>
</tr>
<tr>
<td>1110.8</td>
<td>0.00</td>
<td>84.80</td>
<td>33.76</td>
<td>40.83</td>
<td>93.13</td>
<td>85.62</td>
<td>533</td>
<td>19.417</td>
<td>0</td>
<td>0.000</td>
<td>19.417</td>
</tr>
<tr>
<td>1111.8</td>
<td>0.00</td>
<td>84.80</td>
<td>83.67</td>
<td>40.83</td>
<td>93.13</td>
<td>85.62</td>
<td>533</td>
<td>19.417</td>
<td>0</td>
<td>0.000</td>
<td>19.417</td>
</tr>
<tr>
<td>1112.8</td>
<td>76.89</td>
<td>0.00</td>
<td>31.00</td>
<td>40.87</td>
<td>93.13</td>
<td>85.62</td>
<td>534</td>
<td>19.447</td>
<td>0</td>
<td>0.000</td>
<td>19.447</td>
</tr>
<tr>
<td>1113.8</td>
<td>76.89</td>
<td>0.00</td>
<td>82.40</td>
<td>40.87</td>
<td>93.13</td>
<td>85.62</td>
<td>534</td>
<td>19.447</td>
<td>0</td>
<td>0.000</td>
<td>19.447</td>
</tr>
<tr>
<td>1114.8</td>
<td>0.00</td>
<td>83.57</td>
<td>29.54</td>
<td>40.92</td>
<td>93.13</td>
<td>85.62</td>
<td>535</td>
<td>19.476</td>
<td>0</td>
<td>0.000</td>
<td>19.476</td>
</tr>
<tr>
<td>1115.8</td>
<td>0.00</td>
<td>83.57</td>
<td>82.20</td>
<td>40.92</td>
<td>93.13</td>
<td>85.62</td>
<td>535</td>
<td>19.476</td>
<td>0</td>
<td>0.000</td>
<td>19.476</td>
</tr>
<tr>
<td>1116.8</td>
<td>86.03</td>
<td>0.00</td>
<td>31.61</td>
<td>40.97</td>
<td>93.13</td>
<td>85.62</td>
<td>536</td>
<td>19.507</td>
<td>0</td>
<td>0.000</td>
<td>19.507</td>
</tr>
</tbody>
</table>

- Several different options
  - histo[:live] – Prints a histogram of the heap, list classes, number of objects, size
  - heap – Prints information regarding heap status and usage
  - dump:<dump-options> – Dump java heap
  - No options will print shared object mappings
jstat

Monitor JVM statistics; gc, compilations, classes

<table>
<thead>
<tr>
<th>Timestamp</th>
<th>S0</th>
<th>S1</th>
<th>E</th>
<th>O</th>
<th>M</th>
<th>CCS</th>
<th>YGC</th>
<th>YGCT</th>
<th>FGC</th>
<th>FGCT</th>
<th>GCT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1165.8</td>
<td>90.96</td>
<td>0.00</td>
<td>96.63</td>
<td>48.72</td>
<td>93.13</td>
<td>85.62</td>
<td>530</td>
<td>19.312</td>
<td>0</td>
<td>0.000</td>
<td>19.312</td>
</tr>
<tr>
<td>1166.8</td>
<td>0.00</td>
<td>77.27</td>
<td>40.72</td>
<td>40.76</td>
<td>93.13</td>
<td>85.62</td>
<td>531</td>
<td>19.345</td>
<td>0</td>
<td>0.000</td>
<td>19.345</td>
</tr>
<tr>
<td>1167.8</td>
<td>0.00</td>
<td>77.27</td>
<td>88.27</td>
<td>40.76</td>
<td>93.13</td>
<td>85.62</td>
<td>531</td>
<td>19.345</td>
<td>0</td>
<td>0.000</td>
<td>19.345</td>
</tr>
<tr>
<td>1168.8</td>
<td>83.28</td>
<td>0.00</td>
<td>37.50</td>
<td>40.80</td>
<td>93.13</td>
<td>85.62</td>
<td>532</td>
<td>19.383</td>
<td>0</td>
<td>0.000</td>
<td>19.383</td>
</tr>
<tr>
<td>1169.8</td>
<td>83.28</td>
<td>0.00</td>
<td>85.29</td>
<td>40.80</td>
<td>93.13</td>
<td>85.62</td>
<td>532</td>
<td>19.383</td>
<td>0</td>
<td>0.000</td>
<td>19.383</td>
</tr>
<tr>
<td>1170.8</td>
<td>0.00</td>
<td>84.80</td>
<td>33.76</td>
<td>40.83</td>
<td>93.13</td>
<td>85.62</td>
<td>533</td>
<td>19.417</td>
<td>0</td>
<td>0.000</td>
<td>19.417</td>
</tr>
<tr>
<td>1171.8</td>
<td>0.00</td>
<td>84.80</td>
<td>83.67</td>
<td>40.83</td>
<td>93.13</td>
<td>85.62</td>
<td>533</td>
<td>19.417</td>
<td>0</td>
<td>0.000</td>
<td>19.417</td>
</tr>
<tr>
<td>1172.8</td>
<td>76.89</td>
<td>0.00</td>
<td>31.00</td>
<td>40.87</td>
<td>93.13</td>
<td>85.62</td>
<td>534</td>
<td>19.447</td>
<td>0</td>
<td>0.000</td>
<td>19.447</td>
</tr>
<tr>
<td>1173.8</td>
<td>76.89</td>
<td>0.00</td>
<td>82.40</td>
<td>40.87</td>
<td>93.13</td>
<td>85.62</td>
<td>534</td>
<td>19.447</td>
<td>0</td>
<td>0.000</td>
<td>19.447</td>
</tr>
<tr>
<td>1174.8</td>
<td>0.00</td>
<td>83.57</td>
<td>29.54</td>
<td>40.92</td>
<td>93.13</td>
<td>85.62</td>
<td>535</td>
<td>19.476</td>
<td>0</td>
<td>0.000</td>
<td>19.476</td>
</tr>
<tr>
<td>1175.8</td>
<td>0.00</td>
<td>83.57</td>
<td>82.20</td>
<td>40.92</td>
<td>93.13</td>
<td>85.62</td>
<td>535</td>
<td>19.476</td>
<td>0</td>
<td>0.000</td>
<td>19.476</td>
</tr>
<tr>
<td>1176.8</td>
<td>86.03</td>
<td>0.00</td>
<td>31.61</td>
<td>40.97</td>
<td>93.13</td>
<td>85.62</td>
<td>536</td>
<td>19.507</td>
<td>0</td>
<td>0.000</td>
<td>19.507</td>
</tr>
</tbody>
</table>

• Many useful options: class, compiler, gc, gccapacity, gccause, gcnew, gcold, gcutil, ...
  – class – number of classes loaded/unloaded, Kbytes loaded/unloaded
  – compiler - number of compiled, failed, invalid compilations, with failedtype/method
  – gcutil – garbage collection statistics; survivor space 0,1, eden, old space, permanent, young generation, gc collection time, …
Byteman

A tool that simplifies Java tracing, monitoring and testing
• [http://byteman.jboss.org/](http://byteman.jboss.org/)

```
RULE ServerLogger_serverStopped
CLASS org.jboss.as.server.ApplicationServerService
METHOD stop
AT ENTRY
IF TRUE
DO
    openTrace("stop","/tmp/stop.log");
    traceStack("stop: "+Thread.currentThread.getName()+"\n","stop");
ENDRULE
```
Java Tools

What have we learned
• Java have decent monitoring tools out of the box
• Not as easy to use, but powerful
• Gives a lot of information as a (fairly) low cost

• Also included
  – VisualVM
  – JConsole
Tools

Final thought

• Which tool you use/prefer is not important
Final thought

- Which tool you use/prefer is not important
  - What is important is that you can measure everything
Profilers

High CPU load

Can profilers help?

Profilers (try) to show “where” application time is spent
Different CPU Profilers

• Instrumenting
  – Adds timing code to application

• Sampling
  – Collect thread dumps periodically
  – VisualVM, JProfiler, Yourkit, ….
Sampling CPU Profilers

Which tools do you use for application profiling?

- **46.5%** VisualVM
- **25.7%** JProfiler
- **20.6%** Custom in-house tools
- **17.1%** Java Mission Control
- **15.0%** YourKit
- **13.9%** None
- **8.4%** Don’t know
- **6.2%** Other
- **9.1%** NetBeans profiler
- **3.5%** JProbe
- **3.3%** XRebel

[survey made by RebelLabs]
Sampling Profilers

Samples

MainThread.run() -> Backend.findUser() -> new User() -> fetchUser() -> someLogic() -> Controller.doAction() -> evenMoreLogic() -> moreLogic() -> display() -> evenMoreLogic()
Sampling Profilers

More samples

MainThread.run()
Controller.doAction()
Backend.findUser()
new User()
fetchUser()
someLogic()
moreLogic()
display()
evenMoreLogic()
Sampling Profilers

How do they work?

- Most profilers use JVMTI
  - Using GetCallTrace
    - Triggers a global safepoint
    - Collect stack trace
- Large impact on application
- Samples only at safepoints
What is a safepoint?

A safepoint is a range of execution where the state of the thread is well described.

Java threads poll a global flag
- At method exit/enter
- At “uncounted” (int) loops
What is a safepoint?

A safepoint can be delayed by

– Large Methods
– Long running “uncounted” (int) loops
Sampling Profilers

Safepoint bias

MainThread.run()

Backend.findUser()

Controller.doAction()

new User()

fetchUser()

someLogic()

evenMoreLogic()

moreLogic()

display()

MainThread.run()
Safepoint summary

JVM have many safepoint operations

• To monitor its impact use
  – -XX:+PrintGCApplicationStoppedTime
  – -XX:+PrintSafepointStatistics
  – -XX:+PrintSafepointStatisticsCount=X
Sample Profilers

Java profilers that work better!

• Profilers that use AsyncGetCallTrace
  – Java Mission Control
    • Very good tool, just don’t use it in production (without a license)
  – Oracle Solaris Studio
    • Not well known, it’s free and it works on Linux
  – Honest Profiler
    • Open source, cruder, but works very well
Java Mission Control/Flight Recorder

• Designed for usage in production systems
• Potentially low overhead
  – Be wary of what you record
    • Capture everything is not a good strategy
    • Capture data for one specific thing (memory, contention, cpu)
• Capturing data will not be skewed by safepoints
  – Make sure to use:
    • -XX:+UnlockDiagnosticVMOptions -XX:+DebugNonSafepoints
• Need a license to be used in production, free to use for development
Honest-Profiler

- Open source!!
- [https://github.com/RichardWarburton/honest-profiler](https://github.com/RichardWarburton/honest-profiler)
- Works on OpenJDK
- Will by default enable
  - `-XX:+UnlockDiagnosticVMOptions -XX:+DebugNonSafepoints`
Java Profilers problem

- Java stack only
- Method tracing have large observer effect
- Do not report:
  - GC
  - Deopt
  - Eg System.arraycopy ....
Perf

• A system profiler
• Show the whole stack
  – JVM
  – GC
  – Kernel

• Previously not very useful for Java
  – Missing stacks
  – Method symbols missing
Perf

• Solution
  – Java8u60 added the option -XX:PreserveFramePointer
    • Allows system profilers to capture stack traces

• Visibility of everything
  – Java methods
  – JVM, GC, libraries, kernel

• Low overhead
• Flamegraph
Flamegraph

Top edge show which method is using CPU and how much.
Limitations of Perf Mixed Mode

• Frames might be missing (inlined)

• Disable inlining:
  – -XX:-Inline
  – Many more Java frames
  – Will be slower

• perf-map-agent have experimental un-inline support
Profiling at Red Hat

- Perf for CPU profiling
- Java Mission Control
  - Contention
  - Thread analysis
  - Memory usage
Benchmarking
Benchmarking

Most benchmarks are flawed, wrong and full of errors.
Benchmarking

Most benchmarks are flawed, wrong and full of errors
- Used cautiously it can be a very valuable tool to reveal performance issues, compare applications, libraries, ...
Benchmarking

• Java Benchmarking libraries
  – Microbenchmarks
    • Java Microbenchmark Harness
      – http://openjdk.java.net/projects/code-tools/jmh/
    • JMH is the only library on Java we recommend for microbenchmarks
    • Still possible to write bad benchmarks with JMH
Benchmarking

Load testing frameworks
- Gatling
  - http://gatling.io/

- Faban
  - http://faban.org/
Benchmarking

A Hibernate microbenchmark lesson

• We wanted to compare cache lookup performance between Hibernate 4 and 5
  – We also wanted to test a new cache we added to Infinispan, Simple cache

• Created several JMH benchmarks that tested different use cases
  – Default cache with and without eviction
  – Simple cache with and without eviction
Benchmarking

A Hibernate microbenchmark lesson

<table>
<thead>
<tr>
<th></th>
<th>Operations per Second</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.3 w/ eviction</td>
<td>292050 ops/s</td>
</tr>
<tr>
<td>4.3 w/o eviction</td>
<td>303992 ops/s</td>
</tr>
<tr>
<td>5.0 w/ eviction</td>
<td>314629 ops/s</td>
</tr>
<tr>
<td>5.0 w/o eviction</td>
<td>611487 ops/s</td>
</tr>
<tr>
<td>5.0 w/o eviction (simple cache)</td>
<td>706509 ops/s</td>
</tr>
</tbody>
</table>
Benchmarking

We progressed to a large full scale EE benchmark.

• We expect similar or better results vs EAP6 (Hibernate 4.2)
Benchmarking

We progressed to a large full scale EE benchmark.

- We expect similar or better results vs EAP6 (Hibernate 4.2)
  - The response times are 5-10x slower!
  - And we are now CPU bound
Benchmarking

We progressed to a large full scale EE benchmark.

• We expect similar or better results vs EAP6 (Hibernate 4.2)
  – The response times are 5-10x slower!
  – And we are now CPU bound

• How is that possible?
Benchmarking

We progressed to a large full scale EE benchmark.

• We expect similar or better results vs EAP 6 (Hibernate 4.2)
  – The response times are 5-10x slower!
  – And we are now CPU bound

• How is that possible?
  – We had several microbenchmarks showing better performance on Hibernate 5
  – Could the problem lie in WildFly 10?
  – Let’s profile!
Benchmarking
Benchmarking

DefaultLoadEventListener:convertCacheEntryToEntity (12,242 samples, 5.76%)
Benchmarking

What had happened?

• Why did we spend so much CPU in this method and why do it not reflect the results from the microbenchmarks?
Benchmarking

What had happened?

• Why did we spend so much CPU in this method and why do it not reflect the results from the microbenchmarks?
  – We started debugging
    • -XX+PrintCompilation
      – Show when/which methods are compiled/decompiled
      – Showed that the method only was compiled once, but twice on EAP 6 (Hib4.2)
      – Why?
Benchmarking

What had happened?

• We had made several changes internally in Hibernate 5 to reduce memory usage
Benchmarking

What had happened?

• We had made several changes internally in Hibernate 5 to reduce memory usage
  – One specific class was refactored to an interface
  – Changing a monomorphic call site to a bimorphic call site
    • Which can prevent compiling/inlining of methods!
Benchmarking

What had happened?

• We had made several changes internally in Hibernate 5 to reduce memory usage
  – One specific class was refactored to an interface
  – Changing a monomorphic call site to a bimorphic call site
    • Which can prevent compiling/inlining of methods!

• Direct method calls will “always” be optimized (no subclassing)
• In Java subclassing/inheritance might cause the JVM to no optimize method calls
Benchmarking

Lessons learned

• Code changes might have unexpected consequences
  – In this case, reduced memory, increased CPU usage

• Benchmarks are good, but should always be questioned
  – When you benchmark, analyze (all/most) possible use cases
Benchmarking

Lessons learned

• Code changes might have unexpected consequences
  – In this case, reduced memory, increased CPU usage

• Benchmarks are good, but should always be questioned
  – When you benchmark, analyze (all/most) possible use cases

• Yes, we ended up fixing the CPU usage without causing it to use more memory!
EAP 7 Performance Improvements

Performance highlights

- JPA
- JCA
- Web
EAP 7 Performance Improvements

Hibernate 5 performance improvements

- Memory usage
  - Reduced memory usage 20-50%

- Cache improvements
  - Simple Cache
    - 132% Improvement with immutable objects

- Pooled Optimizer
  - In persistence.xml add property
    - hibernate.id.optimizer.pooled.preferred value="pooled-lotl"
EAP 7 Performance Improvements

Hibernate 5 performance improvements

- Bytecode enhancements
  - hibernate.enhancer.enableDirtyTracking
  - hibernate.enhancer.enableLazyInitialization

- Improved caching
  - hibernate.cache.use_reference_entries
  - hibernate.cache.infinispan.immutable-entity.cfg = “immutable-entity”
EAP 7 Performance Improvements

IronJacamar performance improvements

- Contention issues
  - New default ManagedConnectionPool
- Disabled logging
  - Not everything is disabled
    - `<datasource .... enlistment-trace="false"`>
- Fairness
  - Set to true at default
    - `<datasource> <pool><fair>false</fair></pool>`
EAP 7 Performance Improvements

Undertow vs JBossWeb

- Undertow is nonblocking
- JBossWeb have a thread for each connection
# EAP 7 Performance Improvements

Simple HelloServlet Benchmark

<table>
<thead>
<tr>
<th>RPS</th>
<th>EAP 6.4 Mean</th>
<th>EAP 6.4 95%</th>
<th>EAP 6.4 99%</th>
<th>EAP 7 Mean</th>
<th>EAP 7 95%</th>
<th>EAP 7 99%</th>
</tr>
</thead>
<tbody>
<tr>
<td>5000</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>15000</td>
<td>3</td>
<td>4</td>
<td>56</td>
<td>1</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>25000</td>
<td>13</td>
<td>18</td>
<td>247</td>
<td>1</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>40000</td>
<td>46</td>
<td>246</td>
<td>414</td>
<td>12</td>
<td>23</td>
<td>84</td>
</tr>
<tr>
<td>55000</td>
<td>361</td>
<td>647</td>
<td>1889</td>
<td>65</td>
<td>136</td>
<td>635</td>
</tr>
</tbody>
</table>
EAP 7 Performance Improvements

- Lightweight
- Embeddable
- Scalable
- HTTP/2
- HTTP Upgrade
EAP 7 Performance Improvements

Overall EAP 7 Performance tricks

• Remote byte pooling
  – jboss.remoting.pooled-buffers

• Disable CDI
  – CDI is enabled by default in EAP 7
EAP 7 Performance Improvements

Going from EAP 6 to EAP 7

• Overall 20-25% improvement
• Scales better
• Uses/allocates less memory
LEARN. NETWORK. EXPERIENCE OPEN SOURCE.