Measuring JUC Lock Contention in the IBM JVM

Panagiotis Patros, Eric Aubanel, David Bremner and Michael Dawson
University of New Brunswick, IBM Ottawa
Faculty of Computer Science
{Patros.Panos, Aubanel, Bremner}@unb.ca
Michael_Dawson@ca.ibm.com

Conclusions
• The JUC\(^1\) Park Contention Tool
  ✓ Incorporated in the IBM JVM
  ✓ First such tool in any JVM
  ✓ Measures JUC related \textit{Thread Park} \(^2\) contention
  ✓ Near zero overhead
  ✓ Provides a number of metrics
  ✓ Discovers JUC bottlenecks
  ✓ Controlled with command line parameters
• With experimentation
  ✓ Discovered JUC bottlenecks in Java applications
  ✓ Only by optimizing their critical sections were there speedups
  ✓ Metrics that hit near maximum values for bottlenecks:
    ✓ \textit{Peak Parked}
    ✓ \% \textit{Real Time Util}
    ✓ \% \textit{Thread Time Util}
  ✓ Results comparable to IBM’s HealthCenter for intrinsic locks
  ✓ HealthCenter cannot find JUC bottlenecks
  ✓ \textit{Average Hold Time} metric accurately estimates size of critical section

Overhead\(^3\)

Example of Metrics

Discovering Bottlenecks in Java code

<table>
<thead>
<tr>
<th>Lock</th>
<th>Times Parked</th>
<th>Peak Parked</th>
<th>Avg Park Time (ms)</th>
<th>Avg Hold Time (ms)</th>
<th>% Real Time Util</th>
<th>% Thread Time Util</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1681</td>
<td>63</td>
<td>3703.59</td>
<td>59.48</td>
<td>99.60%</td>
<td>6201.05%</td>
</tr>
<tr>
<td>2</td>
<td>83</td>
<td>50</td>
<td>407.02</td>
<td>16.24</td>
<td>1.34%</td>
<td>33.65%</td>
</tr>
<tr>
<td>1</td>
<td>67</td>
<td>59</td>
<td>126.97</td>
<td>4.30</td>
<td>0.29%</td>
<td>8.47%</td>
</tr>
</tbody>
</table>

\texttt{static final long \texttt{S1} = 4, \texttt{S2} = 16, \texttt{S3} = 64;}

\texttt{static final int \texttt{threadCount} = 64;}

\texttt{static final ReentrantLock \texttt{l1} = \texttt{new ReentrantLock()};}
\texttt{l2 = new ReentrantLock();}
\texttt{l3 = new ReentrantLock();}

\texttt{public void \texttt{JUCLocks}() throws InterruptedException {}
  \texttt{try {}
    \texttt{counter1++;}
    \texttt{Thread.sleep(S1);}
  \} \texttt{finally {}
    \texttt{l1.unlock();}
  \}
  \texttt{localCounter++;}

  /*Similarly for two more locks*/
}

Acknowledgements
• Atlantic Canada Opportunities Agency (ACOA)
• IBM Ottawa
• Fellow researchers in the IBM CAS – Atlantic project

---

\(^1\) Java Util Concurrent (JUC) is the Java library for concurrent and parallel functionality
\(^2\) Thread Park is JUC’s approach on blocking threads’ execution
\(^3\) Overhead measured using SPECjbb 2013