The Other HPC: Profiling Enterprise-scale Applications

Marty Itzkowitz
Senior Principal SW Engineer, Oracle
marty.itzkowitz@oracle.com
Agenda

• HPC Applications
  • Traditional HPC
  • The Other HPC
• Profiling Enterprise-Class Applications
  • SPECjbb, SPECjAppserver, SPECjEnterprise
  • SOA
  • Oracle Database
Traditional HPC

• Intensive numerical calculations
  • Fortran/C/C++
  • OpenMP/MPI
• Run on many CPUs, nodes
  • Many threads (OpenMP)
  • Many processes (MPI)
  • Hybrid runs
• Multiple processes tend to be uniform
• Computations are mostly loop-based
The Other HPC

- Transactions and web services
  - Java/C/C++
  - Ad hoc parallelism
- Also run on many CPUs, nodes
  - Long duration — web servers run forever
  - Many threads
  - Many processes
  - But not quite peta-scale (yet)
- Multiple processes are not uniform
- Often not loop-based
Profiling Enterprise-Class Applications

- Many processes, many threads; long duration
  - Need to track all
  - Typically have long initialization phase
- Multi-thread performance issues
  - Lock contention: lock-global vs. lock-local
    - Synchronization tracing (use `collect -s on`)‏
    - Key issue: scoping of locks
  - Load imbalance
    - Useful work matters, not CPU usage
      - Busy-waits use CPU resources, but are not useful work
Profiling Enterprise-Class Applications (continued)

• Complex start up: launch by script
  • Add env.var. to prepend `collect` command to target invocation
  • No effect if not set; data collection if set
  • `-y` argument for data-collection control (e.g., skip initialization)
  • `-l` argument for event marking (e.g., mark transaction begin/end)
    • API calls in user code can be used to for markers, too
      • Calls ignored if no data being collected

• Filtering to drill down on problems
  • Based on function on stack
  • Based on threads, processes, CPUs
  • Between marked events
SpecJBB

- Benchmark for three-tier enterprise system
  - Based on TPC-C
  - A small enterprise-scale application
- Models a wholesale company and order-entry system
  - Has warehouses that serve districts
    - Run does first 1, then 2, ..., 16 warehouses
      - Up to twice the number of CPUs detected
      - First eight ignored, last eight count for score
  - Processes orders, deliveries, payments, etc.
  - Has no real database interactions
    - Data records stored as HashMaps or TreeMaps
- Run on 8-CPU machine, uses 156 threads
  - New set of 2N threads created for warehouse N
  - Completely CPU-bound
SpecJBB: Call Tree

Shows hottest path
SpecJBB: Timeline

Transition from 15 warehouses to 16
Old threads terminate; new threads are created
SpecJAppServer

- Profile of WebLogic Application Server
  - Simulates standard e-commerce application
  - Processes requests from clients via browser for purchases
  - Processes requests via CORBA/IIOP to manage inventory
- Run on 128-CPU machine, uses ~280 threads
- Data collection paused during initialization phase
  - Recorded data shows active window ~400 seconds
SpecJAppServer: Timeline

Time from ~7500 – 7900 seconds
Threads 157-170; two different types of threads shown
SpecJAppServer: Function List

<table>
<thead>
<tr>
<th>Name</th>
<th>User CPU (sec)</th>
<th>User CPU (sec)</th>
<th>Sys. CPU (sec)</th>
<th>Sys. CPU (sec)</th>
<th>Sys. CPU (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;Total&gt;</td>
<td>4886.282</td>
<td>1925.457</td>
<td>1925.457</td>
<td>2071.017</td>
<td>2071.017</td>
</tr>
<tr>
<td>weblogic.kernel.ExecutableThread.run()</td>
<td>6.234</td>
<td>4379.373</td>
<td>0.300</td>
<td>1897.047</td>
<td>1897.047</td>
</tr>
<tr>
<td>weblogic.kernel.ExecutableThread.execute(weblogic.kernel.ExecutableThread)</td>
<td>0.155</td>
<td>4354.436</td>
<td>0.010</td>
<td>1865.885</td>
<td>1865.885</td>
</tr>
<tr>
<td>weblogic.kernel.ExecutableThread.execute(weblogic.kernel.ExecutableThread)</td>
<td>0.530</td>
<td>4295.785</td>
<td>0.050</td>
<td>1859.292</td>
<td>1859.292</td>
</tr>
<tr>
<td>weblogic.work.ExecuteRequestAdaptor.execute(weblogic.com)</td>
<td>3.352</td>
<td>254.957</td>
<td>0.220</td>
<td>1310.007</td>
<td>1310.007</td>
</tr>
<tr>
<td>weblogic.servlet.internal.ServletRequestImpl.run()</td>
<td>1.361</td>
<td>177.653</td>
<td>0.050</td>
<td>1281.997</td>
<td>1281.997</td>
</tr>
<tr>
<td>weblogic.servlet.servlet.WebAppServletContext.executeServlet</td>
<td>2.121</td>
<td>169.797</td>
<td>0.140</td>
<td>1281.716</td>
<td>1281.716</td>
</tr>
<tr>
<td>weblogic.servlet.servlet.WebAppServletContext.executeServlet</td>
<td>0.791</td>
<td>156.538</td>
<td>0.040</td>
<td>1280.388</td>
<td>1280.388</td>
</tr>
<tr>
<td>weblogic.security.service.SecurityManager.runAs(weblogic)</td>
<td>7.618</td>
<td>155.808</td>
<td>0.470</td>
<td>1280.566</td>
<td>1280.566</td>
</tr>
<tr>
<td>weblogic.security.ncl.internal.AuthenticateSubject.run(Subject.run)</td>
<td>1.301</td>
<td>85.949</td>
<td>0.030</td>
<td>1276.103</td>
<td>1276.103</td>
</tr>
<tr>
<td>weblogic.servlet.servlet.WebAppServletContext.executeServlet</td>
<td>1.221</td>
<td>87.633</td>
<td>0.080</td>
<td>1275.742</td>
<td>1275.742</td>
</tr>
<tr>
<td>weblogic.servlet.servlet.ServletSubServlet.executeServlet</td>
<td>11.908</td>
<td>177.603</td>
<td>0.610</td>
<td>1275.702</td>
<td>1275.702</td>
</tr>
<tr>
<td>weblogic.servlet.servlet.ServletSubServlet.executeServlet</td>
<td>0.250</td>
<td>72.659</td>
<td>0.020</td>
<td>1275.372</td>
<td>1275.372</td>
</tr>
<tr>
<td>weblogic.servlet.servlet.ServletSubServlet.executeServlet</td>
<td>0.901</td>
<td>72.519</td>
<td>0.060</td>
<td>1275.362</td>
<td>1275.362</td>
</tr>
<tr>
<td>weblogic.servlet.servlet.ServletSubServlet.executeServlet</td>
<td>1.051</td>
<td>72.199</td>
<td>0.040</td>
<td>1275.332</td>
<td>1275.332</td>
</tr>
<tr>
<td>java.servlet.http.HttpServletRequestServlet.invokeServlet</td>
<td>1.321</td>
<td>71.176</td>
<td>0.040</td>
<td>1275.292</td>
<td>1275.292</td>
</tr>
<tr>
<td>java.servlet.http.HttpServletRequestServlet.invokeServlet</td>
<td>0.030</td>
<td>69.817</td>
<td>0.0</td>
<td>1275.252</td>
<td>1275.252</td>
</tr>
<tr>
<td>org.spec.jappserver.servlet.HttpServlet.doGet(java.servlet.http.HttpServletRequest)</td>
<td>9.907</td>
<td>69.817</td>
<td>0.520</td>
<td>1275.252</td>
<td>1275.252</td>
</tr>
<tr>
<td>java.net.SocketOutputStream.socketWrite(byte[], int, int)</td>
<td>9.467</td>
<td>142.259</td>
<td>0.191</td>
<td>964.745</td>
<td>964.745</td>
</tr>
<tr>
<td>java.net.SocketOutputStream.socketWrite(byte[], int, int)</td>
<td>5.174</td>
<td>132.783</td>
<td>0.420</td>
<td>963.914</td>
<td>963.914</td>
</tr>
<tr>
<td>java.net.SocketOutputStream.socketWrite(0)</td>
<td>5.086</td>
<td>72.079</td>
<td>0.380</td>
<td>960.722</td>
<td>960.722</td>
</tr>
</tbody>
</table>

Sorted by system CPU time – implies I/O activity
SpecJEnterprise

- Benchmark emulates automobile manufacturer
  - Stresses Java EE 5 servers, JVM, CPU, etc.
  - Three domains: Dealer, Manufacturing and Supplier
  - Driver drives the benchmark
    - Runs on different system
    - Successor benchmark to SPECjAppserver
- Run on 128-CPU machine, uses 282 threads
- Data collection enabled for two 300 second snaps
  - First at 2436 seconds, second at 5026 seconds
  - Data covers only those two intervals
SpecJEnterprise: Timeline

Data was collected only for two intervals
SpecJEnterprise: Call Tree

Most time spent in WebLogic middleware
Oracle SOA Suite

• SOA = Service-Oriented Architecture
• Single service component architecture
  • Based on Fusion Middleware and WebLogic
  • High throughput, low latency
  • Unified event-driven and service-oriented capabilities
    • Handles complex events
• Near real-time performance requirement
• Run on 64-CPU machine, using 166 threads
  • One run, collected clock- and cache-miss-profiles
SOA: Functions

Two main paths: HotSpot compiler and weblogic
(Inferred from function names)
**SOA: Filter by Function in Stack**

Function list shows data only from events with stacks containing `weblogic.work.ExecuteThread.execute()`
Oracle Database Profile

- Collected during TPC-H power test
- Script launches server, with \(-y\) \texttt{USR} flag
- Queries launched by a second script
  - Send \texttt{SIGUSR} to enable data collection
  - Run one query
  - Send \texttt{SIGUSR} to disable data collection
- Experiment has markers for each query
- Run on 128-CPU machine, uses 906 processes
  - Many are ephemeral, with no profile ticks
  - 256 processes do significant work
Oracle Database: Function List

<table>
<thead>
<tr>
<th>Function</th>
<th>User CPU</th>
<th>User CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;Total&gt;</td>
<td>49288.428</td>
<td>49288.428</td>
</tr>
<tr>
<td>_start</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>main</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>opidrv</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>opsimai_real</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>opdirr</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>sou2o</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ssthcmdmain</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>kxftpmdp</td>
<td>0.030</td>
<td>0.030</td>
</tr>
<tr>
<td>kxfxwmi</td>
<td>0.040</td>
<td>0.040</td>
</tr>
<tr>
<td>kxfxsp</td>
<td>0.040</td>
<td>0.040</td>
</tr>
<tr>
<td>upirtrc</td>
<td>0.030</td>
<td>0.030</td>
</tr>
<tr>
<td>kpooodr</td>
<td>0.050</td>
<td>0.050</td>
</tr>
<tr>
<td>opiodr</td>
<td>0.010</td>
<td>0.010</td>
</tr>
<tr>
<td>kfxsexecute</td>
<td>0.120</td>
<td>0.120</td>
</tr>
<tr>
<td>OCISstatExecute</td>
<td>0.100</td>
<td>0.100</td>
</tr>
</tbody>
</table>

~40 minute run
Oracle Database: per-CPU Profile

Sorted by CPU Number
Oracle Database: per-Process Profile

Per-process profile; filter set for top 5 processes
Oracle Database: Top Five Processes

Function list data filtered to show only the top 5 processes
SOFTWARE. HARDWARE. COMPLETE.