Performance Tuning Transaction Processing Systems

Dr. Russ Shermer, CSQA, CSTE
Introduction

• Motivation & background
• Comparison of Real-time and Batch
• Terminology & theory
• Discussion of tools and techniques used
• Results summary
Motivation

• Web-based, real-time systems are popular and well-understood
• Backend-transactional systems are still fundamental to most businesses
• The approach to tuning these systems is radically different than for real-time
Background

- Large claims processing implementation
- Government-mandated deadline
- 3rd Party Java Application Server platform
- Politics, vendors, customers…
Challenges

- Ill-defined requirements
- Aggressive deadlines
- Standard performance products not effective
- Limited diagnostic tools
- Poor test data
- Off-hours environment access
- Translation of test results to real results
System Under Test

- Input Folder
- Output Folder
- Transaction Processor
Performance Engine

- Homegrown
- Generates required data prior to the run
- Controlled feed rate
- Calculates throughput after the run via file timestamps
Batch vs. Realtime

• Realtime: response vs. load is the most important statistic

• Batch: overall throughput is the critical value
Latency vs. Throughput

• Real-time systems need to be low-latency
• Batch systems need high throughput
• Some performance tunings (efficiency types) improve both
• Others improve one at the cost of the other
Processing Metaphor

The obstacle course:

Start

Finish

= 1 processing event
Latency

- Time for one event to complete
Throughput

- Number of event completions per second
Pipelining

• Concurrent processing through serialization
Parallelism

Concurrency through duplication:
Tuning Techniques

• Test system was nearly identical to production except for number of CPUs in the application server
• Maximize throughput then:
• Tune to make system “CPU bound”
Tuning Techniques

• Increased CPU Utilization
  – Focused on opening concurrency where needed
  – Easiest to implement
  – Had to watch memory consumption
Tuning Techniques

- Database indexing
- Oracle Statspack for analysis
- Network traffic reduction
- Ntop for analysis
Tuning Issues

- Portions of application were batch, others Real-time
- Memory consumption – hit machine and Java limits
- Occasionally exposed software defects when increasing concurrency
Test Execution

- Fully scripted
  - Shutdown application
  - Reset application data/database
  - Apply new tuning configuration
  - Restart application
  - Start monitoring
  - Run transactions
  - End test
  - Capture results
Scripting Advantages

- Off-hours performance runs
- Required for long test runs
- Allowed / required codification of tuning configuration
- Complete test run history archived
Captured Monitoring

• %CPU by process over time (homegrown)
• Ntop – network utilization (automated via wget)
• Vmstat – virtual memory statistics
•Mpstat – multiprocessor statistics
• lsstat (disk I/O statistics)
• Statspack – Oracle database statistics
Top

last pid: 26214;
load averages: 2.64, 2.68, 2.68 13:26:10

128 processes:
124 sleeping, 1 running, 1 zombie, 2 on cpu

CPU states:
45.7% idle, 30.9% user, 21.9% kernel, 1.5% iowait,
0.0% swap

Memory: 16G real, 8417M free, 11G swap in use, 9396M swap free
<table>
<thead>
<tr>
<th>PID</th>
<th>USERNAME</th>
<th>THR</th>
<th>PRI</th>
<th>NICE</th>
<th>SIZE</th>
<th>RES</th>
<th>STATE</th>
<th>TIME</th>
<th>CPU</th>
<th>COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>12922</td>
<td>vitria01</td>
<td>14</td>
<td>0</td>
<td>0</td>
<td>632M</td>
<td>207M</td>
<td>cpu/0</td>
<td>171.6H</td>
<td>24.73%</td>
<td>java</td>
</tr>
<tr>
<td>12935</td>
<td>vitria01</td>
<td>13</td>
<td>0</td>
<td>0</td>
<td>159M</td>
<td>115M</td>
<td>run</td>
<td>147.6H</td>
<td>14.65%</td>
<td>java</td>
</tr>
<tr>
<td>15594</td>
<td>vitria01</td>
<td>13</td>
<td>59</td>
<td>0</td>
<td>148M</td>
<td>103M</td>
<td>sleep</td>
<td>57.5H</td>
<td>13.71%</td>
<td>java</td>
</tr>
<tr>
<td>6210</td>
<td>vitria01</td>
<td>52</td>
<td>59</td>
<td>0</td>
<td>379M</td>
<td>337M</td>
<td>sleep</td>
<td>115:21</td>
<td>1.29%</td>
<td>java</td>
</tr>
<tr>
<td>26143</td>
<td>a0cs73</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>2816K</td>
<td>1704K</td>
<td>cpu/1</td>
<td>0:00</td>
<td>0.13%</td>
<td>top</td>
</tr>
<tr>
<td>27629</td>
<td>vtstbot1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1520K</td>
<td>1208K</td>
<td>sleep</td>
<td>16:12</td>
<td>0.08%</td>
<td>HTVQServ</td>
</tr>
<tr>
<td>25990</td>
<td>root</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>3112K</td>
<td>2080K</td>
<td>sleep</td>
<td>0:00</td>
<td>0.08%</td>
<td>sshd</td>
</tr>
<tr>
<td>25360</td>
<td>vitria05</td>
<td>12</td>
<td>59</td>
<td>0</td>
<td>151M</td>
<td>67M</td>
<td>sleep</td>
<td>0:38</td>
<td>0.06%</td>
<td>java</td>
</tr>
<tr>
<td>7484</td>
<td>vitria03</td>
<td>27</td>
<td>59</td>
<td>0</td>
<td>1043M</td>
<td>603M</td>
<td>sleep</td>
<td>8:49</td>
<td>0.02%</td>
<td>chanserv</td>
</tr>
<tr>
<td>17278</td>
<td>vitria05</td>
<td>27</td>
<td>59</td>
<td>0</td>
<td>213M</td>
<td>78M</td>
<td>sleep</td>
<td>1:06</td>
<td>0.02%</td>
<td>java</td>
</tr>
<tr>
<td>1094</td>
<td>root</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1104K</td>
<td>824K</td>
<td>sleep</td>
<td>2:42</td>
<td>0.01%</td>
<td>hardmond</td>
</tr>
<tr>
<td>24759</td>
<td>vitria05</td>
<td>11</td>
<td>59</td>
<td>0</td>
<td>218M</td>
<td>79M</td>
<td>sleep</td>
<td>0:12</td>
<td>0.01%</td>
<td>java</td>
</tr>
<tr>
<td>26142</td>
<td>a0cs73</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2736K</td>
<td>2224K</td>
<td>sleep</td>
<td>0:00</td>
<td>0.01%</td>
<td>tcsh</td>
</tr>
<tr>
<td>26080</td>
<td>a0cs73</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1864K</td>
<td>1376K</td>
<td>sleep</td>
<td>0:00</td>
<td>0.01%</td>
<td>ksh</td>
</tr>
<tr>
<td>12099</td>
<td>root</td>
<td>10</td>
<td>59</td>
<td>0</td>
<td>32M</td>
<td>9144K</td>
<td>sleep</td>
<td>569:39</td>
<td>0.01%</td>
<td>ntop</td>
</tr>
</tbody>
</table>
CPU Utilization vs. Time

• Perl script to capture CPU utilization by process
• Requires that each process is “tagged”
• Provides a visual CPU workload by component for a given test run
• Another homegrown tool
CPU Utilization
Ntop

- “Network Top”
- Freeware network analyzer
- Works by reading IP stack on NIC
- Does not perform individual packet analysis
- Has a web-based reporting interface
- Did I mention it was free?
Ntop

Last 60 Minutes Average Throughput

Throughput

9.5 Mbps
9.0 Mbps
8.5 Mbps
8.0 Mbps
7.5 Mbps
7.0 Mbps
6.5 Mbps
6.0 Mbps
5.5 Mbps
5.0 Mbps
4.5 Mbps
4.0 Mbps
3.5 Mbps
3.0 Mbps
2.5 Mbps
2.0 Mbps
1.5 Mbps
1.0 Mbps
0.5 Mbps
0.0 Mbps
09:50 09:48 09:46 09:44 09:42 09:38 09:36 09:34 09:32 09:30 09:28 09:26 09:24 09:22 09:20 09:16 09:14 09:12 09:10 09:08 09:06 09:04 09:02 08:58 08:56 08:54 08:52
Oracle Statspack

• Allows detailed response analysis
• Rates queries and procs by time and usage
• Identifies important bottlenecks
• Can be run remotely through sqlplus
Oracle Statspack

- Capture a “before” snapshot
- Run performance test
- Capture an “after” snapshot
- Run the statspack analyzer on the two snapshots
- Produces a differential report
### Oracle Statspack

<table>
<thead>
<tr>
<th>Buffer Gets</th>
<th>Executions</th>
<th>Gets per Exec</th>
</tr>
</thead>
<tbody>
<tr>
<td>301,995,711</td>
<td>781</td>
<td>386,678.2</td>
</tr>
<tr>
<td>SELECT RTRIM(&quot;A2&quot;.&quot;CON_PRV_ID&quot;),&quot;A2&quot;.&quot;PAY_...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10,400,496</td>
<td>1,058</td>
<td>9,830.3</td>
</tr>
<tr>
<td>SELECT RTRIM(&quot;A3&quot;.&quot;CON_PRV_ID&quot;),&quot;A2&quot;.&quot;IND_...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>494,289</td>
<td>1</td>
<td>494,289.0</td>
</tr>
</tbody>
</table>
| DECLARE       | fsdTable   | VARCHAR2(30)  | := ...
Analysis & Tuning

• Most of the analysis was performed by the performance team
• For one application, the development team was involved in the tuning (great benefit)

• Cycle
  Execute → Analyze → Tweak
Results

• Routing Application
  – Created three performance configurations
    1. Tiny realtime transactions (low latency)
    2. Medium high volume transactions (high throughput)
    3. Low volume, very large transactions (stability)
Results

• Routing Application
  – In addition to the configuration changes to improve performance,
  – Significant database tuning was identified
  – Code efficiencies were analyzed
  – Network bottlenecks drove format changes
Results

• Processing application
  – Concurrency exposed software defects
  – Database tuning made drastic improvements
  – Did not achieve the full results desired
Some Lessons Learned

- Automating the execution process improved productivity, reduced timelines and allowed testing on shared hardware
- Codification of results was critical
- Cooperation of the development team is key
- Easily accessible, dedicated hardware would have significantly decreased tuning time