JPDM, A Structured approach To Performance Tuning
About Me

Kodewerk Ltd Performance Consulting
Java Performance Tuning Workshops
Co-Founded jClarity
Professor Zapinsky proved that the squid is more intelligent than the housecat when posed with puzzles under similar conditions.
Our Typical Customer

- Application isn't performing to project sponsors' expectations
- Users wait for minutes for the application to respond
Our Typical Customer

Application isn't performing to project sponsors expectations
users wait for minutes for the application to respond
Developers Get Involved

StringBuffer is being used all over the place…
We need to change it to StringBuilder

Looks like a database problem, we need to migrate to [buzzname goes here]
where the @$*%! is the problem?
Managers Get Involved

Form a tiger team
What is a Tiger Team?

A team of specialists in a particular field brought together to work on specific tasks.
What is a Tiger Team?

A forum where experts from different disciplines come together to express an opinion that defends their specialty.
Measure Don’t Guess®

Copyright 2017 Kirk Pepperdine. All rights reserved
Why a System Model?

- Help us to understand
  - what measures are important
  - defines requirements for tooling
  - provides a context for us to understand the measures
  - facilitate the definition of a diagnostic process

- Java Performance Diagnostic Model (JPDM)
Where do Developers Live?
Developers Live Here

Application

Code (algorithms)
Developers Live Here

Work in a virtualized environment

Application

Code (algorithms)

JVM

Managed Memory, Execution Engine

Copyright 2017 Kirk Pepperdine. All rights reserved
public class Software {
    public static void main( String[] args) {
        System.out.println("Software is abstract");
    }
}
Hardware is Real!

- capacities
- volume
- throughput
- clock speed
- granularity
- cache line, sector size
CPU

- capacity: number of cores
- number of units in ALU
- size of caches
- throughput: clock speed, CPI
- bandwidth on various pipes (QPI)
- granularity: cache line size
Memory

- capacity: number of bytes of RAM
- throughput: clock speed of BUS
- tempered by bank cool of time
- chunk size: cache line size
- typically 8 reads per cache line
Disk

- capacity: number of bytes of RAM
- throughput: controller clock speed
  - ~1 Bbit/sec (SATA ~3Gbits/sec)
- granularity: disk sector (512 bytes)
- other latencies
  - sweep arm speed
Network

- capacity: 1 per network card
- bandwidth: maximum volume of data transferred per second (10^9 bits/sec)
- throughput: depends on protocol and overheads
- granularity: depends on protocol
- tcp payload is typically 1500 bytes
Other Limits

- Other hardware devices
  - eg, video, sound cards
- GPU
- Heat
- Battery capacities
Application
Code (algorithms)

JVM
Managed Memory, Execution Engine

OS/Hardware
CPU, memory, disk I/O
network I/O, Locks

Abstract meets Reality
No Dynamics
Add dynamics

Actors
usage patterns

Application
Code (algorithms)

JVM
Managed Memory, Execution Engine

OS/Hardware
CPU, memory, disk I/O
network I/O, Locks

Copyright 2017 Kirk Pepperdine. All rights reserved
Question?

Which is faster?

a) Bubble sort
b) Quick sort
Hint

In Big O notation...
- Bubble sort is $N^2$
- Quick sort of $N\log(N)$
However

quick

bubble
However

Quick bubble

Number of items comes from the actors
Performance Tuning Methodology

- Based on the System model we just developed
- hypothesis free
- methodical
- step wise process to arrive at a conclusion
Hardware Consumption

- Actors drive the application
- Application drives the JVM
- Application assisted by OS consumes Hardware
- Hardware is consumed
- Pattern of consumption is important

- Actors usage patterns
- Application, Code (algorithms)
- JVM, Managed Memory, Execution Engine
- OS/Hardware, CPU, memory, disk I/O, network I/O, Locks
Dominating Consumer

- Activity that dominates how the CPU is utilized
- Determination dominator by analyzing
  - CPU counters
  - Garbage collection logs

- Actors
  - usage patterns
- Application
  - Code (algorithms)
- JVM
  - Managed Memory, Execution Engine
- OS/Hardware
  - CPU, memory, disk I/O
  - network I/O, Locks
Dominating Consumers

Acts
- usage patterns

Application
- Code (algorithms)

JVM
- Managed Memory, Execution Engine

OS/Hardware
- CPU, memory, disk I/O
- network I/O, Locks

None

Copyright 2017 Kirk Pepperdine. All rights reserved
sys cpu > ~10% of user cpu

user CPU ≈ 100%

memory efficient? GC Logs

Thread starvation Thread dump

GC tuning, pool sizes, collectors, ...

Memory profiling, size frequency, life span,...

System profiling: netstat, mpstat, iostat, sar, strace, etc...

Application

None

JVM

Analysis

Copyright 2017 Kirk Pepperdine. All rights reserved
Expression of CPU Consumption

- passively dominant
- System
- JVM
- Application
- aggressivly dominant
- None
Measuring Consumption

Kernel time

System

User time

Application

Idle

None

JVM
<table>
<thead>
<tr>
<th>r</th>
<th>b</th>
<th>swpd</th>
<th>free</th>
<th>buff</th>
<th>cache</th>
<th>si</th>
<th>so</th>
<th>bi</th>
<th>bo</th>
<th>in</th>
<th>cs</th>
<th>us</th>
<th>sy</th>
<th>id</th>
<th>wa</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>9</td>
<td>100</td>
<td>24496</td>
<td>11096</td>
<td>13267036</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>96</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>100</td>
<td>23420</td>
<td>11088</td>
<td>13268328</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>77330</td>
<td>175352</td>
<td>17</td>
<td>26</td>
<td>39</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>100</td>
<td>20836</td>
<td>11088</td>
<td>13270628</td>
<td>0</td>
<td>0</td>
<td>68</td>
<td>105118</td>
<td>227382</td>
<td>14</td>
<td>40</td>
<td>21</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>100</td>
<td>23356</td>
<td>11080</td>
<td>13268272</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>80062</td>
<td>164387</td>
<td>12</td>
<td>30</td>
<td>29</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>100</td>
<td>23180</td>
<td>11084</td>
<td>13267068</td>
<td>0</td>
<td>0</td>
<td>72</td>
<td>98353</td>
<td>234851</td>
<td>15</td>
<td>43</td>
<td>28</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>100</td>
<td>25820</td>
<td>11088</td>
<td>13263676</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>120</td>
<td>100749</td>
<td>214921</td>
<td>11</td>
<td>42</td>
<td>17</td>
<td>30</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>100</td>
<td>22316</td>
<td>11088</td>
<td>13267176</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>103878</td>
<td>246723</td>
<td>16</td>
<td>56</td>
<td>19</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>100</td>
<td>21824</td>
<td>11088</td>
<td>13269140</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>48625</td>
<td>97288</td>
<td>15</td>
<td>16</td>
<td>9</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>100</td>
<td>20932</td>
<td>11080</td>
<td>13269808</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>110760</td>
<td>236774</td>
<td>14</td>
<td>41</td>
<td>24</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>12</td>
<td>100</td>
<td>23624</td>
<td>11084</td>
<td>13267488</td>
<td>0</td>
<td>0</td>
<td>204</td>
<td>69117</td>
<td>148611</td>
<td>15</td>
<td>27</td>
<td>25</td>
<td>33</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>100</td>
<td>24996</td>
<td>11096</td>
<td>13267476</td>
<td>0</td>
<td>0</td>
<td>164</td>
<td>24495</td>
<td>48552</td>
<td>13</td>
<td>10</td>
<td>30</td>
<td>48</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>12</td>
<td>100</td>
<td>20792</td>
<td>11096</td>
<td>13271872</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>25659</td>
<td>54331</td>
<td>8</td>
<td>9</td>
<td>26</td>
<td>56</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>8</td>
<td>100</td>
<td>21984</td>
<td>11080</td>
<td>13269920</td>
<td>0</td>
<td>0</td>
<td>20</td>
<td>46309</td>
<td>101404</td>
<td>16</td>
<td>18</td>
<td>51</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>9</td>
<td>100</td>
<td>22764</td>
<td>11080</td>
<td>13268956</td>
<td>0</td>
<td>0</td>
<td>16</td>
<td>88553</td>
<td>229557</td>
<td>17</td>
<td>35</td>
<td>38</td>
<td>11</td>
<td></td>
</tr>
</tbody>
</table>
Benchmark benchmark = new Benchmark()
benchmark.configure();
performance = benchmark.baseline(application);
user.setHappy(performance.meets(requirements));
while (!( ! user.isHappy()) && (user.hasMoney())) {
    Profiler profiler = performance.identifyDominatingConsumer();
    profilingResults = benchmark.profile(profiler);
    application.fixUsing(profilingResults);
    while ( application.failsQA())
        application.debug();
    performance = benchmark.baseline(application);
    user.setHappy(performance.meets(requirements));
}
Things We Need

- Actors
- Test harness
- Application
- JVM
- Hardware/OS

Monitoring

Data

Copyright 2017 Kirk Pepperdine. All rights reserved
Time for a demo
Dominating Consumer #1

- sys cpu > 10% of user cpu

- System
  - system profiling: netstat, mpstat, iostat, sar, strace, etc...

Question?

- Why the high level of kernel CPU?

Observations

- no disk I/O, network activity or video
- context switching due to lock contention?

Monitor threads with VisualVM
**Dominating Consumer #2**

- **Question?**
  - Why the high memory consumption
  - Profile object creation

- **User CPU ~ 100%**

- **Not memory efficient?**
  - GC Logs

- **Yes**
  - JVM

- **Memory profiling, size frequency, life span,...**

- **No**
  - sys cpu > 10% of user cpu
  - Not user CPU ~ 100%
What About Our Customer

- Need to gather clear requirements
- Develop a sound benchmarking environment
- Get better measurements
- Always identify dominating consumer
- Refocus teams on problems that matter
Questions?