G’Day, I’m Brendan

... also known as “shouting guy”
硬碟也會鬧情緒

癮科技小劇場：硬碟也會鬧情緒

由 Flow Yu 於 1 year 之前發表

文章分類: 儲存装置

這可不是開玩笑來著，而是經過實驗證明的唷！來自 Sun Fishworks 團隊的 Brendan Gregg，或許是在機房裡待久了，於是突發異想跑去對機房中的磁碟陣列大吼大叫一陣，結果就這麼發現了一個真理，那就是：硬碟就跟員工一樣，吼叫並不會讓他們的效率變高，反而還會讓它們心生不爽而降低士氣，私底下就開始搞罷工。
I do performance analysis
and I’m a DTrace addict
Agenda

• Performance
  • Workload Analysis and Resource Monitoring
  • Understanding available and ideal metrics before plotting

• Visualizations
  • Current examples
    • Latency
    • Utilization
  • Future opportunities
    • Cloud Computing
Visualizations like these

- The “rainbow pterodactyl”

- ... which needs quite a bit of explanation
Primary Objectives

- Consider performance metrics before plotting

- See the value of visualizations

- Remember key examples
Secondary Objectives

• Consider performance metrics before plotting
  • Why studying latency is good
  • ... and studying IOPS can be bad

• See the value of visualizations
  • Why heat maps are needed
  • ... and line graphs can be bad

• Remember key examples
  • I/O latency, as a heat map
  • CPU utilization by CPU, as a heat map
Content based on

• “Visualizing System Latency”, Communications of the ACM July 2010, by Brendan Gregg

• and more
Performance

Understanding the metrics before we visualize them
Performance Activities

- Workload analysis
  - Is there an issue? Is an issue real?
  - Where is the issue?
  - Will the proposed fix work? Did it work?

- Resource monitoring
  - How utilized are the environment components?
  - Important activity for capacity planning
Workload Analysis

• Applied during:
  • software and hardware development
  • proof of concept testing
  • regression testing
  • benchmarking
  • monitoring
Workload Performance Issues

• Load

• Architecture
Workload Performance Issues

- Load
  - Workload applied
  - Too much for the system?
  - Poorly constructed?

- Architecture
  - System configuration
  - Software and hardware bugs
Workload Analysis Steps

- Identify or confirm if a workload has a performance issue
  - Quantify
- Locate issue
  - Quantify
- Determine, apply and verify solution
  - Quantify
• Finding a performance issue isn’t the problem ... it’s finding the issue that matters
<table>
<thead>
<tr>
<th>ID#</th>
<th>Date</th>
<th>Type</th>
<th>Status</th>
<th>Sev</th>
<th>Version</th>
<th>Target</th>
<th>OS</th>
<th>Summary</th>
<th>Assigned</th>
</tr>
</thead>
<tbody>
<tr>
<td>53696</td>
<td>2010-05-17</td>
<td>Server: Performance</td>
<td>Verified (36 days)</td>
<td>S3</td>
<td>5.6.99</td>
<td>5.5+</td>
<td>Any</td>
<td>Performance schema engine violates the PSEA API by calling my_error()</td>
<td>Marc Alff</td>
</tr>
<tr>
<td>46886</td>
<td>2009-08-24</td>
<td>Tests: Cluster</td>
<td>Open (397 days)</td>
<td>S3</td>
<td></td>
<td>Any</td>
<td></td>
<td>Make the cluster regression performance testing more stable</td>
<td>Jørgen Austvik</td>
</tr>
<tr>
<td>48767</td>
<td>2009-11-13</td>
<td>Server: DB2SE for IBM</td>
<td>Open</td>
<td>S5</td>
<td></td>
<td>Any</td>
<td></td>
<td>IBMDB2I subselect performance degrades when async buffering enabled</td>
<td>Tim Clark</td>
</tr>
<tr>
<td>37703</td>
<td>2008-06-27</td>
<td>Server: General</td>
<td>Verified (241 days)</td>
<td>S5</td>
<td>MYSQL 6.0.4 (Source Distribution)</td>
<td>Linux (EL5.1)</td>
<td>MySQL performance with and without Fast Mutexes using Sysbench Workload</td>
<td></td>
<td></td>
</tr>
<tr>
<td>57012</td>
<td>2010-09-25</td>
<td>MySQL Workbench:</td>
<td>Verified</td>
<td>S5</td>
<td>5.2.28</td>
<td>wb53</td>
<td>Linux</td>
<td>Workbench on Linux doesn't use GPU to</td>
<td></td>
</tr>
</tbody>
</table>
bugs.opensolaris.org “performance”

Bug ID: 4472277 Sun Blade 100 gigabit network performance
network:performance, Sun Blade 100 gigabit network performance < U10 gigabit network performance, State: 11-Closed, Reported: 20-June-2001, Keywords: Blade100 | Gigabit | Network | Performance | ...
bugs.opensolaris.org/bugdatabase/view_bug.do?bug_id=4472277 - 12 KB

Bug ID: 4016979 performance is poor on 64MB SS20 w/local disk
xserver:performance, performance is poor on 64MB SS20 w/local disk, State: 11-Closed, Reported: 27-November-1996, Keywords: cde | leak | memory | performance, Release Reported Against: cde1.1_45 ...
bugs.opensolaris.org/bugdatabase/view_bug.do?bug_id=4016979 - 12 KB

Bug ID: 4317727 Performance regression from Solaris 7 to Solaris 8 in Unigraphics
xserver:performance, Performance regression from Solaris 7 to Solaris 8 in Unigraphics, State: 10-Fix Delivered, Reported: 1-March-2000, Keywords: 8 | Solaris | performance | regression, Release Reported ...
bugs.opensolaris.org/bugdatabase/view_bug.do?bug_id=4317727 - 14 KB

Bug ID: 4653831 non-ON performance utilities should use 64-bit named per-CPU statistics
utility:performance, non-ON performance utilities should use 64-bit named per-CPU statistics, State: 2-Incomplete, Reported: 16-March-2002, Keywords: SAE | observability | performance, Release Reported ...
bugs.opensolaris.org/bugdatabase/view_bug.do?bug_id=4653831 - 12 KB

Bug ID: 4710147 pool extension to performance provider

Bug ID: 4472277 Sun Blade 100 gigabit network performance
network:performance, Sun Blade 100 gigabit network performance < U10 gigabit network performance, State: 11-Closed, Reported: 20-June-2001, Keywords: Blade100 | Gigabit | Network | Performance | ...
bugs.opensolaris.org/bugdatabase/view_bug.do?bug_id=4472277 - 12 KB

Bug ID: 4016979 performance is poor on 64MB SS20 w/local disk
xserver:performance, performance is poor on 64MB SS20 w/local disk, State: 11-Closed, Reported: 27-November-1996, Keywords: cde | leak | memory | performance, Release Reported Against: cde1.1_45 ...
bugs.opensolaris.org/bugdatabase/view_bug.do?bug_id=4016979 - 12 KB

Bug ID: 4317727 Performance regression from Solaris 7 to Solaris 8 in Unigraphics
xserver:performance, Performance regression from Solaris 7 to Solaris 8 in Unigraphics, State: 10-Fix Delivered, Reported: 1-March-2000, Keywords: 8 | Solaris | performance | regression, Release Reported ...
bugs.opensolaris.org/bugdatabase/view_bug.do?bug_id=4317727 - 14 KB

Bug ID: 4653831 non-ON performance utilities should use 64-bit named per-CPU statistics
utility:performance, non-ON performance utilities should use 64-bit named per-CPU statistics, State: 2-Incomplete, Reported: 16-March-2002, Keywords: SAE | observability | performance, Release Reported ...
bugs.opensolaris.org/bugdatabase/view_bug.do?bug_id=4653831 - 12 KB

Bug ID: 4710147 pool extension to performance provider
This list is too long for Bugzilla’s little mind; the Next/Prev/First/Last buttons won’t appear on individual bugs.
“performance” bugs

• ... and those are just the known performance bugs

• ... and usually only of a certain type (architecture)
How to Quantify

• Observation based
  • Choose a reliable metric
  • Estimate performance gain from resolving issue

• Experimentation based
  • Apply fix
  • Quantify before vs. after using a reliable metric
Observation based

• For example:
  • Observed: application I/O takes 10 ms
  • Observed: 9 ms of which is disk I/O
  • Suggestion: replace disks with flash-memory based SSDs, with an expected latency of ~100 us
  • Estimated gain: 10 ms -> 1.1 ms (10 ms - 9 ms + 0.1 ms) =~ 9x gain

• Very useful - but not possible without accurate quantification
Experimentation based

• For example:
  • Observed: Application transaction latency average 10 ms
  • Experiment: Added more DRAM to increase cache hits and reduce average latency
  • Observed: Application transaction latency average 2 ms
  • Gain: 10 ms -> 2 ms = 5x

• Also very useful - but risky without accurate quantification
Metrics to Quantify Performance

• Choose reliable metrics to quantify performance:
  • IOPS
  • transactions/second
  • throughput
  • utilization
  • latency

• Ideally
  • interpretation is straightforward
  • reliable
Choose reliable metrics to quantify performance:

- IOPS
- transactions/second
- throughput
- utilization
- latency

Ideally

- interpretation is straightforward
- reliable

generally better suited for:

Capacity Planning

Workload Analysis
Metrics Availability

• Ideally (given the luxury of time):
  • design the desired metrics
  • then see if they exist, or,
  • implement them (eg, DTrace)

• Non-ideally
  • see what already exists
  • make-do (eg, vmstat -> gnuplot)
Assumptions to avoid

• Available metrics are implemented correctly
  • all software has bugs
    • eg, CR: 6687884 nxge rbytes and obytes kstat are wrong
  • trust no metric without double checking from other sources

• Available metrics are designed by performance experts
  • sometimes added by the programmer to only debug their code

• Available metrics are complete
  • you won’t always find what you really need
Getting technical

• This will be explained using two examples:
  • Workload Analysis
  • Capacity Planning
Example: Workload Analysis

• Quantifying performance issues with IOPS vs latency
  • IOPS is commonly presented by performance analysis tools
  • eg: disk IOPS via kstat, SNMP, iostat, ...
IOPS

- Depends on where the I/O is measured
  - app -> library -> syscall -> VFS -> filesystem -> RAID -> device
- Depends on what the I/O is
  - synchronous or asynchronous
  - random or sequential
  - size
- Interpretation difficult
  - what value is good or bad?
  - is there a max?
Some disk IOPS problems

- IOPS Inflation
  - Library or Filesystem prefetch/read-ahead
  - Filesystem metadata
  - RAID stripes

- IOPS Deflation
  - Read caching
  - Write cancellation
  - Filesystem I/O aggregation

- IOPS aren’t created equal
IOPS example: iostat -xnz 1

• Consider this disk: 86 IOPS == 99% busy

```
extended device statistics
 r/s  w/s  kr/s  kw/s wait actv  wsvc_t asvc_t  %w  %b  device
 86.6  0.0  655.5  0.0  0.0  1.0  0.0  11.5  0  99 c1d0
```

• Versus this disk: 21,284 IOPS == 99% busy

```
extended device statistics
 r/s  w/s  kr/s  kw/s wait actv  wsvc_t asvc_t  %w  %b  device
21284.7  0.0 10642.4  0.0  0.0  1.8  0.0  0.1  2  99 c1d0
```
**IOPS example: iostat -xnz 1**

- Consider this disk: 86 IOPS == 99% busy

  ```
  extended device statistics
  r/s   w/s  kr/s  kw/s wait  actv  wsvc_t  asvc_t  %w  %b  device
  86.6  0.0   655.5 0.0 0.0 1.0   0.0   11.5 0  99  c1d0
  ```

- Versus this disk: 21,284 IOPS == 99% busy

  ```
  extended device statistics
  r/s   w/s  kr/s  kw/s wait  actv  wsvc_t  asvc_t  %w  %b  device
  21284.7 0.0 10642.4 0.0 0.0 1.8   0.0   0.1 2  99  c1d0
  ```

- ... they are the same disk, different I/O types
  - 1) 8 Kbyte random
  - 2) 512 byte sequential (on-disk DRAM cache)
Using IOPS to quantify issues

- to identify
  - is 100 IOPS an problem? Per disk?
- to locate
  - 90% of IOPS are random. Is that the problem?
- to verify
  - A filesystem tunable caused IOPS to reduce. Has this fixed the issue?
Using IOPS to quantify issues

• to identify
  • is 100 IOPS an problem? Per disk? (depends...)

• to locate
  • 90% of IOPS are random. Is that the problem? (depends...)

• to verify
  • A filesystem tunable caused IOPS to reduce. Has this fixed the issue? (probably, assuming...)

• We can introduce more metrics to understand these, but standalone IOPS is tricky to interpret
Using latency to quantify issues

• to identify
  • is a 100ms I/O a problem?

• to locate
  • 90ms of the 100ms is lock contention. Is that the problem?

• to verify
  • A filesystem tunable caused the I/O latency to reduce to 1ms. Has this fixed the issue?
Using latency to quantify issues

• to identify
  • is a 100ms I/O a problem? (probably - if synchronous to the app.)

• to locate
  • 90ms of the 100ms is lock contention. Is that the problem? (yes)

• to verify
  • A filesystem tunable caused the I/O latency to reduce to 1ms. Has this fixed the issue? (probably - if 1ms is acceptable)

• Latency is much more reliable, easier to interpret
Latency

• Time from I/O or transaction request to completion

• Synchronous latency has a direct impact on performance
  • Application is waiting
  • higher latency == worse performance

• Not all latency is synchronous:
  • Asynchronous filesystem threads flushing dirty buffers to disk eg, zfs TXG synchronous thread
  • Filesystem prefetch
    no one is waiting at this point
  • TCP buffer and congestion window: individual packet latency may be high, but pipe is kept full for good throughput performance
Turning other metrics into latency

• Currency converter (* -> ms):
  • random disk IOPS == I/O service latency
  • disk saturation == I/O wait queue latency
  • CPU utilization == code path execution latency
  • CPU saturation == dispatcher queue latency
  • ...

• Quantifying as latency allows different components to be compared, ratios examined, improvements estimated.
Example: Resource Monitoring

- Different performance activity
  - Focus is environment components, not specific issues
  - incl. CPUs, disks, network interfaces, memory, I/O bus, memory bus, CPU interconnect, I/O cards, network switches, etc.
  - Information is used for capacity planning
  - Identifying future issues before they happen

- Quantifying resource monitoring with IOPS vs utilization
IOPS vs Utilization

• Another look at this disk:

    extended device statistics
    r/s  w/s  kr/s  kw/s wait actv wsrv_t asrc_t %w %b device
     86.6   0.0  655.5  0.0  0.0  1.0  0.0  11.5  0 99 c1d0

    [...]

    extended device statistics
    r/s  w/s  kr/s  kw/s wait actv wsrv_t asrc_t %w %b device
   21284.7  0.0  10642.4  0.0  0.0  1.8  0.0  0.1  2 99 c1d0

• Q. does this system need more spindles for IOPS capacity?
IOPS vs Utilization

• Another look at this disk:

```
extended device statistics
r/s  w/s  kr/s  kw/s  wait  actv  wsvc_t asvc_t  %w  %b  device
86.6  0.0  655.5  0.0  0.0  1.0   0.0  11.5  0  99  c1d0
```

[...]

```
extended device statistics
r/s  w/s  kr/s  kw/s  wait  actv  wsvc_t asvc_t  %w  %b  device
21284.7  0.0 10642.4  0.0  0.0  1.8   0.0  0.1  2  99  c1d0
```

• Q. does this system need more spindles for IOPS capacity?

  • IOPS (r/s + w/s): ???
  
  • Utilization (%b): yes (even considering NCQ)
IOPS vs Utilization

• Another look at this disk:

  extended device statistics
  r/s   w/s   kr/s   kw/s wait actv wsvc_t asvc_t %w %b device
  86.6  0.0   655.5  0.0   0.0  1.0   0.0   11.5  0   99 c1d0
  [...]

  extended device statistics
  r/s   w/s   kr/s   kw/s wait actv wsvc_t asvc_t %w %b device
  21284.7 0.0 10642.4 0.0   0.0  1.8   0.0   0.1  2   99 c1d0

• Q. does this system need more spindles for IOPS capacity?
  
  • IOPS (r/s + w/s): ???
  
  • Utilization (%b): yes (even considering NCQ)
  
  • Latency (wsvc_t): no

• Latency will identify the issue once it is an issue; utilization will forecast the issue - capacity planning
Performance Summary

- Metrics matter - need to reliably quantify performance
  - to identify, locate, verify
  - try to think, design
- Workload Analysis
  - latency
- Resource Monitoring
  - utilization
- Other metrics are useful to further understand the nature of the workload and resource behavior
Objectives

- Consider performance metrics before plotting
  - Why latency is good
  - ... and IOPS can be bad
- See the value of visualizations
  - Why heat maps are needed
  - ... and line graphs can be bad
- Remember key examples
  - I/O latency, as a heat map
  - CPU utilization by CPU, as a heat map
Visualizations

Current Examples

Latency
Visualizations

- So far we’ve picked:
  - Latency
    - for workload analysis
  - Utilization
    - for resource monitoring
Latency

• For example, disk I/O

• Raw data looks like this:

```
# iosnoop -o

<table>
<thead>
<tr>
<th>DTIME</th>
<th>UID</th>
<th>PID</th>
<th>D</th>
<th>BLOCK</th>
<th>SIZE</th>
<th>COMM</th>
<th>PATHNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>125</td>
<td>100</td>
<td>337</td>
<td>R</td>
<td>72608</td>
<td>8192</td>
<td>bash</td>
<td>/usr/sbin/tar</td>
</tr>
<tr>
<td>138</td>
<td>100</td>
<td>337</td>
<td>R</td>
<td>72624</td>
<td>8192</td>
<td>bash</td>
<td>/usr/sbin/tar</td>
</tr>
<tr>
<td>127</td>
<td>100</td>
<td>337</td>
<td>R</td>
<td>72640</td>
<td>8192</td>
<td>bash</td>
<td>/usr/sbin/tar</td>
</tr>
<tr>
<td>135</td>
<td>100</td>
<td>337</td>
<td>R</td>
<td>72656</td>
<td>8192</td>
<td>bash</td>
<td>/usr/sbin/tar</td>
</tr>
<tr>
<td>118</td>
<td>100</td>
<td>337</td>
<td>R</td>
<td>72672</td>
<td>8192</td>
<td>bash</td>
<td>/usr/sbin/tar</td>
</tr>
<tr>
<td>108</td>
<td>100</td>
<td>337</td>
<td>R</td>
<td>72688</td>
<td>4096</td>
<td>bash</td>
<td>/usr/sbin/tar</td>
</tr>
<tr>
<td>87</td>
<td>100</td>
<td>337</td>
<td>R</td>
<td>72696</td>
<td>3072</td>
<td>bash</td>
<td>/usr/sbin/tar</td>
</tr>
<tr>
<td>9148</td>
<td>100</td>
<td>337</td>
<td>R</td>
<td>113408</td>
<td>8192</td>
<td>tar</td>
<td>/etc/default/lu</td>
</tr>
<tr>
<td>8806</td>
<td>100</td>
<td>337</td>
<td>R</td>
<td>104738</td>
<td>7168</td>
<td>tar</td>
<td>/etc/default/lu</td>
</tr>
<tr>
<td>2262</td>
<td>100</td>
<td>337</td>
<td>R</td>
<td>13600</td>
<td>1024</td>
<td>tar</td>
<td>/etc/default/cron</td>
</tr>
<tr>
<td>76</td>
<td>100</td>
<td>337</td>
<td>R</td>
<td>13616</td>
<td>1024</td>
<td>tar</td>
<td>/etc/default/devfsadm</td>
</tr>
</tbody>
</table>

[...many pages of output...]
```

• `iosnoop` is DTrace based

  • examines latency for every disk (back end) I/O
Latency Data

- tuples
  - I/O completion time
  - I/O latency
- can be 1,000s of these per second
Summarizing Latency

- `iostat(1M)` can show **per second average**:

```
$ iostat -xnz 1
[...

extended device statistics
 r/s  w/s  kr/s  kw/s  wait  actv  wsvc_t  asvc_t  %w  %b  device
 471.0  7.0  786.1  12.0  0.1  1.2  0.2  2.5  4  90  c1d0
extended device statistics
 r/s  w/s  kr/s  kw/s  wait  actv  wsvc_t  asvc_t  %w  %b  device
 631.0  0.0 1063.1  0.0  0.2  1.0  0.3  1.6  9  92  c1d0
extended device statistics
 r/s  w/s  kr/s  kw/s  wait  actv  wsvc_t  asvc_t  %w  %b  device
 472.0  0.0  529.0  0.0  0.0  1.0  0.0  2.1  0  94  c1d0
[...]
```
Per second

- Condenses I/O completion time
- Almost always a sufficient resolution
  - (So far I’ve only had one case where examining raw completion
time data was crucial: an interrupt coalescing bug)
Average/second

- Average loses latency outliers
- Average loses latency distribution
- ... but not disk distribution:

```bash
$ iostat -xnz 1
[...]
```

<table>
<thead>
<tr>
<th>r/s</th>
<th>w/s</th>
<th>kr/s</th>
<th>kw/s</th>
<th>wait</th>
<th>activ</th>
<th>wsvc_t</th>
<th>asvc_t</th>
<th>%w</th>
<th>%b</th>
<th>device</th>
</tr>
</thead>
<tbody>
<tr>
<td>43.9</td>
<td>0.0</td>
<td>351.5</td>
<td>0.0</td>
<td>0.0</td>
<td>0.4</td>
<td>0.0</td>
<td>10.0</td>
<td>0</td>
<td>34</td>
<td>c0t5000CCA215C46459d0</td>
</tr>
<tr>
<td>47.6</td>
<td>0.0</td>
<td>381.1</td>
<td>0.0</td>
<td>0.0</td>
<td>0.5</td>
<td>0.0</td>
<td>9.8</td>
<td>0</td>
<td>36</td>
<td>c0t5000CCA215C4521Dd0</td>
</tr>
<tr>
<td>42.7</td>
<td>0.0</td>
<td>349.9</td>
<td>0.0</td>
<td>0.0</td>
<td>0.4</td>
<td>0.0</td>
<td>10.1</td>
<td>0</td>
<td>35</td>
<td>c0t5000CCA215C45F89d0</td>
</tr>
<tr>
<td>41.4</td>
<td>0.0</td>
<td>331.5</td>
<td>0.0</td>
<td>0.0</td>
<td>0.4</td>
<td>0.0</td>
<td>9.6</td>
<td>0</td>
<td>32</td>
<td>c0t5000CCA215C42A4Cd0</td>
</tr>
<tr>
<td>45.6</td>
<td>0.0</td>
<td>365.1</td>
<td>0.0</td>
<td>0.0</td>
<td>0.4</td>
<td>0.0</td>
<td>9.2</td>
<td>0</td>
<td>34</td>
<td>c0t5000CCA215C45541d0</td>
</tr>
<tr>
<td>45.0</td>
<td>0.0</td>
<td>360.3</td>
<td>0.0</td>
<td>0.0</td>
<td>0.4</td>
<td>0.0</td>
<td>9.4</td>
<td>0</td>
<td>34</td>
<td>c0t5000CCA215C458F1d0</td>
</tr>
<tr>
<td>42.9</td>
<td>0.0</td>
<td>343.5</td>
<td>0.0</td>
<td>0.0</td>
<td>0.4</td>
<td>0.0</td>
<td>9.9</td>
<td>0</td>
<td>33</td>
<td>c0t5000CCA215C450E3d0</td>
</tr>
<tr>
<td>44.9</td>
<td>0.0</td>
<td>359.5</td>
<td>0.0</td>
<td>0.0</td>
<td>0.4</td>
<td>0.0</td>
<td>9.3</td>
<td>0</td>
<td>35</td>
<td>c0t5000CCA215C45323d0</td>
</tr>
</tbody>
</table>
[...]

- only because iostat(1M) prints this per-disk
  - but that gets hard to read for 100s of disks, per second!
Latency outliers

• Occasional high-latency I/O
• Can be the sole reason for performance issues
• Can be lost in an average
  • 10,000 fast I/O @ 1ms
  • 1 slow I/O @ 500ms
  • average = 1.05 ms
• Can be seen using max instead of (or as well as) average
Maximum/second

- `iostat(1M)` doesn’t show this, however DTrace can
- can be visualized along with average/second
- does identify outliers
- doesn’t identify latency distribution details
Latency distribution

• Apart from outliers and average, it can be useful to examine the full profile of latency - all the data.
  • For such a crucial metric, keep as much details as possible

• For latency, distributions we’d expect to see include:
  • bi-modal: cache hit vs cache miss
  • tri-modal: multiple cache layers
  • flat: random disk I/O
# Latency Distribution Example

- **Using DTrace:**

```bash
# ./disklatency.d
Tracing... Hit Ctrl-C to end.
^C
sd4 (28,256), us:

<table>
<thead>
<tr>
<th>value</th>
<th>------------- Distribution -------------</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>32</td>
<td>@@</td>
<td>82</td>
</tr>
<tr>
<td>64</td>
<td>@@@</td>
<td>621</td>
</tr>
<tr>
<td>128</td>
<td>@@@@@</td>
<td>833</td>
</tr>
<tr>
<td>256</td>
<td>@@@@@</td>
<td>641</td>
</tr>
<tr>
<td>512</td>
<td>@@@</td>
<td>615</td>
</tr>
<tr>
<td>1024</td>
<td>@@@@@@</td>
<td>1239</td>
</tr>
<tr>
<td>2048</td>
<td>@@@@@@@@@</td>
<td>1615</td>
</tr>
<tr>
<td>4096</td>
<td>@@@@@@@@@@</td>
<td>1483</td>
</tr>
<tr>
<td>8192</td>
<td>@@@@@@@@@@@@@</td>
<td>76</td>
</tr>
<tr>
<td>16384</td>
<td>@@@@@@@@@@@@@@</td>
<td>1</td>
</tr>
<tr>
<td>32768</td>
<td>@@@@@@@@@@@@@@@@@</td>
<td>0</td>
</tr>
<tr>
<td>65536</td>
<td>@@@@@@@@@@@@@@@@@@</td>
<td>2</td>
</tr>
<tr>
<td>131072</td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>
```
not why we are here, but before someone asks...

```c
#!/usr/sbin/dtrace -s

#pragma D option quiet

dtrace:::BEGIN
{
    printf("Tracing... Hit Ctrl-C to end.\n");
}

io:::start
{
    start_time[arg0] = timestamp;
}

io:::done
/this->start = start_time[arg0]/
{
    this->delta = (timestamp - this->start) / 1000;
    @[args[1]->dev_statname, args[1]->dev_major, args[1]->dev_minor] = quantize(this->delta);
    start_time[arg0] = 0;
}

dtrace:::END
{
    printa(" %s (%d,%d), us:@d\n", @);
}
```
Latency Distribution Example

```bash
# ./disklatency.d
Tracing... Hit Ctrl-C to end.
^C
sd4 (28,256), us:

<table>
<thead>
<tr>
<th>value</th>
<th>Distribution</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>32</td>
<td></td>
<td>82</td>
</tr>
<tr>
<td>64</td>
<td>@@@</td>
<td>621</td>
</tr>
<tr>
<td>128</td>
<td>@@@@@@</td>
<td>833</td>
</tr>
<tr>
<td>256</td>
<td>@@@@</td>
<td>641</td>
</tr>
<tr>
<td>512</td>
<td>@@@</td>
<td>615</td>
</tr>
<tr>
<td>1024</td>
<td>@@@@@@@</td>
<td>1239</td>
</tr>
<tr>
<td>2048</td>
<td>@@@@@@@@@@</td>
<td>1615</td>
</tr>
<tr>
<td>4096</td>
<td>@@@@@@@@@@</td>
<td>1483</td>
</tr>
<tr>
<td>8192</td>
<td>@@@@@@@@@@@</td>
<td>76</td>
</tr>
<tr>
<td>16384</td>
<td>@@@@@@@@@@@</td>
<td>1</td>
</tr>
<tr>
<td>32768</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>65536</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>131072</td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>
```

- ... but can we see this distribution per second?
- ... how do we visualize a 3rd dimension?
Column Quantized Visualization
aka “heat map”

• For example:
Heat Map: Offset Distribution

- x-axis: time
- y-axis: offset
- z-axis (color scale): I/O count for that time/offset range

- Identified random vs. sequential very well
- Similar heat maps have been used before by defrag tools
Heat Map: Latency Distribution

• For example:

- x-axis: time
- y-axis: latency
- z-axis (color saturation): I/O count for that time/latency range
... in fact, this is a great example:

- reads served from:
  - DRAM disk
  - disk
  - flash-memory based SSD disk
  - ZFS “L2ARC” enabled
Heat Map: Latency Distribution

- ... in fact, this is a great example:

  - reads served from:

    - DRAM disk
    - flash-memory based SSD

  ZFS “L2ARC” enabled
Latency Heat Map

• A color shaded matrix of pixels
• Each pixel is a time and latency range
• Color shade picked based on number of I/O in that range
• Adjusting saturation seems to work better than color hue.

Eg:
• darker == more I/O
• lighter == less I/O
Pixel Size

• Large pixels (and corresponding time/latency ranges)
  • increases likelihood that adjacent pixels include I/O, have color, and combine to form patterns
  • allows color to be more easily seen

• Smaller pixels (and time/latency ranges)
  • can make heat map look like a scatter plot
  • of the same color - if ranges are so small only one I/O is typically included
Color Palette

• Linear scale can make subtle details (outliers) difficult to see
  • observing latency outliers is usually of high importance
  • outliers are usually < 1% of the I/O
  • assigning < 1% of the color scale to them will washout patterns

• False color palette can be used to emphasize these details
  • although color comparisons become more confusing - non-linear
Outliers

- Heat maps show these very well
- However, latency outliers can compress the bulk of the heat map data
  - eg, 1 second outlier while most I/O is < 10 ms
- Users should have some control to be able to zoom/truncate details
  - both x and y axis
Data Storage

• Since heat-maps are three dimensions, storing this data can become costly (volume)

• Most of the data points are zero
  • and you can prevent storing zero’s by only storing populated elements: associative array

• You can reduce to a sufficiently high resolution, and resample lower as needed

• You can also be aggressive at reducing resolution at higher latencies
  • 10 us granularity not as interesting for I/O > 1 second
  • non-linear resolution
Other Interesting Latency Heat Maps

• The “Icy Lake”
• The “Rainbow Pterodactyl”
• Latency Levels
The “Icy Lake” Workload

• About as simple as it gets:

  • Single client, single thread, sequential synchronous 8 Kbyte writes to an NFS share

  • NFS server: 22 x 7,200 RPM disks, striped pool

• The resulting latency heat map was unexpected
The “Icy Lake”
“Icy Lake” Analysis: Observation

- Examining single disk latency:

- Pattern match with NFS latency: similar lines
  - each disk contributing some lines to the overall pattern
Pattern Match?

• We just associated NFS latency with disk device latency, using our eyeballs
  • see the titles on the previous heat maps

• You can programmatically do this (DTrace), but that can get difficult to associate context across software stack layers (but not impossible!)

• Heat Maps allow this part of the problem to be offloaded to your brain
  • and we are very good at pattern matching
“Icy Lake” Analysis: Experimentation

• Same workload, single disk pool:

• No diagonal lines

  • but more questions - see the line (false color palette enhanced) at 9.29 ms? this is < 1% of the I/O. (I’m told, and I believe, that this is due to adjacent track seek latency.)
“Icy Lake” Analysis: Experimentation

- Same workload, two disk striped pool:

  - Ah-hah! Diagonal lines.

    - ... but still more questions: why does the angle sometimes change? why do some lines slope upwards and some down?
“Icy Lake” Analysis: Experimentation

- ... each disk from that pool:
“Icy Lake” Analysis: Questions

• Remaining Questions:
  • Why does the slope sometimes change?
  • What exactly seeds the slope in the first place?
"Icy Lake" Analysis: Mirroring

• Trying mirroring the pool disks instead of striping:
Another Example: “X marks the spot”
The “Rainbow Pterodactyl” Workload

• 48 x 7,200 RPM disks, 2 disk enclosures

• Sequential 128 Kbyte reads to each disk (raw device), adding disks every 2 seconds

• Goal: Performance analysis of system architecture

  • identifying I/O throughput limits by driving I/O subsystem to saturation, one disk at a time (finds knee points)
The “Rainbow Pterodactyl”
The “Rainbow Pterodactyl”

- Buldge
- Wing
- Shoulders
- Body

- Beak
- Head
- Neck
The “Rainbow Pterodactyl”: Analysis

• Hasn’t been understood in detail
  • Would never be understood (or even known) without heat maps
• It is repeatable
The “Rainbow Pterodactyl”: Theories

- “Beak”: disk cache hit vs disk cache miss -> bimodal
- “Head”: 9th disk, contention on the 2 x4 SAS ports
- “Buldge”: ?
- “Neck”: ?
- “Wing”: contention?
- “Shoulders”: ?
- “Body”: PCI-gen1 bus contention
• Same as “Rainbow Pterodactyl”, stepping disks

• Instead of sequential reads, this is repeated 128 Kbyte reads (read -> seek 0 -> read -> ...), to deliberately hit from the disk DRAM cache to improve test throughput
Latency Levels

Disk: I/O operations per second broken down by latency

At 15:40:16:
1 7.29 ms
4 7.14 ms
819 7.00 ms
16 6.86 ms
1 6.14 ms
98 6.00 ms
610 5.88 ms
123 5.71 ms
1 5.00 ms
3 4.88 ms
271 4.71 ms
3447 4.57 ms

5394 ops per second

Disk: I/O bytes per second broken down by disk

At 15:40:16:
210M /dev/sdb 40
209M /dev/sdb 22
208M /dev/sdb 7
208M /dev/sdb 3
207M /dev/sdb 30
207M /dev/sdb 18
207M /dev/sdb 46

Show hierarchy
5.27G per second

88
Latency Levels Theories

• ???
• This time we do know the source of the latency...
硬碟也會鬧情緒

癲科技小劇場：硬碟也會鬧情緒

由 Flow Yu 於 1 year 之前發表

文章分類：儲存裝置

這可不是開玩笑來著，而是經過實驗證明的唷！來自 Sun Fishworks 團隊的 Brendan Gregg，或許是在機房裡待久了，於是突發異想跑去對機房中的磁碟陣列大吼大叫一陣，結果就這麼發現了一個真理，那就是：硬碟就跟員工一樣，吼叫並不會讓他們的效率變高，反而還會讓它們心生不爽而降低士氣，私底下就開始搞罷工。
Latency Heat Maps: Summary

• Shows latency distribution over time
• Shows outliers (maximums)
• Indirectly shows average
• Shows patterns
  • allows correlation with other software stack layers
Similar Heat Map Uses

• These all have a dynamic y-axis scale:
  • I/O size
  • I/O offset

• These aren’t a primary measure of performance (like latency); they provide secondary information to understand the workload
Heat Map: I/O Offset

- y-axis: I/O offset (in this case, NFSv3 file location)
Heat Map: I/O Size

- y-axis: I/O size (bytes)
Heat Map Abuse

• What can we ‘paint’ by adjusting the workload?
I/O Size

How was this done?
• How was this done?
• How was this done?
Visualizations

Current Examples

Utilization
## CPU Utilization

- Commonly used indicator of CPU performance
- eg, `vmstat(1M)`

```bash
$ vmstat 1 5

kthr memory page disk faults cpu
r b w swap free re mf pi po fr de sr s0 s1 s2 s3 in sy cs us sy id
0 0 0 95125264 28022732 301 1742 1 17 17 0 0 -0 -0 -0 6 5008 21927 3886 4 1 94
0 0 0 91512024 25075924 6 55 0 0 0 0 0 0 0 0 0 4665 18228 4299 10 1 89
0 0 0 91511864 25075796 9 24 0 0 0 0 0 0 0 0 0 0 3504 12757 3158 8 0 92
0 0 0 91511228 25075164 3 163 0 0 0 0 0 0 0 0 0 4104 15375 3611 9 5 86
0 0 0 91510824 25074940 5 66 0 0 0 0 0 0 0 0 0 4607 19492 4394 10 1 89
```
CPU Utilization: Line Graph

- Easy to plot:
CPU Utilization: Line Graph

• Easy to plot:

• Average across all CPUs:
  • Identifies how utilized all CPUs are, indicating remaining headroom - provided sufficient threads to use CPUs
### CPU Utilization by CPU

- *mpstat(1M)* can show utilization by-CPU:

```bash
$ mpstat 1
[...]
```

<table>
<thead>
<tr>
<th>CPU</th>
<th>minf</th>
<th>mjf</th>
<th>xcal</th>
<th>intr</th>
<th>ithr</th>
<th>cswh</th>
<th>icswh</th>
<th>migr</th>
<th>smtx</th>
<th>srwh</th>
<th>syscl</th>
<th>usr</th>
<th>sys</th>
<th>wt</th>
<th>idl</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>313</td>
<td>105</td>
<td>315</td>
<td>0</td>
<td>24</td>
<td>4</td>
<td>0</td>
<td>1331</td>
<td>5</td>
<td>1</td>
<td>0</td>
<td>94</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>65</td>
<td>28</td>
<td>190</td>
<td>0</td>
<td>12</td>
<td>4</td>
<td>0</td>
<td>576</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>98</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>64</td>
<td>20</td>
<td>152</td>
<td>0</td>
<td>12</td>
<td>1</td>
<td>0</td>
<td>438</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>99</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>127</td>
<td>74</td>
<td>274</td>
<td>1</td>
<td>21</td>
<td>3</td>
<td>0</td>
<td>537</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>98</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>32</td>
<td>5</td>
<td>229</td>
<td>0</td>
<td>9</td>
<td>2</td>
<td>0</td>
<td>902</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>98</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>46</td>
<td>19</td>
<td>138</td>
<td>0</td>
<td>7</td>
<td>3</td>
<td>0</td>
<td>521</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>99</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>109</td>
<td>32</td>
<td>296</td>
<td>0</td>
<td>8</td>
<td>2</td>
<td>0</td>
<td>1266</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>96</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>30</td>
<td>8</td>
<td>0</td>
<td>9</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>169</td>
<td>68</td>
<td>311</td>
<td>0</td>
<td>22</td>
<td>2</td>
<td>0</td>
<td>847</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>97</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>30</td>
<td>111</td>
<td>54</td>
<td>274</td>
<td>0</td>
<td>16</td>
<td>4</td>
<td>0</td>
<td>868</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>69</td>
<td>29</td>
<td>445</td>
<td>0</td>
<td>13</td>
<td>7</td>
<td>0</td>
<td>2559</td>
<td>7</td>
<td>1</td>
<td>0</td>
<td>92</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>78</td>
<td>36</td>
<td>303</td>
<td>0</td>
<td>7</td>
<td>8</td>
<td>0</td>
<td>1041</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>98</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>74</td>
<td>34</td>
<td>312</td>
<td>0</td>
<td>10</td>
<td>1</td>
<td>0</td>
<td>1250</td>
<td>7</td>
<td>1</td>
<td>0</td>
<td>92</td>
</tr>
<tr>
<td>13</td>
<td>38</td>
<td>0</td>
<td>15</td>
<td>456</td>
<td>285</td>
<td>336</td>
<td>2</td>
<td>10</td>
<td>1</td>
<td>0</td>
<td>1408</td>
<td>5</td>
<td>2</td>
<td>0</td>
<td>93</td>
</tr>
<tr>
<td>14</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2620</td>
<td>2497</td>
<td>209</td>
<td>0</td>
<td>10</td>
<td>38</td>
<td>0</td>
<td>259</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>96</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>20</td>
<td>8</td>
<td>10</td>
<td>0</td>
<td>4</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100</td>
</tr>
</tbody>
</table>
```

- can identify a single hot CPU (thread)
  - and un-balanced configurations
CPU Resource Monitoring

- Monitor overall utilization for capacity planning
- Also valuable to monitor individual CPUs
  - can identify un-balanced configurations
  - such as a single hot CPU (thread)
- The virtual CPUs on a single host can now reach the 100s
  - its own dimension
  - how can we display this 3rd dimension?
Heat Map: CPU Utilization

- x-axis: time
- y-axis: percent utilization
- z-axis (color saturation): # of CPUs in that time/utilization range
Heat Map: CPU Utilization

- Single ‘hot’ CPUs are a common problem due to application scaleability issues (single threaded)
- This makes identification easy, without reading pages of mpstat(1M) output
Heat Map: Disk Utilization

• Ditto for disks

• Disk Utilization heat map can identify:
  • overall utilization
  • unbalanced configurations
  • single hot disks (versus all disks busy)

• Ideally, the disk utilization heat map is tight (y-axis) and below 70%, indicating a well balanced config with headroom
  • which can’t be visualized with line graphs
• Are typically used to visualize performance, be it IOPS or utilization

• Show patterns over time more clearly than text (higher resolution)

• But graphical environments can do much more
  
  • As shown by the heat maps (to start with); which convey details line graphs cannot

• Ask: what “value add” does the GUI bring to the data?
Resource Utilization Heat Map Summary

- Can exist for any resource with multiple components:
  - CPUs
  - Disks
  - Network interfaces
  - I/O busses
  - ...
- Quickly identifies single hot component versus all components
- Best suited for physical hardware resources
  - difficult to express ‘utilization’ for a software resource
Visualizations

Future Opportunities
So far analysis has been for a single server

What about the cloud?
From one to thousands of servers

Workload Analysis:
latency I/O x cloud

Resource Monitoring:
# of CPUs x cloud
# of disks x cloud
etc.
Heat Maps for the Cloud

- Heat Maps are promising for cloud computing observability:
  - additional dimension accommodates the scale of the cloud
- Find outliers regardless of node
  - cloud-wide latency heat map just has more I/O
- Examine how applications are load balanced across nodes
  - similar to CPU and disk utilization heat maps
- mpstat and iostat’s output are already getting too long
  - multiply by 1000x for the number of possible hosts in a large cloud application
Proposed Visualizations

• Include:
  • Latency heat map across entire cloud
  • Latency heat maps for cloud application components
  • CPU utilization by cloud node
  • CPU utilization by CPU
  • Thread/process utilization across entire cloud
  • Network interface utilization by cloud node
  • Network interface utilization by port
  • lots, lots more
Cloud Latency Heat Map

- Latency at different layers:
  - Apache
  - PHP/Ruby/...
  - MySQL
  - DNS
  - Disk I/O
  - CPU dispatcher queue latency
  - and pattern match to quickly identify and locate latency
### Latency Example: MySQL

- **Query latency (DTrace):**

<table>
<thead>
<tr>
<th>query time (ns)</th>
<th>value</th>
<th>Distribution</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1024</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>2048</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>4096</td>
<td>@</td>
<td>99</td>
</tr>
<tr>
<td></td>
<td>8192</td>
<td></td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>16384</td>
<td>@</td>
<td>114</td>
</tr>
<tr>
<td></td>
<td>32768</td>
<td>@</td>
<td>105</td>
</tr>
<tr>
<td></td>
<td>65536</td>
<td>@</td>
<td>123</td>
</tr>
<tr>
<td></td>
<td>131072</td>
<td>@@@@@@@@@@@@@@</td>
<td>1726</td>
</tr>
<tr>
<td></td>
<td>262144</td>
<td>@@@@@@@@@@@@@</td>
<td>1515</td>
</tr>
<tr>
<td></td>
<td>524288</td>
<td>@@@@</td>
<td>601</td>
</tr>
<tr>
<td></td>
<td>1048576</td>
<td>@@</td>
<td>282</td>
</tr>
<tr>
<td></td>
<td>2097152</td>
<td>@</td>
<td>114</td>
</tr>
<tr>
<td></td>
<td>4194304</td>
<td>@</td>
<td>61</td>
</tr>
<tr>
<td></td>
<td>8388608</td>
<td>@@@@</td>
<td>660</td>
</tr>
<tr>
<td></td>
<td>16777216</td>
<td>@@@@</td>
<td>67</td>
</tr>
<tr>
<td></td>
<td>33554432</td>
<td>@@@@</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>67108864</td>
<td>@@@@</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>134217728</td>
<td>@@@@</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>268435456</td>
<td>@@@@</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>536870912</td>
<td>@@@@</td>
<td>0</td>
</tr>
</tbody>
</table>
Latency Example: MySQL

- Query latency (DTrace):

<table>
<thead>
<tr>
<th>query time (ns)</th>
<th>value</th>
<th>Distribution</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024</td>
<td>1024</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>2048</td>
<td>2048</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>4096</td>
<td>4096</td>
<td>@</td>
<td>99</td>
</tr>
<tr>
<td>8192</td>
<td>8192</td>
<td></td>
<td>20</td>
</tr>
<tr>
<td>16384</td>
<td>16384</td>
<td>@</td>
<td>114</td>
</tr>
<tr>
<td>32768</td>
<td>32768</td>
<td>@</td>
<td>105</td>
</tr>
<tr>
<td>65536</td>
<td>65536</td>
<td>@</td>
<td>123</td>
</tr>
<tr>
<td>131072</td>
<td>131072</td>
<td>@@@@@@@@@@@@@</td>
<td>1726</td>
</tr>
<tr>
<td>262144</td>
<td>262144</td>
<td>@@@@@@@@@@@@@</td>
<td>1515</td>
</tr>
<tr>
<td>524288</td>
<td>524288</td>
<td>@@@</td>
<td>601</td>
</tr>
<tr>
<td>1048576</td>
<td>1048576</td>
<td>@@</td>
<td>282</td>
</tr>
<tr>
<td>2097152</td>
<td>2097152</td>
<td>@</td>
<td>114</td>
</tr>
<tr>
<td>4194304</td>
<td>4194304</td>
<td>@</td>
<td>61</td>
</tr>
<tr>
<td>8388608</td>
<td>8388608</td>
<td>@@@@@@</td>
<td>660</td>
</tr>
<tr>
<td>16777216</td>
<td>16777216</td>
<td>@@@@@@</td>
<td>67</td>
</tr>
<tr>
<td>33554432</td>
<td>33554432</td>
<td>@</td>
<td>12</td>
</tr>
<tr>
<td>67108864</td>
<td>67108864</td>
<td>@</td>
<td>7</td>
</tr>
<tr>
<td>134217728</td>
<td>134217728</td>
<td>@</td>
<td>4</td>
</tr>
<tr>
<td>268435456</td>
<td>268435456</td>
<td>@</td>
<td>5</td>
</tr>
<tr>
<td>536870912</td>
<td>536870912</td>
<td>@</td>
<td>0</td>
</tr>
</tbody>
</table>

What is this? (8-16 ms latency)
### Latency Example: MySQL

#### query time (ns)

<table>
<thead>
<tr>
<th>value</th>
<th>-------------- Distribution --------------</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>2048</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>2048</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>4096</td>
<td>@</td>
<td>99</td>
</tr>
<tr>
<td>8192</td>
<td></td>
<td>20</td>
</tr>
<tr>
<td>16384</td>
<td>@</td>
<td>114</td>
</tr>
<tr>
<td>32768</td>
<td>@</td>
<td>105</td>
</tr>
<tr>
<td>65536</td>
<td>@</td>
<td>123</td>
</tr>
<tr>
<td>131072</td>
<td>@@@@@@@@@@@@@@@@</td>
<td>1726</td>
</tr>
<tr>
<td>262144</td>
<td>@@@@@@@@@@@@@@@@</td>
<td>1515</td>
</tr>
<tr>
<td>524288</td>
<td>@@@@@@@@</td>
<td>601</td>
</tr>
<tr>
<td>1048576</td>
<td>@@@@</td>
<td>282</td>
</tr>
<tr>
<td>2097152</td>
<td>@@</td>
<td>114</td>
</tr>
<tr>
<td>4194304</td>
<td>@@@@</td>
<td>61</td>
</tr>
<tr>
<td>8388608</td>
<td>@@@@@@</td>
<td>660</td>
</tr>
<tr>
<td>16777216</td>
<td>@@@@@</td>
<td>67</td>
</tr>
<tr>
<td>33554432</td>
<td>@@@</td>
<td>12</td>
</tr>
<tr>
<td>67108864</td>
<td>@@</td>
<td>7</td>
</tr>
<tr>
<td>134217728</td>
<td>@</td>
<td>4</td>
</tr>
<tr>
<td>268435456</td>
<td>@</td>
<td>5</td>
</tr>
<tr>
<td>536870912</td>
<td>@</td>
<td>0</td>
</tr>
</tbody>
</table>

#### innodb srv sleep (ns)

<table>
<thead>
<tr>
<th>value</th>
<th>-------------- Distribution --------------</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>4194304</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>8388608</td>
<td>@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@</td>
<td>841</td>
</tr>
<tr>
<td>16777216</td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>
Latency Example: MySQL

- Spike of MySQL query latency: 8 - 16 ms
- innodb thread concurrency back-off sleep latency: 8 - 16 ms
- Both have a similar magnitude (see “count” column)
- Add the dimension of time as a heat map, for more characteristics to compare
- ... quickly compare heat maps from different components of the cloud to pattern match and locate latency
Cloud Latency Heat Map

- Identify latency outliers, distributions, patterns
- Can add more functionality to identify these by:
  - cloud node
  - application, cloud-wide
  - I/O type (eg, query type)
- Targeted observability (DTrace) can be used to fetch this
- Or, we could collect it for everything
  - ... do we need a 4th dimension?
4th Dimension!

- Bryan Cantrill @Joyent coded this 11 hours ago
  - assuming it’s now about 10:30am during this talk
  - ... and I added these slides about 7 hours ago
4th Dimension Example: Thread Runtime

- x-axis: time
- y-axis: thread runtime
- z-axis (color saturation): count at that time/runtime range
4th Dimension Example: Thread Runtime

- x-axis: time
- y-axis: thread runtime
- z-axis (color saturation): count at that time/runtime range
- omega-axis (color hue): application
  - blue == “coreaudiod”
4th Dimension Example: Thread Runtime

- x-axis: time
- y-axis: thread runtime
- z-axis (color saturation): count at that time/runtime range
- omega-axis (color hue): application
  - green == “iChat”
4th Dimension Example: Thread Runtime

- x-axis: time
- y-axis: thread runtime
- z-axis (color saturation): count at that time/runtime range
- omega-axis (color hue): application
- violet == “Chrome”
4th Dimension Example: Thread Runtime

• x-axis: time
• y-axis: thread runtime
• z-axis (color saturation): count at that time/runtime range
• omega-axis (color hue): application
• All colors
While the data supports the 4th dimension, visualizing this properly may become difficult (we are eager to find out)

- The image itself is still only 2 dimensional

May be best used to view a limited set, to limit the number of different hues; uses can include:

- Highlighting different cloud application types: DB, web server, etc.
- Highlighting one from many components: single node, CPU, disk, etc.

- Limiting the set also helps storage of data
More Visualizations

• We plan much more new stuff

  • We are building a team of engineers to work on it; including Bryan Cantrill, Dave Pacheo, and mysqlf

  • Dave and I have only been at Joyent for 2 1/2 weeks
Beyond Performance Analysis

• Visualizations such as heat maps could also be applied to:

• Security, with pattern matching for:
  • robot identification based on think-time latency analysis
  • inter-keystroke-latency analysis
  • brute force username latency attacks?

• System Administration
  • monitoring quota usage by user, filesystem, disk

• Other multi-dimensional datasets
Objectives

• Consider performance metrics before plotting
  • Why latency is good
  • ... and IOPS can be bad

• See the value of visualizations
  • Why heat maps are needed
  • ... and line graphs can be bad

• Remember key examples
  • I/O latency, as a heat map
  • CPU utilization by CPU, as a heat map
Heat Map: I/O Latency

- Latency matters
  - synchronous latency has a direct impact on performance
- Heat map shows
  - outliers, balance, cache layers, patterns
Heat Map: CPU Utilization

- Identify single threaded issues
  - single CPU hitting 100%
- Heat map shows
  - fully utilized components, balance, overall headroom, patterns
Tools Demonstrated

• For Reference:

• DTraceTazTool
  • 2006; based on TazTool by Richard McDougall 1995. Open source, unsupported, and probably no longer works (sorry).

• Analytics
  • 2008; Oracle Sun ZFS Storage Appliance

• “new stuff” (not named yet)
  • 2010; Joyent; Bryan Cantrill, Dave Pacheco, Brendan Gregg
• Thank you!

• How to find me on the web:
  • http://dtrace.org/blogs/brendan
  • http://blogs.sun.com/brendan <-- is my old blog
  • twitter @brendangregg