I work at the top of the performance support chain

I also write open source performance tools out of necessity to solve issues

- http://github.com/brendangregg
- http://www.brendangregg.com/#software

And books (DTrace, Solaris Performance and Tools)

Was Brendan @ Sun Microsystems, Oracle, now Joyent
Joyent

- Cloud computing provider
- Cloud computing software
- SmartOS
  - host OS, and guest via OS virtualization
- Linux, Windows
  - guest via KVM
Agenda

• Example Problem
• Performance Methodology
  • Problem Statement
  • The USE Method
  • Workload Characterization
  • Drill-Down Analysis
• Specific Tools

Saturday, July 28, 2012
Example Problem

• Recent cloud-based performance issue

• Customer problem statement:
  • “Database response time sometimes take multiple seconds. Is the network dropping packets?”
  • Tested network using traceroute, which showed some packet drops
Example: Support Path

- Performance Analysis

![Support Path Diagram]

Top

2nd Level

1st Level

Customer Issues
Example: Support Path

- Performance Analysis

Customer: “network drops?”

my turn
“network looks ok, CPU also ok”
“ran traceroute, can’t reproduce”
Example: Network Drops

• Old fashioned: network packet capture (sniffing)
  • Performance overhead during capture (CPU, storage) and post-processing (wireshark)
  • Time consuming to analyze: not real-time
Example: Network Drops

• New: dynamic tracing
  • Efficient: only drop/retransmit paths traced
  • Context: kernel state readable
  • Real-time: analysis and summaries

# ./tcplistendrop.d

<table>
<thead>
<tr>
<th>TIME</th>
<th>SRC-IP</th>
<th>PORT</th>
<th>DST-IP</th>
<th>PORT</th>
</tr>
</thead>
</table>

[...]
Example: Methodology

- Instead of network drop analysis, I began with the USE method to check system health
Example: Methodology

- Instead of network drop analysis, I began with the USE method to check system health

- In < 5 minutes, I found:
  - **CPU**: ok (light usage)
  - **network**: ok (light usage)
  - **memory**: available memory was exhausted, and the system was paging
  - **disk**: periodic bursts of 100% utilization

- The method is simple, fast, directs further analysis
Customer was surprised (are you sure?) I used latency analysis to confirm. Details (if interesting):

- **memory**: using both microstate accounting and dynamic tracing to confirm that anonymous pagins were hurting the database; worst case app thread spent 97% of time waiting on disk (data faults).

- **disk**: using dynamic tracing to confirm latency at the application / file system interface; included up to 1000ms fsync() calls.

- **Different methodology, smaller audience (expertise), more time (1 hour).**
**Example: Summary**

• **What happened:**
  
  • customer, 1st and 2nd level support spent much time chasing network packet drops.

• **What could have happened:**
  
  • customer or 1st level follows the USE method and quickly discover memory and disk issues
    
    • memory: fixable by customer reconfig
    
    • disk: could go back to 1st or 2nd level support for confirmation

• Faster resolution, frees time
Performance Methodology

• Not a tool
• Not a product
• Is a procedure (documentation)
Performance Methodology

• Not a tool -> but tools can be written to help
• Not a product -> could be in monitoring solutions
• Is a procedure (documentation)
Why Now: past

Performance analysis circa ‘90s, metric-orientated:

- Vendor creates metrics and performance tools
- Users develop methods to interpret metrics

Common method: “Tools Method”

- List available performance tools
- For each tool, list useful metrics
- For each metric, determine interpretation

Problematic: vendors often don’t provide the best metrics; can be blind to issue types
Why Now: changes

- Open Source
- Dynamic Tracing
  - See anything, not just what the vendor gave you
  - Only practical on open source software
  - Hardest part is knowing what questions to ask
Why Now: present

- Performance analysis now (post dynamic tracing), question-orientated:
  - Users pose questions
  - Check if vendor has provided metrics
  - Develop custom metrics using dynamic tracing

- Methodologies pose the questions
  - What would previously be an academic exercise is now practical
Methology Audience

• Beginners: provides a starting point
• Experts: provides a checklist/reminder
Performance Methodologies

• Suggested order of execution:
  1. Problem Statement
  2. The USE Method
  3. Workload Characterization
  4. Drill-Down Analysis (Latency)
Problem Statement

• Typical support procedure (1st Methodology):

1. What makes you think there is a problem?
2. Has this system ever performed well?
4. Can the performance degradation be expressed in terms of latency or run time?
5. Does the problem affect other people or applications?
6. What is the environment? What software and hardware is used? Versions? Configuration?
The USE Method

• Quick System Health Check (2nd Methodology):
  • For every resource, check:
    • Utilization
    • Saturation
    • Errors
The USE Method

- Quick System Health Check (2nd Methodology):
  - For every resource, check:
    - Utilization: time resource was busy, or degree used
    - Saturation: degree of queued extra work
    - Errors: any errors
The USE Method: Hardware Resources

- CPUs
- Main Memory
- Network Interfaces
- Storage Devices
- Controllers
- Interconnects
The USE Method: Hardware Resources

- A great way to determine resources is to find (or draw) the server *functional diagram*
  - The hardware team at vendors should have these
- Analyze every component in the data path
The USE Method: Functional Diagrams, Generic Example

- CPU 1
- CPU 2
- DRAM
- I/O Bridge
- I/O Controller
- Network Controller
- Disk
- Port

Memory Bus
CPU Interconnect
I/O Bus
Expander Interconnect
Interface Transports
The USE Method: Resource Types

- There are two different resource types, each define utilization differently:
  - **I/O Resource**: eg, network interface
    - utilization: time resource was busy. current IOPS / max or current throughput / max can be used in some cases
  - **Capacity Resource**: eg, main memory
    - utilization: space consumed
- Storage devices act as both resource types
The USE Method: Software Resources

- Mutex Locks
- Thread Pools
- Process/Thread Capacity
- File Descriptor Capacity
The USE Method: Flow Diagram

Choose Resource

Errors Present?

High Utilization?

Saturation?

Problem Identified
The USE Method: Interpretation

• Utilization

• 100% usually a bottleneck

• 70%+ often a bottleneck for I/O resources, especially when high priority work cannot easily interrupt lower priority work (e.g., disks)

• Beware of time intervals. 60% utilized over 5 minutes may mean 100% utilized for 3 minutes then idle

• Best examined per-device (unbalanced workloads)
The USE Method: Interpretation

- **Saturation**
  - Any non-zero value adds latency

- **Errors**
  - Should be obvious
## The USE Method: Easy Combinations

<table>
<thead>
<tr>
<th>Resource</th>
<th>Type</th>
<th>Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>utilization</td>
<td></td>
</tr>
<tr>
<td>CPU</td>
<td>saturation</td>
<td></td>
</tr>
<tr>
<td>Memory</td>
<td>utilization</td>
<td></td>
</tr>
<tr>
<td>Memory</td>
<td>saturation</td>
<td></td>
</tr>
<tr>
<td>Network Interface</td>
<td>utilization</td>
<td></td>
</tr>
<tr>
<td>Storage Device I/O</td>
<td>utilization</td>
<td></td>
</tr>
<tr>
<td>Storage Device I/O</td>
<td>saturation</td>
<td></td>
</tr>
<tr>
<td>Storage Device I/O</td>
<td>errors</td>
<td></td>
</tr>
</tbody>
</table>
## The USE Method: Easy Combinations

<table>
<thead>
<tr>
<th>Resource</th>
<th>Type</th>
<th>Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>utilization</td>
<td>CPU utilization</td>
</tr>
<tr>
<td>CPU</td>
<td>saturation</td>
<td>run-queue length</td>
</tr>
<tr>
<td>Memory</td>
<td>utilization</td>
<td>available memory</td>
</tr>
<tr>
<td>Memory</td>
<td>saturation</td>
<td>paging or swapping</td>
</tr>
<tr>
<td>Network Interface</td>
<td>utilization</td>
<td>RX/TX tput/bandwidth</td>
</tr>
<tr>
<td>Storage Device I/O</td>
<td>utilization</td>
<td>device busy percent</td>
</tr>
<tr>
<td>Storage Device I/O</td>
<td>saturation</td>
<td>wait queue length</td>
</tr>
<tr>
<td>Storage Device I/O</td>
<td>errors</td>
<td>device errors</td>
</tr>
</tbody>
</table>
## The USE Method: Harder Combinations

<table>
<thead>
<tr>
<th>Resource</th>
<th>Type</th>
<th>Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>errors</td>
<td></td>
</tr>
<tr>
<td>Network</td>
<td>saturation</td>
<td></td>
</tr>
<tr>
<td>Storage Controller</td>
<td>utilization</td>
<td></td>
</tr>
<tr>
<td>CPU Interconnect</td>
<td>utilization</td>
<td></td>
</tr>
<tr>
<td>Mem. Interconnect</td>
<td>saturation</td>
<td></td>
</tr>
<tr>
<td>I/O Interconnect</td>
<td>saturation</td>
<td></td>
</tr>
</tbody>
</table>
## The USE Method: Harder Combinations

<table>
<thead>
<tr>
<th>Resource</th>
<th>Type</th>
<th>Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>errors</td>
<td>eg, correctable CPU cache ECC events</td>
</tr>
<tr>
<td>Network</td>
<td>saturation</td>
<td>“nocanputs”, buffering</td>
</tr>
<tr>
<td>Storage Controller</td>
<td>utilization</td>
<td>active vs max controller IOPS and tput</td>
</tr>
<tr>
<td>CPU Interconnect</td>
<td>utilization</td>
<td>per port tput / max bandwidth</td>
</tr>
<tr>
<td>Mem. Interconnect</td>
<td>saturation</td>
<td>memory stall cycles</td>
</tr>
<tr>
<td>I/O Interconnect</td>
<td>saturation</td>
<td>bus throughput / max bandwidth</td>
</tr>
</tbody>
</table>
The USE Method: tools

- To be thorough, you will need to use:
  - CPU performance counters
    - For bus and interconnect activity; eg, perf events, cpustat
  - Dynamic Tracing
    - For missing saturation and error metrics; eg, DTrace
- Both can get tricky; tools can be developed to help
  - Please, no more top variants! ... unless it is *interconnect*-top or *bus*-top
  - I’ve written dozens of open source tools for both CPC and DTrace; much more can be done
Workload Characterization

• May use as a 3rd Methodology

• Characterize workload by:
  • **who** is causing the load? PID, UID, IP addr, ...
  • **why** is the load called? code path
  • **what** is the load? IOPS, tput, type
  • **how** is the load changing over time?

• Best performance wins are from *eliminating unnecessary work*

• Identifies class of issues that are load-based, not architecture-based
Drill-Down Analysis

- May use as a 4th Methodology
- Peel away software layers to drill down on the issue
- Eg, software stack I/O latency analysis:

<table>
<thead>
<tr>
<th>Layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
</tr>
<tr>
<td>System Call Interface</td>
</tr>
<tr>
<td>File System</td>
</tr>
<tr>
<td>Block Device Interface</td>
</tr>
<tr>
<td>Storage Device Drivers</td>
</tr>
<tr>
<td>Storage Devices</td>
</tr>
</tbody>
</table>
Drill-Down Analysis: Open Source

• With Dynamic Tracing, all function entry & return points can be traced, with nanosecond timestamps.

• One Strategy is to measure latency pairs, to search for the source; eg, A->B & C->D:

```c
static int
arc_cksum_equal(arc_buf_t *buf)
{
    zio_cksum_t zc;
    int equal;

    mutex_enter(&buf->b_hdr->b_freeze_lock);
    fletcher_2_native(buf->b_data, buf->b_hdr->b_size, &zc);
    equal = ZIO_CHECKSUM_EQUAL(*buf->b_hdr->b_freeze_cksum, zc);
    mutex_exit(&buf->b_hdr->b_freeze_lock);

    return (equal);
}
```
• **Method R**

  • A latency-based analysis approach for Oracle databases. See “Optimizing Oracle Performance" by Cary Millsap and Jeff Holt (2003)

• **Experimental approaches**

  • Can be very useful: eg, validating network throughput using iperf
Specific Tools for the USE Method
illuminos-based


<table>
<thead>
<tr>
<th>Resource</th>
<th>Type</th>
<th>Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>Utilization</td>
<td>per-cpu: mpstat 1, “idl”; system-wide: vmstat 1, “id”; per-process: prstat -c 1 (“CPU” == recent), prstat -mLc 1 (“USR” + “SYS”); per-kernel-thread: lockstat -Il rate, DTrace profile stack()</td>
</tr>
<tr>
<td>CPU</td>
<td>Saturation</td>
<td>system-wide: uptime, <strong>load averages</strong>; vmstat 1, “r”; DTrace dispqlen.d (DTT) for a better “vmstat r”; per-process: prstat -mLc 1, “LAT”</td>
</tr>
<tr>
<td>CPU</td>
<td>Errors</td>
<td>fmadm faulty; cpustat (CPC) for whatever error counters are supported (eg, thermal throttling)</td>
</tr>
<tr>
<td>Memory</td>
<td>Saturation</td>
<td>system-wide: vmstat 1, “sr” (bad now), “w” (was very bad); vmstat -p 1, “api” (anon page ins == pain), “apo”; per-process: prstat -mLc 1, “DFL”; DTrace anonpgpid.d (DTT), vminfo:::anonpgin on execname</td>
</tr>
</tbody>
</table>

• ... etc for all combinations (would span a dozen slides)
• http://dtrace.org/blogs/brendan/2012/03/07/the-use-method-linux-performance-checklist/

<table>
<thead>
<tr>
<th>Resource</th>
<th>Type</th>
<th>Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>Utilization</td>
<td>per-cpu: mpstat -P ALL 1, “%idle”; sar -P ALL, “%idle”; system-wide: vmstat 1, “id”; sar -u, “%idle”; dstat -c, “idl”; per-process: top, “%CPU”; htop, “CPU%”; ps -o pcpu; pidstat 1, “%CPU”; per-kernel-thread: top/htop (“K” to toggle), where VIRT == 0 (heuristic). [1]</td>
</tr>
<tr>
<td>CPU</td>
<td>Saturation</td>
<td>system-wide: vmstat 1, “r” &gt; CPU count [2]; sar -q, “runq-sz” &gt; CPU count; dstat -p, “run” &gt; CPU count; per-process: /proc/PID/schedstat 2nd field (sched_info.run_delay); perf sched latency (shows “Average” and “Maximum” delay per-schedule); dynamic tracing, eg, SystemTap schedtimes.stp “queued(us)” [3]</td>
</tr>
<tr>
<td>CPU</td>
<td>Errors</td>
<td>perf (LPE) if processor specific error events (CPC) are available; eg, AMD64’s “04Ah Single-bit ECC Errors Recorded by Scrubber” [4]</td>
</tr>
</tbody>
</table>

... etc for all combinations (would span a dozen slides)
• Earlier I said methodologies could be supported by monitoring solutions
• At Joyent we develop Cloud Analytics:
Future

• Methodologies for advanced performance issues
  • I recently worked a complex KVM bandwidth issue where no current methodologies really worked

• Innovative methods based on open source + dynamic tracing

• Less performance mystery. Less guesswork.

• Better use of resources (price/performance)

• Easier for beginners to get started
Thank you

• Resources:

  • http://dtrace.org/blogs/brendan
    • http://dtrace.org/blogs/brendan/2012/02/29/the-use-method/
    • http://dtrace.org/blogs/brendan/tag/usemethod/
    • http://dtrace.org/blogs/brendan/2011/12/18/visualizing-device-utilization/ - ideas if you are a monitoring solution developer

  • brendan@joyent.com