PERFORMANCE TUNING

*THIS SLIDE IS MOSTLY BASED ON “LINUX PERFORMANCE ANALYSIS AND TOOLS” PRESENTED AT SCALE11X AND “LINUX PERFORMANCE TOOLS” PRESENTED AT LINUXCON EUROPE 2014 BY BRENDAN GREGG

UNIX Programming 2014 Fall by Euiseong Seo
Need for Performance Tuning

- Reduce IT spend - find and eliminate waste, find areas to tune, and do more with less
- Build scalable architectures - understand system limits and develop around them
- Solve issues - locate bottlenecks and latency outliers
System Performance

- Find and fix kernel-based performance issues
  - 2-20% wins: I/O or buffer size tuning, NUMA config, etc
  - 2-200x wins: bugs, disabled features, perturbations causing latency outliers
- Kernels change, new devices are added, workloads scale, and new performance issues are encountered
- Analyze application performance from kernel/system context
  - 2-2000x wins: identifying and eliminating unnecessary work
- This is why we study OS and system software
Perspectives

System analysis can be top-down, or bottom-up:

- Workload
- Application
- System Libraries
- System Calls
- Kernel
- Devices

Developers

Operating System Software Stack

Sysadmins

Workload Analysis

Resource Analysis

Sunday, February 24, 13
Anti-Methodologies

- The lack of a deliberate methodology

- Street light anti-method
  1. Pick observability tools that are
     - Familiar
     - Found on the Internet, or at random
  2. Run tools
  3. Look for obvious issues

- Drunk man anti-method
  - Tune things at random until problem goes away
Methodologies

- For example, USE method
  - For every resource, check:
    - Utilization
    - Saturation
    - Errors

- 5 Whys
  - Ask “why” 5 times

- Other methods include
  - Workload characterization, drill-down analysis, event-tracing, baseline stats, static performance tuning

- Start with questions, then find tools
Command Line Tools

- Useful to study even if you never use them
  - GUIs and commercial products often use the same interfaces

```bash
$ vmstat 1
procs -----------memory---------- ---swap-- ...
 r  b  swpd free  buff  cache  si  so ...
 9  0   0 29549320  29252  9299060  0  ...
 2  0   0 29547876  29252  9299332  0  ...
 4  0   0 29548124  29252  9299460  0  ...
 5  0   0 29548840  29252  9299592  0  ...
```

Kernel

/proc, /sys, ...
Observability Tools

- Tools that watch diverse activities during execution of given workloads
- Useful for troubleshooting or performance optimization of specific workloads
How Do You Measure These?
Basic Tools

- `uptime`
- `top` (or `htop`)
- `ps`
- `vmstat`
- `iostat`
- `mpstat`
- `free`
uptime

- One way to print load averages

```
MacPro:~ euiseong$ uptime
19:30 up 5:26, 2 users, load averages: 1.05 0.90 0.75
MacPro:~ euiseong$
```

- A measure of resource demand
- Exponentially-damped moving averages
  - Historic trend without line graphs
- “Load > number of CPUs” may mean CPU saturation
top (or htop)

- System and per-process interval summary
- Can miss short-lived processes
- Can consume noticeable CPU to read /proc
## Process status listing

```bash
$ ps -ef f
```

<table>
<thead>
<tr>
<th>UID</th>
<th>PID</th>
<th>PPID</th>
<th>C</th>
<th>STIME</th>
<th>TTY</th>
<th>STAT</th>
<th>TIME</th>
<th>CMD</th>
</tr>
</thead>
<tbody>
<tr>
<td>[..]</td>
<td>[..]</td>
<td>[..]</td>
<td>[..]</td>
<td>[..]</td>
<td>[..]</td>
<td>[..]</td>
<td>[..]</td>
<td>[..]</td>
</tr>
<tr>
<td>root</td>
<td>4546</td>
<td>1</td>
<td>0</td>
<td>11:08</td>
<td>?</td>
<td>Ss</td>
<td>0:00</td>
<td>/usr/sbin/sshd -D</td>
</tr>
<tr>
<td>root</td>
<td>28261</td>
<td>4546</td>
<td>0</td>
<td>17:24</td>
<td>?</td>
<td>Ss</td>
<td>0:00</td>
<td>_ sshd: prod [priv]</td>
</tr>
<tr>
<td></td>
<td>28287</td>
<td>28261</td>
<td>0</td>
<td>17:24</td>
<td>?</td>
<td>S</td>
<td>0:00</td>
<td>_ sshd: prod@pts/0</td>
</tr>
<tr>
<td></td>
<td>28288</td>
<td>28287</td>
<td>0</td>
<td>17:24</td>
<td>pts/0</td>
<td>Ss</td>
<td>0:00</td>
<td>_ -bash</td>
</tr>
<tr>
<td></td>
<td>3156</td>
<td>28288</td>
<td>0</td>
<td>19:15</td>
<td>pts/0</td>
<td>R+</td>
<td>0:00</td>
<td>_ ps -ef f</td>
</tr>
<tr>
<td>root</td>
<td>4965</td>
<td>1</td>
<td>0</td>
<td>11:08</td>
<td>?</td>
<td>Ss</td>
<td>0:00</td>
<td>/bin/sh /usr/bin/svscanboot</td>
</tr>
<tr>
<td>root</td>
<td>4969</td>
<td>4965</td>
<td>0</td>
<td>11:08</td>
<td>?</td>
<td>S</td>
<td>0:00</td>
<td>_ svscan /etc/service</td>
</tr>
<tr>
<td>[..]</td>
<td>[..]</td>
<td>[..]</td>
<td>[..]</td>
<td>[..]</td>
<td>[..]</td>
<td>[..]</td>
<td>[..]</td>
<td>[..]</td>
</tr>
</tbody>
</table>


vmstat

- Virtual memory statistics and more

```
eulseong@accept:~$ vmstat -Sm 1
procs ----------memory-------- swap----- io----- system----- cpu-----
r  b   swpd free  buff  cache  si  so  bi  bo  in  cs  us  sy  id  wa  st
0  0   1823  256  5733  0  0   0   1  0  0  0  0  100  0  0
0  0   1824  256  5733  0  0   0   0  60  87  0  0  100  0  0
0  0   1824  256  5733  0  0   0   0  62  97  0  0  100  0  0
0  0   1824  256  5733  0  0   0   0  56  75  0  0  100  0  0
0  0   1824  256  5733  0  0   0   0  62  97  0  0  100  0  0
0  0   1824  256  5733  0  0   0   0  56  75  0  0  100  0  0
0  0   1824  256  5733  0  0   0   0  65  104  0  0  100  0  0
0  0   1824  256  5733  0  0   0   0  56  86  0  0  100  0  0
0  0   1824  256  5733  0  0   0   0  65  115  0  0  100  0  0
```

- First output line has some summary since boot values
**iostat**

- **Block I/O (disk I/O) stats**

```
euisong@accept:~$ iostat -xmdz 2
Linux 3.13.0-37-generic (accept)  12/09/2014   __x86_64__   (8 CPU)
Device: rrqm/s  wrqm/s  r/s  w/s  rMB/s  wMB/s  avgrq-sz  avgqu-sz   await  r-await  w-await  svctm  %util
sda   0.00    0.31    0.02    0.35    0.00     0.01    38.33     0.01    13.97     4.02   14.64    10.08    0.38
```
mpstat

- Multi-processor statistics
- Look for unbalanced workloads, hot CPUs
free

- **Main memory usage**

  - **buffers**: block device I/O cache
  - **cached**: virtual page cache
Basic Observability Tools

Operating System

Applications
- System Libraries

System Call Interface
- VFS
- Sockets
- File Systems
- TCP/UDP
- Volume Manager
- IP
- Block Device Interface
- Ethernet

Device Drivers

Linux Kernel

Hardware
- CPU
- Memory Bus
- DRAM

Tools:
- iostat
- uptime
- mpstat
- top ps
- vmstat
- free

I/O Bus

Expander Interconnect

I/O Bridge

I/O Controller

Disk

Swap

Network Controller

Port

Port

Interface Transports
Advanced Observability Tools

Linux & Observability & Tools

- strace
- lsof
- pcstat
- pidstat
- perf
- ftrace
- stap
- ktrace
- ebpf
- dtrace
- lttng

- Operating System
  - netstat
  - sysdig
  - perf
  - mpstat
  - CPU Interconnect

- System Call Interface
  - VFS
  - Sockets
  - TCP/UDP
  - IP

- Scheduler
  - Virtual Memory
    - system
    - iptraf
    - tcpdump

- Virtual Memory
  - Memory Bus
  - perf
  - tiptop

- Device Drivers
  - I/O Bridge
  - I/O Bus
  - Expander Interconnect

- Network Controller
  - Port
  - Port
  - Interface Transports
  - ethtool
  - snmpget
  - llidptool

- Disk
  - Swap
  - swapon

- Various:
  - sar
  - /proc
dstat
  - rdmsr

- Hardware
  - Various: CPU
  - CPU Interconnect
  - Memory Bus
  - perf
  - tiptop

- DRAM
  - nicstat
  - netstat
  - ip
Where to Observe

- **gprof**
  - Shows total amount of time your program spent executing each function
  - Also shows call graph of your program

- **Use gprof to first figure out where to attack**

- **Steps to use gprof**
  1. Compile and link your program with profiling enabled
  2. Execute your program to generate a profile data file
  3. Run gprof to analyze profile data
Using gprof

- Compiling
  - cc -g -c myprog.c -pg

- Execution of compile program will produce gmon.out

- Invoking gprof will produce analysis results

```
Compiling
```

```
Execution of compile program will produce gmon.out
```

```
Invoking gprof will produce analysis results
```