### Solaris Performance: Introduction

**Brendan Gregg**  
Sun Microsystems  
May 2007

```
# vmstat 1

kthr      memory            page
r b w   swap  free  re  mf pi po fr de

0 0 0 4596848 120908 0 3 0 0 0 0
0 0 0 4411920 48652 14 27 0 0 0 0
0 0 0 4411576 48316 80 476 0 0 0 0
0 0 0 4411576 48316 37 240 0 0 0 0
0 0 0 4411196 48004 45 467 0 0 0 0
0 0 0 4411196 48004 0 3 0 0 0 0
2 0 0 4410852 47728 23 236 0 0 0 0
1 0 0 4410852 47728 0 0 0 0 0 0
5 0 0 4410504 47448 23 235 0 0 0 0
3 0 0 4410208 47220 23 237 0 0 0 0
3 0 0 4410208 47220 0 0 0 0 0 0
3 0 0 4410208 47224 0 0 0 0 0 0
3 0 0 4410208 47224 0 3 0 0 0 0
4410648 47596 0 0 0 0 0 0
4410648 47596 0 0 0 0 0 0
4410648 47596 0 0 0 0 0 0
4411384 48204 0 9 0 0 0 0

Solaris Performance:
Introduction

**Brendan Gregg**  
Sun Microsystems  
May 2007
```
Solaris Performance: Introduction

• This presentation is an introduction to the field of Solaris performance.

• These slides cover:
  > Solaris Performance Features
    – Top Features
    – Solaris
    – Solaris 10
  > Solaris Performance Observability
    – By-Layer Strategy
    – 3-Metric Strategy
    – System Components
Performance Matters

• How performance helps you:
  1. Shipped performance features
     – Solaris can do more with less
  2. Tune performance features
     – Solaris tunables, library features, compiler optimisation, ...
  3. Manage resources
     – Get the best ROI
  4. Solve performance issues
     – Solaris has outstanding performance observability
Solaris Performance Features

• Solaris is a mature operating system with numerous performance features

• Top performance features are,
  > CPU and Memory Scaleability
  > 64-bit Support
  > Fully Preemptive Kernel
  > Resource Management
  > Compiler Technology
  > Observability
CPU and Memory Scaleability

- Sun bet on SMP in early 90's
  - Symmetric Multi Processing: user and kernel work distributed across all CPUs - best scaleability
- Per-CPU dispatcher queues
- Thread CPU affinity
- Processor sets and interrupt masking
- CMP and CMT support and optimisations
- Memory locality aware
- Kernel page relocation - for hot plug and DR
64-Bit Support

- Since Solaris 7 (October 1998)
- Originally for SPARC, now also AMD64 and IA-64

Fully Preemptive Kernel

- Allows Real Time scheduling class
Resource Management

- Standard tools: pbind, ulimit
- Processor sets, pools
- IPQoS - IP Quality of Service (network priorities)
- SRM - Solaris Resource Manager
- Zones + SRM = Containers
- FSS - Fair Share Schedular
- Resource Controls
  > CPU shares
  > Max threads, CPU time, file descriptors, ...
Compiler Technology

- Sun Studio compiler optimises for SPARC, x86
- Both gcc and cc can be used (try both and see)
- Java VM - hotspot compiler
Observability

• DTrace
• Microstate Accounting - prstat -mL
• kstat - vmstat, mpstat, ...
• procfs - ps, prstat, truss, ...
• PICs - cpustat/cputrack, busstat
Solaris Performance Feature List

- Scaleability
- Reliability
- Fully preemptive kernel
- Real-Time scheduling class
- Cyclic page cache
- Inode cache
- UFS buffer cache
- DNLC

- 64-bit support
- direct I/O
- cpustat/cputrack
- truss/apptrace
- libumem
- lgroups
- TCP MDT
- cyclics
- processor sets

- kstat
- procfs
- SNMP
- DISM
- NCA
- MPSS
- MPO
- rcapd
- SRM
Solaris 10 Performance Feature List

- DTrace
- ZFS
- Zones
- FireEngine - faster TCP/IP
- SMF - faster boot
- CMT, Niagara
- Numerous performance improvements (many found using DTrace)
Status

• Just Covered,
  > Solaris Performance Features
    – Top features
    – Solaris
    – Solaris 10

• Next up,
  > Solaris Performance Observability
    – By-Layer Strategy
    – 3-Metric Strategy
    – System Components
Solaris Performance Observability

- Solaris provides numerous performance tools; the trick is knowing what questions to ask - *performance analysis strategy*
By-Layer Strategy

- All software stack layers are observable
  > locate latency regardless of location
By-Layer Strategy

• For an application transaction, is the latency,
  > In the application code?
    – e.g., bad scalability architecture
  > In library code?
    – e.g., synchronisation locks
  > In syscalls?
    – e.g., disk or network I/O
  > In devices?
    – e.g., memory bus latency

• Solaris observability tools can provide the answers
  > especially DTrace
3-Metric Strategy

• For every system component, look for,
  1. Utilisation
  2. Saturation
  3. Errors
System Components

How do you measure utilisation, saturation and errors for these?

CPUs

Memory

Disks

Net

System Busses

* Your Architecture Will Vary

Simple diagram, simple question, this should be easy to answer.
System CPU

- Load average = overall utilisation + saturation

$ uptime$

2:30pm up 39 day(s), 12:40, 5 users, load average: 0.07, 0.07, 0.11

> printed by uptime, prstat
> 1, 5 and 15 minute averages.
> Divide load average by CPU count,
  - value < 1.0 suggests idle, and value = utilisation
  - value == 1.0 suggests 100% utilisation
  - value > 1.0 suggests saturation
> Useful for an initial impression, then move onto other tools like vmstat and mpstat
System CPU

- `vmstat` - utilisation and saturation as metrics

```
$ vmstat 1

kthr          memory          page          disk          faults          cpu
  r    b    w        swap   free   re    mf    pi    po    fr   de    sr   cd   s0   --   --   --   in   sy   cs   us   sy   id
0    0    0   4592308   120572   0    3    0    0    0    5   30   -1   0    0   967  5343  861  2    1    97
2    0    0   4349740   48280    10    28   0    0    0    0    0    0    0    0   602  1253  791  55   0    45
0    0    0   4349756   48320    0    0    0    0    0    0    0    0    0    0   608 1059  723  50   1    49
[...]```

- first line is summary since boot
- kthr:r = saturation, total threads on the run queues (but sampled at a low rate)
- cpu:us + cpu:sy = utilisation, CPU user and system time
System CPU

• **mpstat** - *utilisation* by-CPU

```bash
$ mpstat 1
CPU minf mjf xcal intr ithr csw icsw migr smtx srw syscl usr sys wt idl
 0 2 0 108 607 338 434 33 18 22 0 2580 2 1 0 96
 1 2 0 80 360 61 427 24 18 22 0 2762 2 1 0 97
CPU minf mjf xcal intr ithr csw icsw migr smtx srw syscl usr sys wt idl
 0 0 0 8 451 323 203 74 24 5 0 261 85 1 0 14
 1 6 0 5 137 1 503 44 25 0 0 727 14 0 0 86
CPU minf mjf xcal intr ithr csw icsw migr smtx srw syscl usr sys wt idl
 0 0 0 6 620 328 279 51 34 9 0 238 84 0 0 16
 1 0 0 175 143 1 450 62 19 5 0 685 17 1 0 82
[...]```

• **Classic performance problem** - under utilised CPUs due to poor threading architecture
System CPU

- Solaris 10 FMA detects and can automatically respond to CPU errors
- `fmadm faulty` - what faults currently exist
- `fmstat -m cpumem-retire` - raw statistics

```
$ fmstat -m cpumem-retire

<table>
<thead>
<tr>
<th>NAME</th>
<th>VALUE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>auto_flts</td>
<td>0</td>
<td>auto-close faults received</td>
</tr>
<tr>
<td>bad_flts</td>
<td>0</td>
<td>invalid fault events received</td>
</tr>
<tr>
<td>cpu_blfails</td>
<td>0</td>
<td>failed cpu blacklists</td>
</tr>
<tr>
<td>cpu_blsupp</td>
<td>0</td>
<td>cpu blacklists suppressed</td>
</tr>
<tr>
<td>cpu_fails</td>
<td>0</td>
<td>cpu faults unresolveable</td>
</tr>
<tr>
<td>cpu_flts</td>
<td>0</td>
<td>cpu faults resolved</td>
</tr>
<tr>
<td>cpu_supp</td>
<td>0</td>
<td>cpu offlines suppressed</td>
</tr>
<tr>
<td>nop_flts</td>
<td>0</td>
<td>inapplicable fault events received</td>
</tr>
</tbody>
</table>

[...]
System Memory

- **vmstat** - swap and physical memory **utilisation** and **saturation**

```
$ vmstat 1

kthr   memory   page   disk   faults   cpu
r b w  swap  free  re  mf  pi  po  fr  de  sr  cd  s0  --  --  in  sy  cs  us  sy  id
 0 0 0  4592236  120548  0  3  0  0  0  0  5  30  -1  0  0  967  5342  861  2  1  97
 0 0 0  4350572  48096  18  30  0  0  0  0  0  0  0  0  0  687  1114  781  0  1  99
 0 0 0  4350572  48124  0  0  0  0  0  0  0  0  0  0  0  6206  37271  11979  3  12  85

[...]
```

- > **swap** - free virtual memory (RAM + disk based swap)
- > **free** - available physical memory (RAM)
- > **page:sr** - values suggest physical memory saturation

- **mdb -k** - provides breakdown with ::memstat
System Memory

- Solaris 10 FMA detects and can automatically respond to memory **errors**
- For example, blacklisting a page of RAM that has had too many (correctable) ECC errors
- `fmadm faulty` - what is currently faulted
- `fmstat -m cpumem-retire` - raw statistics
System Disks

- **iostat** - disk utilisation, saturation, errors

```
$ iostat -xnmpz 5

extended device statistics

<table>
<thead>
<tr>
<th>device</th>
<th>r/s</th>
<th>w/s</th>
<th>kr/s</th>
<th>kw/s</th>
<th>wait</th>
<th>actv</th>
<th>wsvc_t</th>
<th>asvc_t</th>
<th>%w</th>
<th>%b</th>
</tr>
</thead>
<tbody>
<tr>
<td>c0t0d0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>1.1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c1t0d0s0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>11.3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c1t0d0s1</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.1</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>8.8</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c1t0d0s3</td>
<td>0.7</td>
<td>1.9</td>
<td>7.3</td>
<td>21.8</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>15.7</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>c1t0d0s4</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>13.6</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
```

> first output is summary since boot
> %b - percent busy, a measure of utilisation
> wait - transactions waiting, a measure of saturation

- **iostat** `-E` - error summaries
# System Network

- **kstat** - network utilisation, saturation, errors

```bash
$ kstat -n nge0 10

module: nge           instance: 0
name: nge0            class:    net
  brdcstrcv        0
  brdcstxmt        0
  collisions       0
  ctime            61.227502261
  ierrors          0
  ifspeed          100000000
  ipackets         145866056

[...]
```

> output includes byte counts, various errors

- **netstat** and **nicstat** (opensource) provide useful summaries of network stats
System Busses

- Measuring **utilisation, saturation and errors** is hard, but usually still possible with some effort
  - `cpustat` - measure CPU Performance Instrumentation Counters (PICs)
    - PICs for cache activity, memory bus activity, instruction events
  - `cputrack` - CPU PICs for a process
  - `busstat` - On some SPARC systems, provides hardware bus PICs
Processes

• Apart from performance observability by-system, also examine performance observability by-process.

• prstat -mL - useful microstates by thread

```
$ prstat -mL

<table>
<thead>
<tr>
<th>PID</th>
<th>USERNAME</th>
<th>USR</th>
<th>SYR</th>
<th>TRP</th>
<th>TFL</th>
<th>DFL</th>
<th>LCK</th>
<th>SLP</th>
<th>LAT</th>
<th>VCX</th>
<th>ICX</th>
<th>SCL</th>
<th>SIG</th>
<th>PROCESS/LWPID</th>
</tr>
</thead>
<tbody>
<tr>
<td>557</td>
<td>brendan</td>
<td>7.9</td>
<td>0.3</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>91</td>
<td>0.5</td>
<td>579</td>
<td>141</td>
<td>2K</td>
<td>96</td>
<td>Xorg/1</td>
</tr>
<tr>
<td>828</td>
<td>brendan</td>
<td>0.6</td>
<td>0.4</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>95</td>
<td>4.1</td>
<td>434</td>
<td>299</td>
<td>2K</td>
<td>0</td>
<td>ssh/1</td>
</tr>
<tr>
<td>830</td>
<td>brendan</td>
<td>0.2</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>99</td>
<td>0.3</td>
<td>36</td>
<td>11</td>
<td>160</td>
<td>0</td>
<td>gnome-termin/1</td>
</tr>
<tr>
<td>788</td>
<td>brendan</td>
<td>0.1</td>
<td>0.1</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>100</td>
<td>0.0</td>
<td>58</td>
<td>0</td>
<td>910</td>
<td>0</td>
<td>dtwm/1</td>
</tr>
<tr>
<td>1437</td>
<td>brendan</td>
<td>0.0</td>
<td>0.1</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>100</td>
<td>0.0</td>
<td>44</td>
<td>2</td>
<td>297</td>
<td>0</td>
<td>prstat/1</td>
</tr>
<tr>
<td>791</td>
<td>brendan</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>100</td>
<td>0.3</td>
<td>7</td>
<td>11</td>
<td>129</td>
<td>0</td>
<td>dtterm/1</td>
</tr>
</tbody>
</table>
[...]```

• DTrace - measure custom microstates
  > in terms of application activity, across all software layers
Further Observability

• Much more can be observed and analysed on Solaris
  > DTrace is its own field of study
• “You don't miss what you never had”
  > Once you start exploring Solaris observability, other OSes won't feel the same again
References

- [http://www.solarisinternals.com](http://www.solarisinternals.com)
  > Latest Solaris Performance Slides
  > Performance wiki
- The “Solaris Performance and Tools” book,
  [http://www.sun.com/books/catalog/solaris_perf_tools.xml](http://www.sun.com/books/catalog/solaris_perf_tools.xml)
- Performance Community,
Ctrl-D

Brendan Gregg
brendan@sun.com