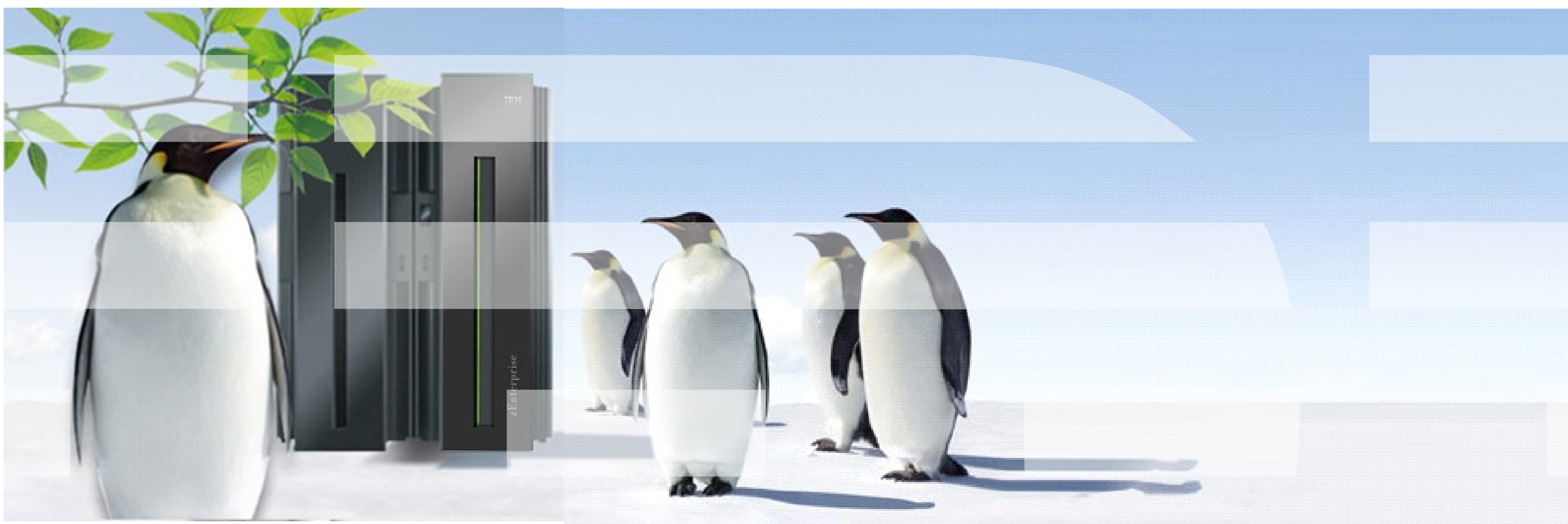
# How to Surprise by being a Linux Performance “know-it-all”

Martin Schwidefsky
IBM Lab Böblingen, Germany
February 7 2013



# Trademarks & Disclaimer

The following are trademarks of the International Business Machines Corporation in the United States and/or other countries. For a complete list of IBM Trademarks, see www.ibm.com/legal/copytrade.shtml:

IBM, the IBM logo, BladeCenter, Calibrated Vectored Cooling, ClusterProven, Cool Blue, POWER, PowerExecutive, Predictive Failure Analysis, ServerProven, System p, System Storage, System x , System z, WebSphere, DB2 and Tivoli are trademarks of IBM Corporation in the United States and/or other countries. For a list of additional IBM trademarks, please see http://ibm.com/legal/copytrade.shtml.

The following are trademarks or registered trademarks of other companies: Java and all Java based trademarks and logos are trademarks of Sun Microsystems, Inc., in the United States and other countries or both Microsoft, Windows,Windows NT and the Windows logo are registered trademarks of Microsoft Corporation in the United States, other countries, or both. Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.
UNIX is a registered trademark of The Open Group in the United States and other countries or both. Linux is a trademark of Linus Torvalds in the United States, other countries, or both. Cell Broadband Engine is a trademark of Sony Computer Entertainment Inc. InfiniBand is a trademark of the InfiniBand Trade Association.
Other company, product, or service names may be trademarks or service marks of others.

NOTES: Linux penguin image courtesy of Larry Ewing (lewing@isc.tamu.edu) and The GIMP

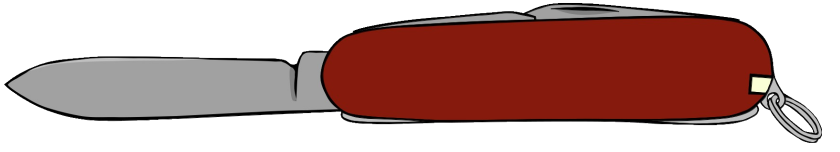
Any performance data contained in this document was determined in a controlled environment. Actual results may vary significantly and are dependent on many factors including system hardware configuration and software design and configuration. Some measurements quoted in this document may have been made on development-level systems. There is no guarantee these measurements will be the same on generally-available systems. Users of this document should verify the applicable data for their specific environment. IBM hardware products are manufactured from new parts, or new and serviceable used parts. Regardless, our warranty terms apply.
Information is provided “AS IS” without warranty of any kind. All customer examples cited or described in this presentation are presented as illustrations of the manner in which some customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics will vary depending on individual customer configurations and conditions.

This publication was produced in the United States. IBM may not offer the products, services or features discussed in this document in other countries, and the information may be subject to change without notice. Consult your local IBM business contact for information on the product or services available in your area.
All statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.
Information about non-IBM products is obtained from the manufacturers of those products or their published announcements. IBM has not tested those products and cannot confirm the performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Prices are suggested US list prices and are subject to change without notice. Starting price may not include a hard drive, operating system or other features. Contact your IBM representative or Business Partner for the most current pricing in your geography. Any proposed use of claims in this presentation outside of the United States must be reviewed by local IBM country counsel prior to such use. The information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any

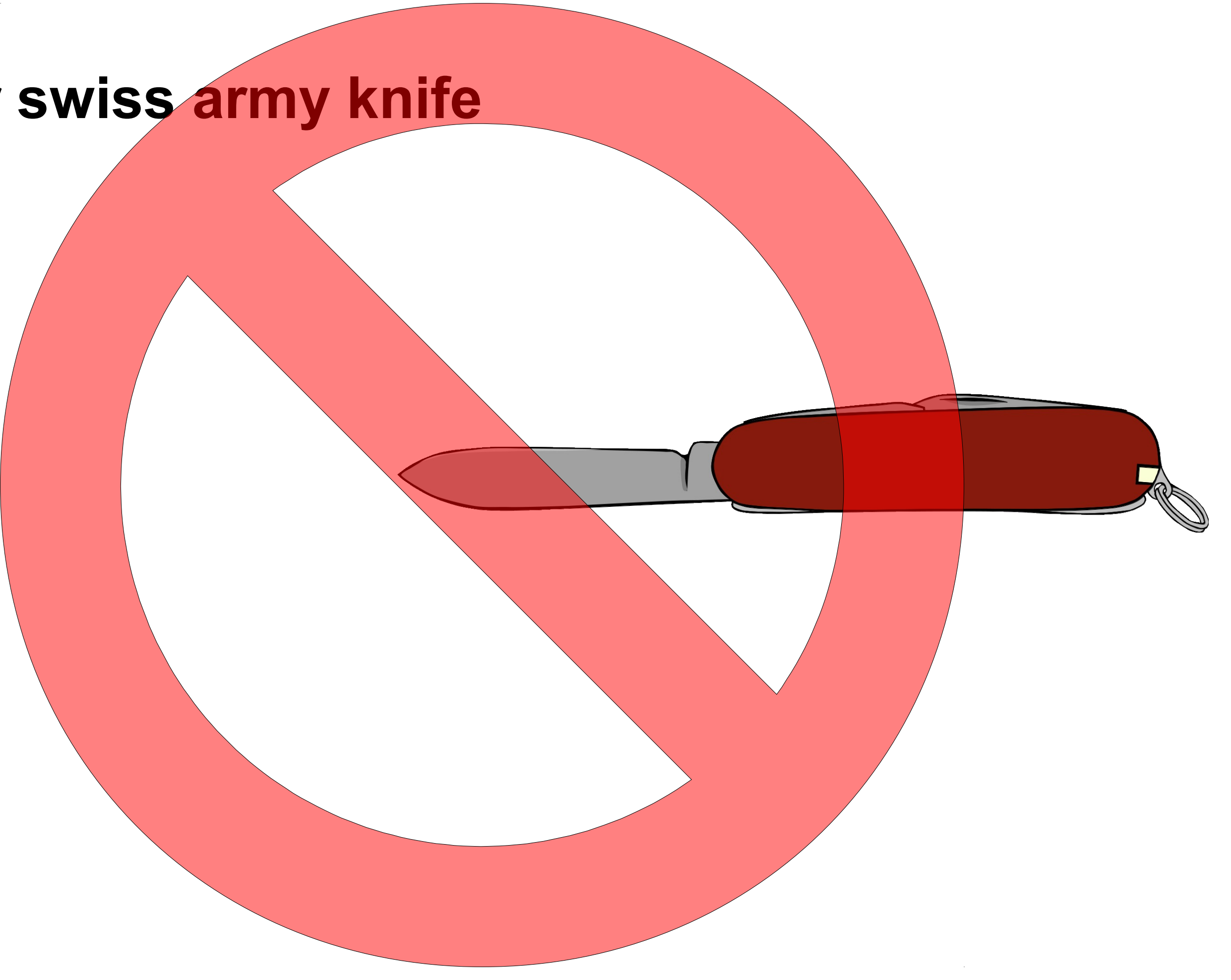
# Agenda (what it used to be)

- **Tools are your swiss army knife**
  - ps
  - top
  - sadc/sar
  - iostat
  - vmstat
  - netstat



# Agenda (what it used to be)

- **Tools are your swiss army knife**
  - ps
  - top
  - sadc/sar
  - iostat
  - vmstat
  - netstat



# Ready for take-off

# Agenda

- **Tools are your swiss army knife**
  - htop
  - dstat
  - pidstat
  - irqstats
  - strace/ltrace
  - blktrace
  - hyptop
  - profiling
  - valgrind
  - iptraf
  - tracepoints

# General thoughts on performance tools

- **Things that are always to consider**
  - Monitoring can impact the system
  - Most data gathering averages over a certain period of time
    → this flattens peaks
  - Start with defining the problem
    - which parameter(s) from the application/system indicates the problem
    - which range is considered as bad, what is considered as good
  - monitor the good case and save the results
    - comparisons when a problem occurs can save days and weeks
- **Staged approach saves a lot of work**
  - Try to use general tools to isolate the area of the issue
  - Create theories and try to quickly verify/falsify them
  - Use advanced tools to debug the identified area

# Orientation - where to go

| Tool | 1st overview | CPU consumption | latencies | Hot spots | Disk I/O | Memory | Network |
|---|---|---|---|---|---|---|---|
| top / ps | x | x | | | | | |
| sysstat | x | x | | | x | x | |
| vmstat | x | x | | | | x | |
| iostat | x | | | | x | | |
| dasdstat | | | | | x | | |
| scsistat | | | | | x | | |
| netstat | x | | | | | | x |
| htop / dstat / pidstat | x | x | x | | x | | |
| irqstats | x | x | x | | | | |
| strace / ltrace | | | x | | | | |
| hyptop | | x | | | | | |
| profiling | | x | | x | | | |
| blktrace | | | | | x | | |
| valgrind | | | | | | x | |
| iptraf | x | | | | | | x |
| tracepoints | | | x | x | x | x | x |

# DSTAT

- Characteristics: Live easy to use full system information
- Objective: Flexible set of statistics
- Usage:

```
dstat -tv -aio -disk-util -n -net-packets -i -ipc -D total,
      [diskname] -top-io […] [interval]
dstat -tinv
```

- Shows
  - Throughput and utilization
  - Summarized and per Device queue information
  - Much more, it more or less combines several classic tools like iostat and vmstat
- Hints
  - Powerful plugin concept
    - "`--top-io`" for example identifies the application causing the most I/Os
  - Colorization allows fast identification of deviations

# DSTAT – the limit is your screen width

# HTOP

- Characteristics: Process overview with extra features
- Objective: Get a understanding about your running processes
- Usage:

```
htop
```

- Shows
  - Running processes
  - CPU and memory utilization
  - Accumulated times
  - I/O rates
  - System utilization visualization
- Hints
  - Htop can display more uncommon fields (in menu)
  - Able to send signals out of its UI for administration purposes
  - Processes can be sorted/filtered for a more condensed view

# HTOP

Configurable utilization visualization

```
0  [||||||||||||||||||||||||||||||||||||||||||||13.1%]     Tasks: 101, 80 thr; 60 running
1  [||||||||||||||||||||||||||||||||||||||||||||14.5%]     Load average: 42.03 16.67 6.24
2  [|||||||||||||||||||||||||||||||||||||||||||||1.9%]     Uptime: 00:17:11
3  [||||||||||||||||||||||||||||||||||||||||||||13.6%]
4  [|||||||||||||||||||||||||||||||||||||||||||||0.0%]
5  [||||||||||||||||||||||||||||||||||||||||||||14.1%]
Mem[||||||||||||||||||||||||||||||||||||||||945/12059MB]
Swp[                                              0/0MB]

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM% - -  UTIME+   STIME+   IORR  IOWR   TIME+  Command
51931 postgres   20   0 3264M  142M  140M S  1.0  1.2 - -  0:00.47  0:00.21   627     0  0:00.68 postgres:
51962 postgres   20   0 3264M  157M  154M R  3.0  1.3 - -  0:00.56  0:00.24   483     0  0:00.80 postgres:
51981 postgres   20   0 3264M  170M  168M R  3.0  1.4 - -  0:00.61  0:00.26   424     0  0:00.87 postgres:
51921 postgres   20   0 3264M  164M  162M R  1.0  1.4 - -  0:00.57  0:00.25   398     0  0:00.83 postgres:
51953 postgres   20   0 3264M  169M  166M R  1.0  1.4 - -  0:00.62  0:00.27   280     0  0:00.89 postgres:
51934 postgres   20   0 3264M  174M  172M R  2.0  1.4 - -  0:00.64  0:00.27   269     0  0:00.91 postgres:
51923 postgres   20   0 3264M  156M  153M R  3.0  1.3 - -  0:00.55  0:00.26   269     0  0:00.81 postgres:
51933 postgres   20   0 3264M  154M  151M S  1.0  1.3 - -  0:00.55  0:00.26   251     0  0:00.81 postgres:
51942 postgres   20   0 3264M  178M  175M R  1.0  1.5 - -  0:00.68  0:00.31   205     0  0:00.99 postgres:
51946 postgres   20   0 3264M  139M  136M R  1.0  1.2 - -  0:00.47  0:00.22   200     0  0:00.69 postgres:
51979 postgres   20   0 3264M  128M  126M S  1.0  1.1 - -  0:00.38  0:00.21   187     0  0:00.59 postgres:
```

Common process info

Accumulated Usage and IO rates

Hierarchy

# PIDSTAT

- Characteristics: Easy to use extended per process statistics
- Objective: Identify processes with peak activity
- Usage:

```
pidstat [-w | -r | -d]
```

- Shows
  - `-w` context switching activity and if it was voluntary
  - `-r` memory statistics, especially minor/major faults per process
  - `-d` disk throughput per process
- Hints
  - Also useful if run as background log due to its low overhead
    - Good extension to sadc in systems running different applications/services
  - `-p <pid>` can be useful to track activity of a specific process

# PIDSTAT examples

```
12:46:18 PM       PID   cswch/s nvcswch/s  Command
12:46:18 PM         3      2.39      0.00  ksoftirqd/0
12:46:18 PM         4      0.04      0.00  migration/0
12:46:18 PM      1073    123.42    180.18  Xorg
```

Voluntarily / Involuntary

```
12:47:51 PM       PID  minflt/s  majflt/s     VSZ    RSS   %MEM  Command
12:47:51 PM       985      0.06      0.00   15328   3948   0.10  smbd
12:47:51 PM       992      0.04      0.00    5592   2152   0.05  sshd
12:47:51 PM      1073    526.41      0.00 1044240 321512   7.89  Xorg
```

Faults per process

```
12:49:18 PM       PID   kB_rd/s   kB_wr/s kB_ccwr/s  Command
12:49:18 PM       330      0.00      1.15      0.00  kjournald
12:49:18 PM      2899      4.35      0.09      0.04  notes2
12:49:18 PM      3045     23.43      0.01      0.00  audacious2
```

How much KB disk I/O per process

# IRQ Statistics

- Characteristics: Low overhead IRQ information
- Objective: Condensed overview of IRQ activity
- Usage:

```
cat / proc/interrupts
```

- Shows
  - Which interrupts happen on which cpu
- Hints
  - Recent Versions (SLES11-SP2) much more useful
  - If interrupts are unintentionally unbalanced
  - If the amount of interrupts matches I/O
    - This can point to non-working IRQ avoidance

# IRQ Statistics

- Example
  - Network focus on CPU zero (in this case unwanted)
  - Scheduler covered most of that avoiding idle CPU 1-3
  - But caused a lot migrations, IPI's and cache misses

```
            CPU0       CPU1       CPU2       CPU3
EXT:       21179      24235      22217      22959
I/O:     1542959     340076     356381     325691
CLK:       15995      16718      15806      16531   [EXT] Clock Comparator
EXC:         255        325        332        227   [EXT] External Call
EMS:        4923       7129       6068       6201   [EXT] Emergency Signal
TMR:           0          0          0          0   [EXT] CPU Timer
TAL:           0          0          0          0   [EXT] Timing Alert
PFL:           0          0          0          0   [EXT] Pseudo Page Fault
DSD:           0          0          0          0   [EXT] DASD Diag
VRT:           0          0          0          0   [EXT] Virtio
SCP:           6         63         11          0   [EXT] Service Call
IUC:           0          0          0          0   [EXT] IUCV
CPM:           0          0          0          0   [EXT] CPU Measurement
CIO:         163        310        269        213   [I/O] Common I/O Layer Interrupt
QAI:     1541773     338857     354728     324110   [I/O] QDIO Adapter Interrupt
DAS:        1023        909       1384       1368   [I/O] DASD
[…] 3215, 3270, Tape, Unit Record Devices, LCS, CLAW, CTC, AP Bus, Machine Check
```

# STRACE

- Characteristics: High overhead, high detail tool
- Objective: Get insights about the ongoing system calls of a program
- Usage:

```
strace -p [pid of target program]
```

- Shows
  - Identify kernel entries called more often or taking too long
    - Can be useful if you search for increased system time
  - Time in call (`-T`)
  - Relative time-stamp (`-r`)
- Hints
  - The option "`-c`" allows medium overhead by just tracking counters and durations

# STRACE - example

a lot or slow calls?

shares to rate importance

name (see man pages)

```
strace -cf -p 26802
Process 26802 attached - interrupt to quit
^CProcess 26802 detached
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
 58.43    0.007430          17       450           read
 24.33    0.003094           4       850       210 access
  5.53    0.000703           4       190        10 open
  4.16    0.000529           3       175           write
  2.97    0.000377           2       180           munmap
  1.95    0.000248           1       180           close
  1.01    0.000128           1       180           mmap
  0.69    0.000088          18         5           fdatasync
  0.61    0.000078           0       180           fstat
  0.13    0.000017           3         5           pause
------ ----------- ----------- --------- --------- ----------------
100.00    0.012715                  2415       225 total
```

# LTRACE

- Characteristics: High overhead, high detail tool
- Objective: Get insights about the ongoing library calls of a program
- Usage:

```
ltrace -p [pid of target program]
```

- Shows
  - Identify library calls that are too often or take too long
    - Good if you search for additional user time
    - Good if things changed after upgrading libs
  - Time in call (`-T`)
  - Relative time-stamp (`-r`)
- Hints
  - The option "`-c`" allows medium overhead by just tracking counters and durations
  - The option -S allows to combine ltrace and strace

# LTRACE - example

a lot or slow calls?

shares to rate importance

name (see man pages)

```
ltrace -cf -p 26802
% time     seconds  usecs/call     calls      function
------ ----------- ----------- --------- --------------------
 98.33   46.765660     5845707         8 pause
  0.94    0.445621          10     42669 strncmp
  0.44    0.209839          25      8253 fgets
  0.08    0.037737          11      3168 __isoc99_sscanf
  0.07    0.031786          20      1530 access
  0.04    0.016757          10      1611 strchr
  0.03    0.016479          10      1530 snprintf
  0.02    0.010467        1163         9 fdatasync
  0.02    0.008899          27       324 fclose
  0.02    0.007218          21       342 fopen
  0.01    0.006239          19       315 write
  0.00    0.000565          10        54 strncpy
------ ----------- ----------- --------- --------------------
100.00   47.560161                 59948 total
```

# STRACE / LTRACE – full trace

- Without -c both tools produce a full detail log
  - Via -f child processes can be traced as well
  - Extra options "`-Tr`" are useful to search for latencies follow time in call / relative time-stamp
  - Useful to "read" what exactly goes on when

```
Example strace'ing a sadc data gatherer
0.000028 write(3, "\0\0\0\0\0\0\0\17\0\0\0\0\0\0\0\17\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"..., 680) = 680 <0.000007>
0.000027 write(3, "\0\0\0\0\0\0\0\17\0\0\0\0\0\0\0\17\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"..., 680) = 680 <0.000007>
0.000026 fdatasync(3)              = 0 <0.002673>
0.002688 pause()                   = 0 <3.972935>
3.972957 --- SIGALRM (Alarm clock) @ 0 (0) ---
0.000051 rt_sigaction(SIGALRM, {0x8000314c, [ALRM], SA_RESTART}, {0x8000314c, [ALRM], SA_RESTART}, 8) = 0 <0.000005>
0.000038 alarm(4)                  = 0 <0.000005>
0.000031 sigreturn()               = ? (mask now []) <0.000005>
0.000024 stat("/etc/localtime", {st_mode=S_IFREG|0644, st_size=2309, ...}) = 0 <0.000007>
0.000034 open("/proc/uptime", O_RDONLY) = 4 <0.000009>
0.000024 fstat(4, {st_mode=S_IFREG|0444, st_size=0, ...}) = 0 <0.000005>
0.000029 mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x3fffd20a000 <0.000006>
0.000028 read(4, "11687.70 24836.04\n", 1024) = 18 <0.000010>
0.000027 close(4)                  = 0 <0.000006>
0.000020 munmap(0x3fffd20a000, 4096) = 0 <0.000009>
```

# BLKTRACE

- Characteristics: High detail info of the block device layer actions
- Objective: Understand whats going with your I/O in the kernel and devices
- Usage:

```
blktrace -d [device(s)]
blkparse -st [commontracefilepart]
```

- Shows
  - Events like merging, request creation, I/O submission, I/O completion, ...
  - Timestamps and disk offsets for each event
  - Associated task and executing CPU
  - Application and CPU summaries
- Hints
  - Filter masks allow lower overhead if only specific events are of interest
  - Has an integrated client/server mode to stream data away
    - Avoids extra disk I/O on a system with disk I/O issues

# BLKTRACE – when is it useful

- Often its easy to identify that I/O is slow, but
  → Where?
  → Because of what?

- Block trace allows to
  - Analyze Disk I/O characteristics like sizes and offsets
    - Maybe your I/O is split in a layer below
  - Analyze the timing with details about all involved Linux layers
    - Often useful to decide if HW or SW causes stalls
  - Summaries per CPU / application can identify imbalances

# BLKTRACE - events

Common

| | | |
|---|---|---|
| A -- | remap | For stacked devices, incoming i/o is remapped to device below it in the i/o stack. The remap action details what exactly is being remapped to what. |
| Q -- | queued | This notes intent to queue i/o at the given location. No real requests exists yet. |
| G -- | get request | To send any type of request to a block device, a struct request container must be allocated first. |
| I -- | inserted | A request is being sent to the i/o scheduler for addition to the internal queue and later service by the driver. The request is fully formed at this time. |
| D -- | issued | A request that previously resided on the block layer queue or in the i/o scheduler has been sent to the driver. |
| C -- | complete | A previously issued request has been completed. The output will detail the sector and size of that request, as well as the success or failure of it. |

Plugging & Merges:

| | | |
|---|---|---|
| P -- | plug | When i/o is queued to a previously empty block device queue, Linux will plug the queue in anticipation of future I/Os being added before this data is needed. |
| U -- | unplug | Some request data already queued in the device, start sending requests to the driver. This may happen automatically if a timeout period has passed (see next entry) or if a number of requests have been added to the queue.Recent kernels associate the queue with the submitting task and unplug also on a context switch. |
| T -- | unplug | due to timer If nobody requests the i/o that was queued after plugging the queue, Linux will automatically unplug it after a defined period has passed. |
| M -- | back merge | A previously inserted request exists that ends on the boundary of where this i/o begins, so the i/o scheduler can merge them together. |
| F -- | front merge | Same as the back merge, except this i/o ends where a previously inserted requests starts. |

Special

| | | |
|---|---|---|
| B -- | bounced | The data pages attached to this bio are not reachable by the hardware and must be bounced to a lower memory location. This causes a big slowdown in i/o performance, since the data must be copied to/from kernel buffers. Usually this can be fixed with using better hardware -- either a better i/o controller, or a platform with an IOMMU. |
| S -- | sleep | No available request structures were available, so the issuer has to wait for one to be freed. |
| X -- | split | On raid or device mapper setups, an incoming i/o may straddle a device or internal zone and needs to be chopped up into smaller pieces for service. This may indicate a performance problem due to a bad setup of that raid/dm device, but may also just be part of normal boundary conditions. dm is notably bad at this and will clone lots of i/o. |

# BLKTRACE - events

Common

| | | |
|---|---|---|
| A -- | remap | For stacked devices, incoming i/o is remapped to device below it in the i/o stack. The remap action details what exactly is being remapped to what. |
| Q -- | queued | This notes intent to queue i/o at the given location. No real requests exists yet. |
| G -- | get request | To send any type of request to a block device, a struct request container must be allocated first. |
| I -- | inserted | A request is being sent to the i/o scheduler for addition to the internal queue and later service by the driver. The request is fully formed at this time. |
| D -- | issued | A request that previously resided on the block layer queue or in the i/o scheduler has been sent to the driver. |
| C -- | complete | A previously issued request has been completed. The output will detail the sector and size of that request, as well as the success or failure of it. |

Plugging & Merges:

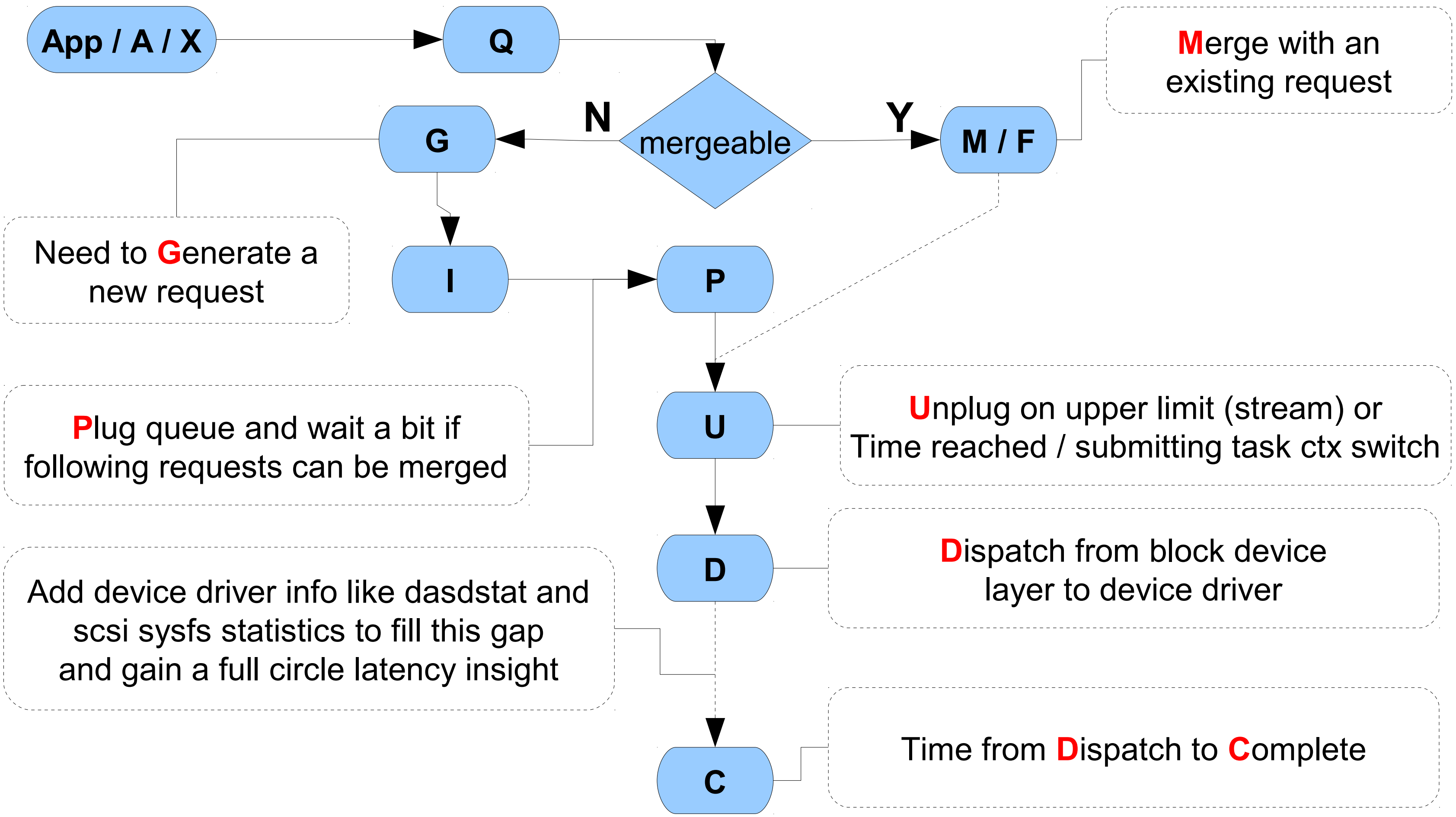
| | | |
|---|---|---|
| P -- | plug | When i/o is queued to a previously empty block device queue, Linux will plug the queue in anticipation of future I/Os being added before this data is r [illegible] |
| U -- | unplug | Some request data alread [illegible] driver. This may happen automatically if a timeout period has passed [illegible] en added to the queue.Recent kernels associate the queue with t [illegible] ch. |
| T -- | unplug | due to timer If nobody requ [illegible] ue, Linux will automatically unplug it after a defined period has passed. |
| M -- | back merge | A previously inserted request exists that ends on the boundary of where this i/o begins, so the i/o scheduler can merge them together. |
| F -- | front merge | Same as the back merge, except this i/o ends where a previously inserted requests starts. |

**Good as documentation, but hard to understand/remember**

Special

| | | |
|---|---|---|
| B -- | bounced | The data pages attached to this bio are not reachable by the hardware and must be bounced to a lower memory location. This causes a big slowdown in i/o performance, since the data must be copied to/from kernel buffers. Usually this can be fixed with using better hardware -- either a better i/o controller, or a platform with an IOMMU. |
| S -- | sleep | No available request structures were available, so the issuer has to wait for one to be freed. |
| X -- | split | On raid or device mapper setups, an incoming i/o may straddle a device or internal zone and needs to be chopped up into smaller pieces for service. This may indicate a performance problem due to a bad setup of that raid/dm device, but may also just be part of normal boundary conditions. dm is notably bad at this and will clone lots of i/o. |

# Block device layer – events (simplified)



# BLKTRACE - example

- Example Case
  - The snippet shows a lot of 4k requests (8x512 byte sectors)
    - We expected the I/O to be 32k
  - Each one is dispatched separately (no merges)
    - This caused unnecessary overhead and slow I/O

```
Maj/Min CPU   Seq-nr        sec.nsec   pid Action RWBS sect + size  map source / task
94,4      27       21     0.059363692 18994      A     R 20472832 + 8 <- (94,5) 20472640
94,4      27       22     0.059364630 18994      Q     R 20472832 + 8 [qemu-kvm]
94,4      27       23     0.059365286 18994      G     R 20472832 + 8 [qemu-kvm]
94,4      27       24     0.059365598 18994      I     R 20472832 + 8 (     312) [qemu-kvm]
94,4      27       25     0.059366255 18994      D     R 20472832 + 8 (     657) [qemu-kvm]
94,4      27       26     0.059370223 18994      A     R 20472840 + 8 <- (94,5) 20472648
94,4      27       27     0.059370442 18994      Q     R 20472840 + 8 [qemu-kvm]
94,4      27       28     0.059370880 18994      G     R 20472840 + 8 [qemu-kvm]
94,4      27       29     0.059371067 18994      I     R 20472840 + 8 (     187) [qemu-kvm]
94,4      27       30     0.059371473 18994      D     R 20472840 + 8 (     406) [qemu-kvm]
```

# BLKTRACE - example

- Example Case
  - Analysis turned out that the I/O was from the swap code
    - Same offsets were written by kswapd
  - A recent code change there disabled the ability to merge
  - The summary below shows the difference after a fix

```
Total initially
 Reads Queued:        560,888,     2,243MiB  Writes Queued:       226,242,   904,968KiB
 Read Dispatches:     544,701,     2,243MiB  Write Dispatches:    159,318,   904,968KiB
 Reads Requeued:            0                Writes Requeued:           0
 Reads Completed:     544,716,     2,243MiB  Writes Completed:    159,321,   904,980KiB
 Read Merges:          16,187,    64,748KiB  Write Merges:         61,744,   246,976KiB
 IO unplugs:          149,614                Timer unplugs:         2,940

Total after Fix
 Reads Queued:        734,315,     2,937MiB  Writes Queued:       300,188,     1,200MiB
 Read Dispatches:     214,972,     2,937MiB  Write Dispatches:    215,176,     1,200MiB
 Reads Requeued:            0                Writes Requeued:           0
 Reads Completed:     214,971,     2,937MiB  Writes Completed:    215,177,     1,200MiB
 Read Merges:         519,343,     2,077MiB  Write Merges:         73,325,   293,300KiB
 IO unplugs:          337,130                Timer unplugs:        11,184
```

# HYPTOP

- Characteristics: Easy to use Guest/LPAR overview
- Objective: Check CPU and overhead statistics of your own and sibling images
- Usage:

```
hyptop
```

- Shows
  - CPU load & Management overhead
  - Memory usage
  - Can show image overview or single image details
- Hints
  - Good “first view” tool for linux admins that want to look “out of their linux”
  - Requirements:
    - For z/VM the Guest needs Class B
    - For LPAR “Global performance data control” check-box in HMC

# HYPTOP

Why are exactly 4 CPUs used in all 6 CPU guests

Weights are equal

All these do not fully utilize their 2 CPUs

service guest weights

No peaks in service guests

LPAR images would see other LPARs

11:12:56 CPU-T: UN(64)

| system (str) | #cpu (#) | cpu (%) | Cpu+ (hm) | online (dhm) | memuse (GiB) | memmax (GiB) | wcur (#) |
|---|---|---|---|---|---|---|---|
| R3729003 | 6 | 399.11 | 2:24 | 0:03:05 | 11.94 | 12.00 | 100 |
| R3729004 | 6 | 399.07 | 2:24 | 0:03:05 | 11.94 | 12.00 | 100 |
| R3729001 | 6 | 398.99 | 2:26 | 0:03:09 | 11.95 | 12.00 | 100 |
| R3729005 | 6 | 398.76 | 2:24 | 0:03:05 | 11.94 | 12.00 | 100 |
| R3729009 | 4 | 398.62 | 2:22 | 0:03:05 | 4.20 | 6.00 | 100 |
| R3729008 | 4 | 398.49 | 2:22 | 0:03:05 | 4.21 | 6.00 | 100 |
| R3729007 | 4 | 398.39 | 2:21 | 0:03:05 | 4.18 | 6.00 | 100 |
| R3729010 | 4 | 398.02 | 2:21 | 0:03:05 | 4.18 | 6.00 | 100 |
| R3729002 | 6 | 397.99 | 2:24 | 0:03:05 | 11.94 | 12.00 | 100 |
| R3729006 | 4 | 393.09 | 2:21 | 0:03:05 | 4.17 | 6.00 | 100 |
| R3729012 | 2 | 117.37 | 0:43 | 0:03:05 | 0.25 | 2.00 | 100 |
| R3729014 | 2 | 117.27 | 0:44 | 0:03:05 | 0.25 | 2.00 | 100 |
| R3729011 | 2 | 117.13 | 0:43 | 0:02:37 | 0.25 | 2.00 | 100 |
| R3729013 | 2 | 117.08 | 0:43 | 0:03:05 | 0.25 | 2.00 | 100 |
| R3729015 | 2 | 116.63 | 0:43 | 0:03:05 | 0.25 | 2.00 | 100 |
| VMSERVU | 1 | 0.00 | 0:00 | 0:03:10 | 0.01 | 0.03 | 1500 |
| VMSERVP | 1 | 0.00 | 0:00 | 0:03:10 | 0.01 | 0.06 | 1500 |
| VMSERVR | 1 | 0.00 | 0:00 | 0:03:10 | 0.01 | 0.03 | 1500 |
| RACFVM | 1 | 0.00 | 0:00 | 0:03:10 | 0.01 | 0.02 | 100 |
| OPERSYMP | 1 | 0.00 | 0:00 | 0:03:10 | 0.00 | 0.03 | 100 |
| TCPIP | 1 | 0.00 | 0:00 | 0:03:10 | 0.01 | 0.12 | 3000 |
| DTCVSW2 | 1 | 0.00 | 0:00 | 0:03:10 | 0.01 | 0.03 | 100 |

# Profiling

- Characteristics: Easy to use profiling and kernel tracing
- Objective: Get detailed information where & why CPU is consumed
- Usage:

```
perf top
```

- Shows
  - Sampling for CPU hotspots
    - Annotated source code along hotspot
  - CPU event counters
  - Further integrated non-sampling tools
- Hints
  - Without HW support only userspace can be reasonably profiled
  - “successor” of oprofile that is available with HW support (SLES11-SP2)
  - Perf HW support code upstream, wait for next distribution releases

# Profiling

- What profiling can and what it can't
  - + Search hot-spots of CPU consumption worth to optimize
  - + List functions according to their usage
  - - Search where time is lost (I/O, Stalls)

- Perf is not just a sampling tool
  - Integrated tools to evaluate tracepoints like "perf sched", "perf timechart", …
  - Opposite to real sampling this can help to search for stalls

# Profiling

- Perf example howto
  - We had a case where new code caused cpus to scale badly
  - `perf record "workload"`
    - Creates a file called perf.data that can be analyzes
  - We used "`perf diff`" on both data files to get a comparison

- "Myriad" of further options/modules
  - Live view with perf top
  - Perf sched for an integrated analysis of scheduler tracepoints
  - Perf annotate to see samples alongside code
  - Perf stat for a counter based analysis
  - [...]

# Profiling

- Perf example (perf diff)
  - found a locking issue causing increased cpu consumption

```
# Baseline  Delta                          Symbol
# ........ ..........  ................... ......
#
    12.14%     +8.07%  [kernel.kallsyms]   [k] lock_acquire
     8.96%     +5.50%  [kernel.kallsyms]   [k] lock_release
     4.83%     +0.38%  reaim               [.] add_long
     4.22%     +0.41%  reaim               [.] add_int
     4.10%     +2.49%  [kernel.kallsyms]   [k] lock_acquired
     3.17%     +0.38%  libc-2.11.3.so      [.] msort_with_tmp
     3.56%     -0.37%  reaim               [.] string_rtns_1
     3.04%     -0.38%  libc-2.11.3.so      [.] strncat
```

# Valgrind

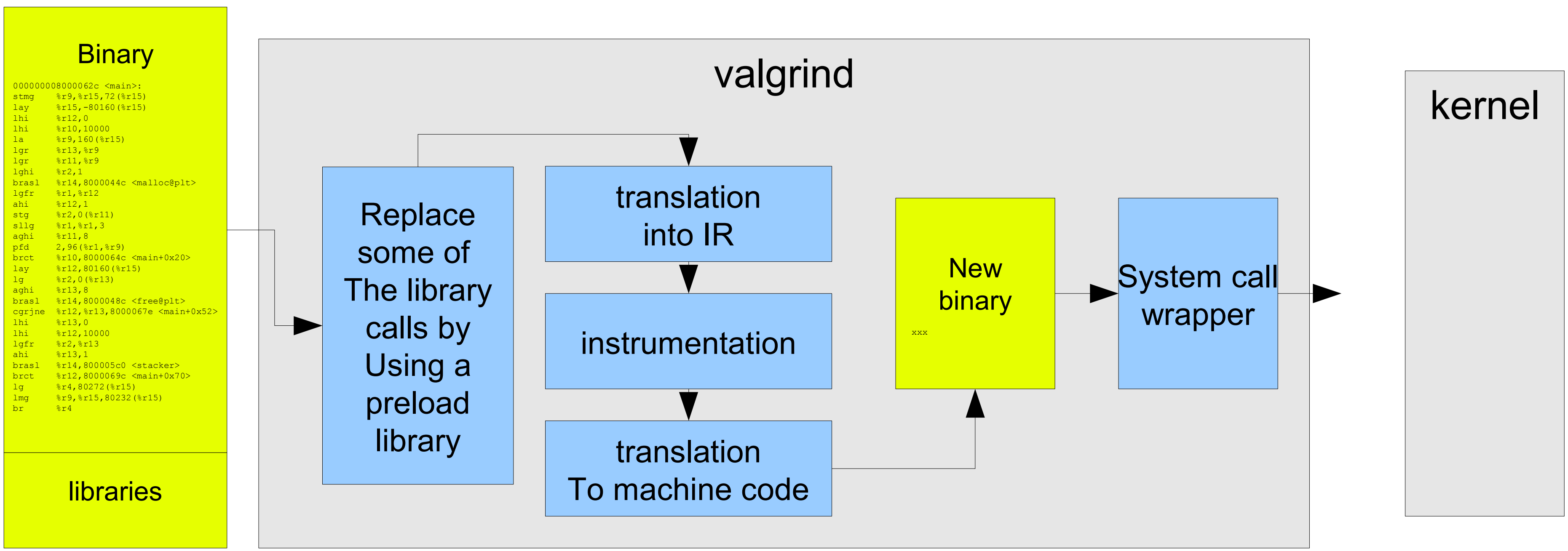
- Characteristics: In depth memory analysis
- Objective: Find out where memory is leaked, sub-optimally cached, ...
- Usage:

```
valgrind [program]
```

- Shows
  - Memory leaks
  - Cache profiling
  - Heap profiling
- Hints
  - Runs on binaries, therefore easy to use
  - Debug Info not required but makes output more useful

# Valgrind Overview

- Technology is based on a JIT (Just-in-Time Compiler)
- Intermediate language allows debugging instrumentation



# Valgrind – sample output of “memcheck”

```
# valgrind buggy_program
==2799== Memcheck, a memory error detector
==2799== Copyright (C) 2002-2010, and GNU GPL'd, by Julian Seward et al.
==2799== Using Valgrind-3.6.1 and LibVEX; rerun with -h for copyright info
==2799== Command: buggy_program
==2799==
==2799== HEAP SUMMARY:
==2799==     in use at exit: 200 bytes in 2 blocks
==2799==   total heap usage: 2 allocs, 0 frees, 200 bytes allocated
==2799==
==2799== LEAK SUMMARY:
==2799==    definitely lost: 100 bytes in 1 blocks
==2799==    indirectly lost: 0 bytes in 0 blocks
==2799==      possibly lost: 0 bytes in 0 blocks
==2799==    still reachable: 100 bytes in 1 blocks
==2799==         suppressed: 0 bytes in 0 blocks
==2799== Rerun with --leak-check=full to see details of leaked memory
[...]
```

- Important parameters:
  - --leak-check=full
  - --track-origins=yes

# Valgrind - Tools

- Several tools
  - Memcheck (default): detects memory and data flow problems
  - Cachegrind: cache profiling
  - Massif: heap profiling
  - Helgrind: thread debugging
  - DRD: thread debugging
  - None: no debugging (for valgrind JIT testing)
  - Callgrind: codeflow and profiling
- Tool can be selected with –tool=xxx
- System z support since version 3.7 (SLES-11-SP2)
- Backports into 3.6 (SLES-10-SP4, RHEL6-U1)

# IPTRAF

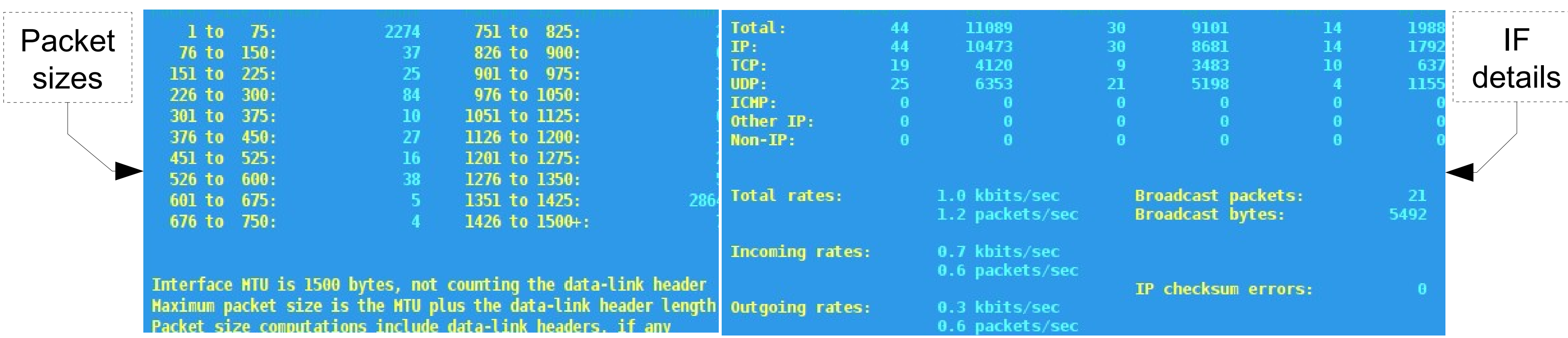
- Characteristics: Live information on network devices / connections
- Objective: Filter and format network statistics
- Usage:

```
iptraf
```

- Shows
  - Details per Connection / Interface
  - Statistical breakdown of ports / packet sizes
  - LAN station monitor
- Hints
  - Can be used for background logging as well
    - Use SIGUSR1 and logrotate to handle the growing amount of data
  - Knowledge of packet sizes important for the right tuning

# IPTRAF

- Questions that usually can be addressed
  - Connection behavior overview
  - Do you have peaks in your workload characteristic
  - With whom does your host really communicate
- Comparison to wireshark
  - Not as powerful, but much easier and faster to use
  - Lower overhead and no sniffing needed (often prohibited)



# Tracepoints (Events)

- Characteristics: Complex interface, but a vast source of information
- Objective: In kernel latency and activity insights
- Usage: Access debugfs mount point /tracing
- Shows
  - Time-stamp and activity name
  - Tracepoints can provide event specific context data
  - Infrastructure adds extra common context data like cpu, preempts depth, ...
- Hints
  - Very powerful and customizable, there are hundreds of tracepoints
    - Some tracepoints have tools to be accessed “perf sched”, “blktrace” both base on them
    - Others need custom post processing
  - There are much more things you can handle with tracepoints check out Kernel Documentation/trace/tracepoint-analysis.txt (via perf stat) and Kernel Documentation/trace/events.txt (custom access)

# Tracepoints – example I/III

- Here we use custom access since there was tool
  - We searched for 1.2ms extra latency
    - Target is it lost in HW, User-space, Kernel or all of them
  - Workload was a simple 1 connection 1 byte←→1 byte load
  - Call "`perf list`" for a list of currently supported tracepoints
  - We used the following tracepoints

| Abbreviation | Tracepoint | Meaning |
|---|---|---|
| R | netif_receive_skb | low level receive |
| P | napi_poll | napi work related to receive |
| Q | net_dev_queue | enqueue in the stack |
| S | net_dev_xmit | low level send |

# Tracepoints – example II/III

- Simplified script (full versions might tune buffer sizes, check files, and so on)

```
echo latency-format > /sys/kernel/debug/tracing/trace_options           # enable tracing type
echo net:* >> /sys/kernel/debug/tracing/set_event                       # select specific events
echo napi:* >> /sys/kernel/debug/tracing/set_event                      # “
echo "name == ${dev}" > /sys/kernel/debug/tracing/events/net/filter     # set filters
echo "dev_name == ${dev}" > /sys/kernel/debug/tracing/events/napi/filter # “
cat /sys/kernel/debug/tracing/trace >> ${output}                        # synchronous
echo !*:* > /sys/kernel/debug/tracing/set_event                         # disable tracing
```

- Output

```
#                 _------=> CPU#
#                / _-----=> irqs-off
#               | / _----=> need-resched
#               || / _---=> hardirq/softirq
#               ||| / _--=> preempt-depth
#               |||| /     delay
#  cmd     pid  ||||| time   |   caller
#     \   /     |||||  \     |   /
   <...>-24116 0..s. 486183281us+: net_dev_xmit: dev=eth5 skbaddr=0000000075b7e3e8 len=67 rc=0
  <idle>-0     0..s. 486183303us+: netif_receive_skb: dev=eth5 skbaddr=000000007ecc6e00 len=53
  <idle>-0     0.Ns. 486183306us+: napi_poll: napi poll on napi struct 000000007d2479a8 fordevice eth
   <...>-24116 0..s. 486183311us+: net_dev_queue: dev=eth5 skbaddr=0000000075b7e3e8 len=67
   <...>-24116 0..s. 486183317us+: net_dev_xmit: dev=eth5 skbaddr=0000000075b7e3e8 len=67 rc=0
```

# Tracepoints – example III/III

- Example postprocessed

| | SUM | COUNT | AVERAGE | MIN | MAX | STD-DEV |
|---|---|---|---|---|---|---|
| P2Q: | 8478724 | 1572635 | 5.39 | 4 | 2140 | 7.41 |
| Q2S: | 12188675 | 1572638 | 7.65 | 3 | 71 | 4.89 |
| S2R: | 38562294 | 1572636 | 24.42 | 1 | 2158 | 9.08 |
| R2P: | 4197486 | 1572633 | 2.57 | 1 | 43 | 2.39 |
| SUM: | 63427179 | 1572635 | 40.03 | | | |

| | SUM | COUNT | AVERAGE | MIN | MAX | STD-DEV |
|---|---|---|---|---|---|---|
| P2Q: | 7191885 | 1300897 | 5.53 | 4 | 171 | 1.31 |
| Q2S: | 10622270 | 1300897 | 8.17 | 3 | 71 | 5.99 |
| S2R: | 32078550 | 1300898 | 24.66 | 2 | 286 | 5.88 |
| R2P: | 3707814 | 1300897 | 2.85 | 1 | 265 | 2.59 |
| SUM: | 53600519 | 1300897 | 41.20 | | | |

- Confirmed that most of the 1.2 ms were lost in our image
- Confirmed that it was not at/between a specific function
  - Eventually it was an interrupt locality issue causing bad caching

# Orientation – where to go

| Tool | 1st overview | CPU consumption | latencies | Hot spots | Disk I/O | Memory | Network |
|---|---|---|---|---|---|---|---|
| top / ps | x | x | | | | | |
| sysstat | x | x | | | x | x | |
| vmstat | x | x | | | | x | |
| iostat | x | | | | x | | |
| dasdstat | | | | | x | | |
| scsistat | | | | | x | | |
| netstat | x | | | | | | x |
| htop / dstat / pidstat | x | x | x | | x | | |
| irqstats | x | x | x | | | | |
| strace / ltrace | | | x | | | | |
| hyptop | | x | | | | | |
| profiling | | x | | x | | | |
| blktrace | | | | | x | | |
| valgrind | | | | | | x | |
| iptraf | x | | | | | | x |
| tracepoints | | | x | x | x | x | x |

# Questions ?

- Further information is available at
  - Live Virtual Classes for z/VM and Linux
    http://www.vm.ibm.com/education/lvc/
  - Linux on System z – Tuning hints and tips
    http://www.ibm.com/developerworks/linux/linux390/perf/index.html

IBM

***Martin Schwidefsky***

*Linux on System z*
*Development*

*Schönaicher Strasse 220*
*71032 Böblingen, Germany*

*Phone +49 (0)7031-16-2247*
*schwidefsky@de.ibm.com*

# Please Evaluate!