How to surprise by being a Linux-performance “Know-it-all”

Martin Schwidefsky
IBM Lab Böblingen, Germany
12th August 2015
Sessions 17672 & 17673
Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at www.ibm.com/legal/copytrade.shtml.
Agenda

- Your swiss army knife for the complex cases
  - Pidstat – per process statistics
  - Slabtop – kernel memory pool consumption
  - Lsof – check file flags of open files
  - Blktrace – low level disk I/O analysis
  - Hyptop – cross guest cpu consumption monitor
  - Iptraf – network traffic monitor
  - Dstat – very configurable live system overview
  - Irqstats – check irq amount and cpu distribution
  - Smem – per process/per mapping memory overview
  - Jinsight – Java method call stack analysis
  - Htop – top on steroids
  - Strace – system call statistics
  - Ltrace – library call statistics
  - Kernel tracepoints – get in-depth timing inside the kernel
  - Vmstat – virtual memory statistics
  - Sysstat – full system overview
  - Iostat – I/O related statistics
  - Dasdstat – disk statistics
  - scsi statistics – disk statistics
  - Perf – hw counters, tracepoint based evaluations, profiling to find hotspots
  - Valgrind – in depth memory/cache analysis and leak detection
  - Java Health Center – high level java overview and monitoring
  - Java Garbage Collection and Memory visualizer – in depth gc analysis
  - Netstat – network statistics and overview
  - Socket Statistics – extended socket statistics
  - top / ps – process overview
  - Icactats / Ikszcrypt – check usage of crypto hw support
  - Lsluns / multipath – check multipath setup
  - Lsqeth – check hw checksummung and buffer count
  - Ethtool – check offloading functions
  - Collectl – full system monitoring
  - Ftrace – kernel function tracing
  - Lttng – complex latency tracing infrastructure
  - Ziomon – Analyze FCP setup and I/O
  - Systemtap – another kernel tracing infrastructure
  - Wireshark / Tcpdump – analyze network traffic in depth
  - Iotop – order processes by disk I/O
  - Iftop - per connection traffic overview
  … ever growing
## Agenda – approximately 5 x 60 minutes

<table>
<thead>
<tr>
<th><strong>Basic</strong></th>
<th><strong>Intermediate</strong></th>
<th><strong>Advanced</strong></th>
<th><strong>Master</strong></th>
<th><strong>Elite</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>– Utilization</td>
<td>– General thoughts</td>
<td>– Strace</td>
<td>– Perf</td>
<td>– Cachestat</td>
</tr>
<tr>
<td>– Scheduling</td>
<td>– Sysstat</td>
<td>– Ltrace</td>
<td>– slabtop</td>
<td>– Smem</td>
</tr>
<tr>
<td>– Page Cache</td>
<td>– Dasdstat</td>
<td>– Lsof</td>
<td>– Blktrace</td>
<td>– Valgrind</td>
</tr>
<tr>
<td></td>
<td>– iotop</td>
<td>– Multipath</td>
<td>– Tcpendump</td>
<td>– Wireshark</td>
</tr>
<tr>
<td>– ps</td>
<td>–icastats</td>
<td>– Dstat</td>
<td>– Java Garbage Collection and Memory visualizer</td>
<td>Tracepoints</td>
</tr>
<tr>
<td>– vmstat</td>
<td>– Lsqeth</td>
<td>– Htop</td>
<td>– Jinsight</td>
<td>– Systemtap</td>
</tr>
<tr>
<td></td>
<td>– Ethtool</td>
<td>– Netstat</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Agenda

**Today we make up for the so far missing introduction(s)**

## Basic
- Utilization
- Scheduling
- Page Cache
- Swapping
- top
- ps
- vmstat

## Intermediate
- General thoughts
- Sysstat
- Dasdstat
- Scsi I/O statistics
- iotop
- Lszerpt
- icastats
- Lsqeth
- Ethtool
- Preparation

## Advanced
- Strace
- Ltrace
- Lsof
- Lsluns
- Multipath
- hyptop
- Dstat
- Htop
- Netstat
- Socket Statistics
- Iptraf

## Master
- Perf
- slabtop
- Blktrace
- Ziomon
- Tcpdump
- Java Health Center
- Java Garbage Collection and Memory visualizer
- Jinsight

## Elite
- Cachestat
- Smem
- Valgrind
- Irqstats
- Wireshark
- Kernel Tracepoints
- Systemtap
Utilization

- Utilization means that a cpu core is used
- Categories qualify what the core was used for
  - System, IRQ, SoftIRQ
  - Userspace, Guest
  - Idle, IOWait, Steal
  - Nice
- Accounting unit is Jiffy, reports usually as percentage
  - Percentage is better to express relative ratios
- The majority of those basics is usually known, but it still has a purpose
  - Clarify and synchronize the understanding between everybody
  - Provide metaphors that allow to explain it more easily next time
Utilization - Metaphor

- For all the clarifications on basic terms I will use metaphors
  - Based on well known real world examples
  - At the beginning of a topic the matching metaphor is provided

- Imagine a laptop is a CPU core

- People using that laptop are different Programs, that could be
  - Application(s)
  - Kernel
  - Hypervisor

*Roles are defined by clothes and equipment not people (bad actors)
Utilization - USR

- If a userspace application is running it is accounted as **USR**
  - This is usually what you want
  - Also known as problem state, because there your problems are solved
  - If this “application” is actually a virtualized guest it is accounted to Guest instead
Utilization – SYS, (H)IRQ, SIRQ

- For some tasks you need certain privileges
  - so you call an administrator (System Call to the Kernel)
  - He executes the privileged stuff for you (accounted as SYStem time)
Utilization – SYS, (H)IRQ, SIRQ

- For some tasks you need certain privileges
  – so you call an administrator (System Call to the Kernel)
  – He executes the privileged stuff for you (accounted as SYSstem time)
  – There are subcategories
    - (H)IRQ: privileged work driven by interrupts instead by the user
    - SIRQ: privileged work driven by soft interrupts and tasklets
Utilization - Idle

- This is the most simple case of “doing nothing”
- The CPU executes nothing because it is not requested to do so
- This is accounted as idle
Utilization - IOWait

- Again “doing nothing”, but no more that simple
- The System was requested to do some synchronous I/O
  - Still the CPU executes nothing because it is not requested to do so
  - But it knows it “could” do some work if that I/O would complete
- This is accounted as **iowait**
Utilization – IOWait

- If no one is waiting for I/O, (asynchronous)
  - Example 1: real Linux AIO
  - Example 2: writes via page cache
- Not accounted as `iowait`, but `idle`
Utilization - Steal

- The CPU is doing something, just not for you
- CPU doesn't exist for you, but you'd need a CPU to realize that
  - Accounted as **steal** time
  - Based on Virtual vs Real timers
- Imagine the laptop is used for multiple groups of people and switched between their docking stations
  - A group of people in front of one laptop cause Context switches (later)

Application thought it could work (scheduled)

CPU seems to be non existent
Utilization - Steal

- In case a CPU shouldn't run anyway the stealing isn't even recognized
- Still accounted as **idle** or **iowait**

If the Application doesn't work

Nobody cares about the CPU being “non existant”
Utilization - Steal

- A Linux does not know for which purpose the cpu was stolen
- Steal can indicate issues
  - Too high cpu or memory overcommitment
- But steal also isn't always bad
  - It could be work you requested, just like “USR->SYS in Linux alone”

Thought it would transmit network packets
It got from the App

CPU is still busy but for other things than Linux thought

Asked the kernel for I/O handling
Utilization – Steal Quiz

- I: Driving I/O synchronously e.g. causing a vswitch to work for your submission
  => Steal? Yes

- II: Driving sync reads from a file, mdisk does work for you
  => Steal? No, IOWait

- III: Driving async writes (AIO) by a DB, causing mdisk work in the HV
  => Steal? Kind of, depends on if the CPU is idle or not

- IV: Paging in the HV takes place while you were running a Java based BI load utilizing all cores
  => Steal? Yes

- V: Paging in the HV takes place while your system is an idling development testbed
  => Steal? No, Idle
Scheduling on Cores

- Most systems have more programs than CPUs
- So the OS will have to schedule them (time multiplexing)
- Such scheduling is called Context switching

- So in our metaphor we have multiple people of the same privilege level using a laptop …
Scheduling on Cores

- Well, that is easy a single program means no context switches
Scheduling on Cores

- With two programs the OS has to switch them every now and then
  - Every program shall get its fair share
  - Switching causes overhead
    - Some CPU time is no more used for the actual work done by the programs (red)
Scheduling on Cores

Applications trying to working

CPU busy, but primarily for overhead

Add steal time if you want real trouble

- With too much runnable programs per CPU the OS gets in trouble
  - Actually with any shared resource being shared a lot
- Eventually there are two bad options to chose from
  - Real throughput converging to zero (latency optimized)
  - Individual applications have to wait longer (throughput optimized)
Scheduling on Cores

- There are ways to switch cooperatively
  - On all blocking system calls like reads, timer sleeps
  - On explicit generic or directed context switch
    - local (yield / yield_to)
    - virtual (diagnose X'44'/ diagnose X'9C')
Scheduling on Cores

- There are ways to switch cooperatively … or not
  - OS can always interrupt and switch
  - Actually it looks more like one works and one has to wait, but we fight …

- greedy Applications
- OS has to interrupt and switch → non voluntary
Scheduling between Cores

- Most systems also have more than one CPU
- So the OS will have to schedule/dispatch programs on them
  - most “classic” System z Operating Systems use single queue dispatchers (z/OS, z/VM)
  - Linux has a multi queue scheduler (one queue per CPU)
    - Tasks are migrated to or pulled from cpus

- So in our metaphor we have
  - multiple people and a scheduler
  - in our office being a 4 laptop (CPU core)
Scheduling between Cores

- Single Queue scheduling
  - One scheduler instance directs programs to the CPU cores
  - Benefit of single control point and easy synchronization
Scheduling between Cores

- Multi queue scheduler
  - Here with the usual optimum of queues with 1 Program each
  - There are cases where it is better to leave one CPU idle
    - When two tasks are bound by the speed of their communication
  - You can also see topology here, as some cores are more “remote” than others

Applications in local runqueue

No need for the scheduler(s) to do anything
Scheduling between Cores

- Multi queue scheduler
  - Here one queue got rather full and a scheduler starts migrating tasks

3 Applications in local runqueue
Scheduling / Utilization – Combo Quiz

- Based on a real customer case
  - Database with a lot of stored procedures that does parallelization
  - 0-75 runnable processes varying a lot in a “spiky” fashion
  - They tried various setups, initially 10, later 4 to 20 real CPU cores
  - They didn't achieve their expected target utilization of 85%+ USR

- Question 1: Why was the utilization with 10 CPUs so low at an average of ~45% despite up to 75 runnable processes?
Scheduling / Utilization – Combo Quiz

- Question 1: why was the utilization so low at ~45%?
- Eventually one can never get >100%, but easily less
  - Could the scheduler do anything about it? → No
  - Would it be different with a single queue scheduler? → No

CPU Utilization of spiky applications

![Graph showing CPU Utilization over time](image-url)

10 CPUs - 0-75 Processes - oversimplified for illustrational purposes
Scheduling / Utilization – Combo Quiz

Question 2: Why did they achieve the following by changing cpu count?
– low # of CPUs: fully utilized, but now with a lot of SYS overhead
– high # of CPUs: even more underutilized but with almost no SYS overhead
Scheduling / Utilization – Combo Quiz

Question 2: Why did they achieve the following by changing cpu count?
- low # of CPUs: fully utilized, but now with a lot of SYS overhead
  - Context switch overhead
- high # of CPUs: even more underutilized but with almost no SYS overhead
  - Even more times of #runnable << #CPUs
Scheduling / Utilization – Combo Quiz

- Based on a real customer case
  - Database with a lot of stored procedures that does parallelization
  - Runnable processes varied a lot in a “spiky” fashion (range 0-75)
  - They tried various setups, initially 10, later 4 to 20 CPU cores
  - They didn't achieve their expected target utilization of 85%+ USR

- Eventually one can never get >100%, but easily less

- Question 3: Real fix approaches?
  - Application design (recommended)
  - Try to let the Hypervisor make underutilized resources otherwise usable (needs lower priority workload)
Page Cache

- Keeping disk data available in memory is caching
  - Certain strategies take place
    - What should be cached
    - Read ahead of data that will likely be used
    - Coalesce writes to issue a single disk write for several memory writes
  - Proper management of caches can be complex (read cpu intensive)

- Imagine processes are kids
  - There is obviously some privileged person (kernel) watching
  - If I learned something they surely will make a mess over time (dirty pages)
  - As long as the kids just watch all their toys (read) things stay clean
  - But when they really play with things (write) the room gets messy (dirty)
  - How are things cleaned up?
Page Cache – Cleaning I

- If things stay relatively clean nobody cleans anything
  Linux tunable: \( \% \text{dirty pages} < \text{dirty\_background\_ratio} \)
  - Only long unused items are put back where they belong
    Linux tunable: \( \text{dirty\_expire\_centisecs} \)
Page Cache – Cleaning II

- As long as the amount of dirtiness is in a sane range it is likely that the parents will clean a bit in background

Linux tunable: \( \text{dirty pages in } \% > \text{dirty\_background\_ratio} \)

- Cleaning of dirty pages consumes CPU, done by the kernel
- Run by the kswap thread(s) and accounted as SYS
- Kernel tries to be nice and stay in background
  (as you don't bother the kids too much the kernel tries not to take away cpu)
Page Cache – Cleaning III

- If the amount of dirtiness rises too a really high level the parents will force the kids to help cleaning up
  
  **Linux tunable:** (dirty pages in % > dirty\_ratio)
  
  - Now writing processes have to contribute parts of their time slices to the kernel
  - No more nice, but trying to stall those who make pages dirty
Page Cache – further details

- Another complex topic is the “proper” size of cache
  - Ever realized your flat/house is always too small except for cleaning it
  - If you use the kids room as office from 9am-6pm obviously toys have to be moved aside (cache shrink due to other workload)
  - On the other hand if you organize a party it is likely that they will consume more than just the kids room (cache grow due to more I/O)

- Even cleaning up before going to bed exists in IT
  - for actions like hibernate cache has to be cleaned up before going to sleep
Paging/Swapping

- Spending more memory than available is overcommitment
- In case the accessed memory exceeds the real memory paging has to take place
  - Paging (z/VM) is the same as swapping (Linux)

- As metaphor imagine a notebook page to be your memory ...
Paging/Swapping

- Initially a page is empty, so a process might write onto it
Paging/Swapping

- Later on the process (or someone else) might read from it
Paging/Swapping

- later the process might have something in mind that he wants to write to the next page
- But all pages the OS could provide are full
  - This now requires swapping (OS level)
  - Or paging (z/VM level)
Paging/Swapping

- The kernel takes over
  - Swaps out a page (based on least recently used plus some extras)
  - This makes room for a clean new page in real memory
  - Impact high, due to the orders of magnitude between disk and memory
- As you see this burden can literally break the back of your system :-)

Literally “back breaking”
Top

- Characteristics: Easy to use
- Objective: Shows resource usage on process level
- Usage: `top -b -d [interval in sec] > [outfile]`
- Package: RHEL: procps SLES: procps

- Shows
  - CPU utilization
  - Detailed memory usage

- Hints
  - Parameter -b enables to write the output for each interval into a file
  - Use -p [pid1, pid2,...] to reduce the output to the processes of interest
  - Configure displayed columns using 'f' key on the running top program
  - Use the 'W' key to write current configuration to ~/.toprc
    → becomes the default
top (cont.)

- Output

```
top - 11:12:52 up 1:11, 3 users, load average: 1.21, 1.61, 2.03
Tasks: 53 total, 5 running, 48 sleeping, 0 stopped, 0 zombie
Cpu(s): 3.0%us, 5.9%sy, 0.0%ni, 79.2%id, 9.9%wa, 0.0%hi, 1.0%si, 1.0%st
Mem: 5138052k total, 801100k used, 4336952k free, 447868k buffers
Swap: 88k total, 0k used, 88k free, 271436k cached
```

<table>
<thead>
<tr>
<th>PID</th>
<th>USER</th>
<th>PR</th>
<th>NI</th>
<th>VIRT</th>
<th>RES</th>
<th>SHR</th>
<th>S</th>
<th>%CPU</th>
<th>%MEM</th>
<th>TIME+</th>
<th>P</th>
<th>SWAP</th>
<th>DATA</th>
<th>WCHAN</th>
<th>COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>3224</td>
<td>root</td>
<td>18</td>
<td>0</td>
<td>1820</td>
<td>604</td>
<td>444</td>
<td>R</td>
<td>2.0</td>
<td>0.0</td>
<td>0:00.56</td>
<td>0</td>
<td>1216</td>
<td>252</td>
<td>-</td>
<td>dbench</td>
</tr>
<tr>
<td>3226</td>
<td>root</td>
<td>18</td>
<td>0</td>
<td>1820</td>
<td>604</td>
<td>444</td>
<td>R</td>
<td>2.0</td>
<td>0.0</td>
<td>0:00.56</td>
<td>0</td>
<td>1216</td>
<td>252</td>
<td>-</td>
<td>dbench</td>
</tr>
<tr>
<td>2737</td>
<td>root</td>
<td>16</td>
<td>0</td>
<td>9512</td>
<td>3228</td>
<td>2540</td>
<td>R</td>
<td>1.0</td>
<td>0.1</td>
<td>0:00.46</td>
<td>0</td>
<td>6284</td>
<td>868</td>
<td>-</td>
<td>sshd</td>
</tr>
<tr>
<td>3225</td>
<td>root</td>
<td>18</td>
<td>0</td>
<td>1820</td>
<td>604</td>
<td>444</td>
<td>R</td>
<td>1.0</td>
<td>0.0</td>
<td>0:00.56</td>
<td>0</td>
<td>1216</td>
<td>252</td>
<td>-</td>
<td>dbench</td>
</tr>
<tr>
<td>3230</td>
<td>root</td>
<td>16</td>
<td>0</td>
<td>2652</td>
<td>1264</td>
<td>980</td>
<td>R</td>
<td>1.0</td>
<td>0.0</td>
<td>0:00.01</td>
<td>0</td>
<td>1388</td>
<td>344</td>
<td>-</td>
<td>top</td>
</tr>
<tr>
<td>1</td>
<td>root</td>
<td>16</td>
<td>0</td>
<td>848</td>
<td>304</td>
<td>256</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00.54</td>
<td>0</td>
<td>544</td>
<td>232</td>
<td>select</td>
<td>init</td>
</tr>
<tr>
<td>2</td>
<td>root</td>
<td>RT</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00.00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>migration</td>
</tr>
<tr>
<td>3</td>
<td>root</td>
<td>34</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00.00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ksoftirqd</td>
</tr>
<tr>
<td>4</td>
<td>root</td>
<td>10</td>
<td>-5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00.13</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>worker_th</td>
</tr>
<tr>
<td>5</td>
<td>root</td>
<td>20</td>
<td>-5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00.00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>worker_th</td>
</tr>
</tbody>
</table>

- Hints
  - virtual memory: VIRT = SWAP + RES  unit KB
  - physical memory used: RES = CODE + DATA  unit KB
  - shared memory SHR  unit KB
Linux ps command

- Characteristics: very comprehensive, statistics data on process level
- Objective: reports a snapshot of the current processes
- Usage: “ps axlf”
- Package: RHEL: procps SLES: procps

Shows
- IDs: Pid, Tid, User, ...
- Status: stat and wchan
- Details: command, memory consumption and accumulated cpu time

### Example Output

<table>
<thead>
<tr>
<th>PID</th>
<th>TID</th>
<th>NLWP</th>
<th>POL</th>
<th>USER</th>
<th>TTY</th>
<th>NI</th>
<th>PRI</th>
<th>PSR</th>
<th>STAT</th>
<th>WCHAN</th>
<th>START</th>
<th>TIME</th>
<th>%CPU</th>
<th>%MEM</th>
<th>VSZ</th>
<th>SZ</th>
<th>RSS</th>
<th>COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>871</td>
<td>871</td>
<td>1</td>
<td>TS</td>
<td>root</td>
<td>?</td>
<td>-5</td>
<td>29</td>
<td>0</td>
<td>*S&lt;</td>
<td>kauditd_thre</td>
<td>10:01</td>
<td>00:00:00</td>
<td>0.0</td>
<td>0.0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2835</td>
<td>2835</td>
<td>1</td>
<td>TS</td>
<td>root</td>
<td>pts/2</td>
<td>0</td>
<td>23</td>
<td>0</td>
<td>*S+</td>
<td>read_chan</td>
<td>10:38</td>
<td>00:00:00</td>
<td>0.0</td>
<td>0.0</td>
<td>5140</td>
<td>824</td>
<td>2644</td>
<td>-bash</td>
</tr>
<tr>
<td>3437</td>
<td>3437</td>
<td>1</td>
<td>TS</td>
<td>root</td>
<td>pts/1</td>
<td>0</td>
<td>23</td>
<td>0</td>
<td>*S+</td>
<td>wait4</td>
<td>11:39</td>
<td>00:00:00</td>
<td>0.0</td>
<td>0.0</td>
<td>1816</td>
<td>248</td>
<td>644</td>
<td>-dbench 3</td>
</tr>
<tr>
<td>3438</td>
<td>3438</td>
<td>1</td>
<td>TS</td>
<td>root</td>
<td>pts/1</td>
<td>0</td>
<td>20</td>
<td>0</td>
<td>R+</td>
<td>-</td>
<td>11:39</td>
<td>00:00:24</td>
<td>33.1</td>
<td>0.0</td>
<td>1820</td>
<td>252</td>
<td>604</td>
<td>-dbench 3</td>
</tr>
</tbody>
</table>

### Hints
- Do not specify blanks inside the -o format string
- Status is a one time shot, most interactive or I/O bound processes might sleep
vmstat

- Characteristics: Easy to use, high-level information
- Objective: First and fast impression of the current state
- Usage: vmstat [interval in sec]
- Package: RHEL: sysstat.s390x SLES: sysstat
- Output sample:

```
vmstat 1

procs --- memory --- swap --- io --- system --- cpu ---
r  b   swpd  free  buff  cache  si  so  bi  bo  in  cs  us  sy  id  wa  st
2  2    0 4415152  64068  554100  0   0   4  63144 350  55  29  64  0  3  4
3  0    0 4417632  64832  551272  0   0   0  988  125  60  32  67  0  0  1
3  0    0 4411804  72188  549592  0   0   0  8984 230  42  32  67  0  0  1
3  0    0 4405232  72896  555592  0   0   0  16 106  52  32  68  0  0  0
```

- Shows
  - Data per time interval
  - CPU utilization
  - Disk I/O
  - Memory usage/Swapping

- Hints
  - Shared memory usage is listed under 'cache'
Thanks (and complaints) go to

Christian Ehrhardt
End of Part I
## Agenda

<table>
<thead>
<tr>
<th>Basic</th>
<th>Intermediate</th>
<th>Advanced</th>
<th>Master</th>
<th>Elite</th>
</tr>
</thead>
<tbody>
<tr>
<td>– Utilization</td>
<td>– General thoughts</td>
<td>– Strace</td>
<td>– Perf</td>
<td>– Cachestat</td>
</tr>
<tr>
<td>– Scheduling</td>
<td>– Sysstat</td>
<td>– Ltrace</td>
<td>– slabtop</td>
<td>– Smem</td>
</tr>
<tr>
<td>– Page Cache</td>
<td>– Dasdstat</td>
<td>– Lsof</td>
<td>– Blktrace</td>
<td>– Valgrind</td>
</tr>
<tr>
<td>– Swapping</td>
<td>– Scsi I/O</td>
<td>– Lsluns</td>
<td>– Ziomon</td>
<td>– Irqstats</td>
</tr>
<tr>
<td></td>
<td>– icastats</td>
<td>– Htop</td>
<td>Collection and Memory visualizer</td>
<td></td>
</tr>
<tr>
<td></td>
<td>– Lsqeth</td>
<td>– Netstat</td>
<td>– Jinsight</td>
<td></td>
</tr>
<tr>
<td></td>
<td>– Ethtool</td>
<td>– Socket Statistics</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>– Preparation</td>
<td>– Iptraf</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

49 March 3, 2015 Linux-Performance-know it all series © 2015 IBM Corporation
Non-legal Disclaimer

- This is an introduction and cheat sheet
  - Know what is out there
  - What could be useful in which case
  - How could I debug even further

- These descriptions are not full explanations
  - Most tools could get at least 1-2 presentations on their own
  - Don't start using them without reading howtos / man pages

- This is not about monitoring
  - Some tools used to start performance analysis CAN be monitors, but that's not part of the presentation
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

- Work with the tools before an issue occurs
  - Learn what should be normal on your system
## Orientation - where to go if something is broken

<table>
<thead>
<tr>
<th>Tool</th>
<th>Over-View</th>
<th>CPU cons.</th>
<th>lat.</th>
<th>Hot spots</th>
<th>Disk</th>
<th>Mem</th>
<th>Net</th>
</tr>
</thead>
<tbody>
<tr>
<td>top / ps</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sysstat</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>vmstat</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>iostat</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dasdstat</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>scsistat</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>netstat / ss</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>htop / dstat / pidstat</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>irqstats</td>
<td>x</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>strace / ltrace</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>hytop</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>perf</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>jinsight</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Health Center</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GMVC</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>blktrace / ziomon / lsof</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Valgrind / smem / slabtop</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>iptraf</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tracepoints</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>iotop</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>lszcrypt / icastat</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lsqeth / ethtool / iftop / iptraf / tcpdump / wireshark</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>lslns / multipath</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Preparation</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>
Sysstat - sadc/sar

- Characteristics: Very comprehensive, statistics data on device level
- Objective: Suitable for permanent system monitoring and detailed analysis
- Usage (recommended):
  - monitor /usr/lib64/sa/sadc [-S XALL] [interval in sec] [outfile]
  - View sar -A -f [outfile]
- Package: RHEL: sysstat.s390x SLES: sysstat
- Shows
  - CPU utilization
  - Disk I/O overview and on device level
  - Network I/O and errors on device level
  - Memory usage/Swapping
  - … and much more
  - Reports statistics data over time and creates average values for each item
- Hints
  - sadc parameter “-S XALL” enables the gathering of further optional data
  - Shared memory is listed under 'cache'
  - [outfile] is a binary file, which contains all values. It is formatted using sar
    - enables the creation of item specific reports, e.g. network only
    - enables the specification of a start and end time → time of interest
Processes created per second usually small except during startup. If constantly at a high rate your application likely has an issue. Be aware – the numbers scale with your system size and setup.
SAR - Context Switch Rate

Context switches per second usually < 1000 per cpu except during startup or while running a benchmark if > 10000 your application might have an issue.
## SAR - CPU utilization

Per CPU values:
- Watch out for:
  - System time (kernel)
  - User (applications)
  - Irq/soft (kernel, interrupt handling)
  - Idle (nothing to do)
  - Iowait time (runnable but waiting for I/O)
  - Steal time (runnable but utilized somewhere else)

### Table:

<table>
<thead>
<tr>
<th>Time</th>
<th>CPU</th>
<th>%user</th>
<th>%nice</th>
<th>%system</th>
<th>%iowait</th>
<th>%steal</th>
<th>%idle</th>
</tr>
</thead>
<tbody>
<tr>
<td>14:14:55</td>
<td>all</td>
<td>26.64</td>
<td>0.00</td>
<td>12.03</td>
<td>25.92</td>
<td>6.24</td>
<td>29.16</td>
</tr>
<tr>
<td>14:15:05</td>
<td>all</td>
<td>43.81</td>
<td>0.00</td>
<td>5.49</td>
<td>23.25</td>
<td>4.99</td>
<td>22.46</td>
</tr>
<tr>
<td>14:15:05</td>
<td>1</td>
<td>4.30</td>
<td>0.00</td>
<td>10.19</td>
<td>28.67</td>
<td>9.89</td>
<td>46.95</td>
</tr>
<tr>
<td>14:15:05</td>
<td>2</td>
<td>11.81</td>
<td>0.00</td>
<td>28.03</td>
<td>45.15</td>
<td>5.01</td>
<td>10.01</td>
</tr>
<tr>
<td>14:15:05</td>
<td>3</td>
<td>46.61</td>
<td>0.00</td>
<td>4.49</td>
<td>6.79</td>
<td>4.99</td>
<td>37.13</td>
</tr>
<tr>
<td>14:14:55</td>
<td>all</td>
<td>27.19</td>
<td>0.00</td>
<td>11.93</td>
<td>25.11</td>
<td>7.75</td>
<td>28.01</td>
</tr>
<tr>
<td>14:15:15</td>
<td>0</td>
<td>90.60</td>
<td>0.00</td>
<td>3.70</td>
<td>0.00</td>
<td>5.70</td>
<td>0.00</td>
</tr>
<tr>
<td>14:15:15</td>
<td>1</td>
<td>9.24</td>
<td>0.00</td>
<td>22.49</td>
<td>41.57</td>
<td>9.24</td>
<td>17.47</td>
</tr>
<tr>
<td>14:15:15</td>
<td>2</td>
<td>5.98</td>
<td>0.00</td>
<td>14.64</td>
<td>46.71</td>
<td>9.06</td>
<td>23.61</td>
</tr>
<tr>
<td>14:15:15</td>
<td>3</td>
<td>2.90</td>
<td>0.00</td>
<td>6.99</td>
<td>12.09</td>
<td>7.09</td>
<td>70.93</td>
</tr>
</tbody>
</table>
SAR - Network traffic

Per interface statistic of packets/bytes
You can easily derive average packet sizes from that.
Sometimes people expect - and planned for – different sizes.

Has another panel for errors, drops and such events.
### SAR – Disk I/O I – overall

<table>
<thead>
<tr>
<th>Time</th>
<th>tps</th>
<th>rtps</th>
<th>wtps</th>
<th>bread/s</th>
<th>bwrtn/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>14:14:55</td>
<td>445.71</td>
<td>61.38</td>
<td>384.33</td>
<td>7715.77</td>
<td>55529.74</td>
</tr>
<tr>
<td>14:15:05</td>
<td>192.20</td>
<td>32.90</td>
<td>159.30</td>
<td>7308.80</td>
<td>68233.60</td>
</tr>
<tr>
<td>14:15:15</td>
<td>171.70</td>
<td>1.20</td>
<td>170.50</td>
<td>9.60</td>
<td>70798.40</td>
</tr>
<tr>
<td>14:15:25</td>
<td>327.25</td>
<td>174.95</td>
<td>152.30</td>
<td>1399.60</td>
<td>68261.88</td>
</tr>
<tr>
<td>14:15:35</td>
<td>444.74</td>
<td>310.51</td>
<td>134.23</td>
<td>2484.88</td>
<td>59704.50</td>
</tr>
<tr>
<td>Average:</td>
<td>316.35</td>
<td>116.15</td>
<td>200.20</td>
<td>3784.61</td>
<td>64504.50</td>
</tr>
</tbody>
</table>

Overview of
- operations per second
- transferred amount
Is your I/O balanced across devices? Imbalances can indicate issues with a LV setup.

tps and avgrq-sz combined can be important. Do they match your sizing assumptions?

Await shows the time the application has to wait.
SAR - Memory statistics - the false friend

Be aware that high %memused and low kbmemfree is no indication of a memory shortage (common mistake).

Same for swap – to use swap is actually good, but to access it (swapin/-out) all the time is bad.
SAR - Memory pressure - Swap

The percentage seen before can be high, but the swap rate shown here should be low. Ideally it is near zero after a rampup time. High rates can indicate memory shortages.
SAR - Memory pressure – faults and reclaim

Don't trust pgpgin/-out absolute values
Faults populate memory
Major faults need I/O
Scank/s is background reclaim by kswap/flush (modern)
Scand/s is reclaim with a “waiting” allocation
Steal is the amount reclaimed by those scans
SAR - System Load

Runqueue size are the currently runnable programs. It's not bad to have many, but if they exceed the amount of CPUs you could do more work in parallel.

Plist-sz is the overall number of programs, if that is always growing you have likely a process starvation or connection issue.

Load average is a runqueue length average for 1/5/15 minutes.
Sysstat - iostat

- Characteristics: Easy to use, information on disk device level
- Objective: Detailed input/output disk statistics
- Usage: iostat -xtdk [interval in sec]
- Package: RHEL: sysstat.s390x SLES: sysstat

- Shows
  - Throughput
  - Request merging
  - Device queue information
  - Service times

- Hints
  - Most critical parameter often is *await*
    - average time (in milliseconds) for I/O requests issued to the device to be served.
    - includes the time spent by the requests in queue and the time spent servicing them.
  - Also suitable for network file systems
### iostat

#### Output sample:

**Time: 10:56:35 AM**

<table>
<thead>
<tr>
<th>Device</th>
<th>rrqm/s</th>
<th>wrqm/s</th>
<th>r/s</th>
<th>w/s</th>
<th>rkB/s</th>
<th>wkB/s</th>
<th>avgrq-sz</th>
<th>avgqu-sz</th>
<th>await</th>
<th>svctm</th>
<th>%util</th>
</tr>
</thead>
<tbody>
<tr>
<td>dasda</td>
<td>0.19</td>
<td>1.45</td>
<td>1.23</td>
<td>0.74</td>
<td>64.43</td>
<td>9.29</td>
<td>74.88</td>
<td>0.01</td>
<td>2.65</td>
<td>0.80</td>
<td>0.16</td>
</tr>
<tr>
<td>dasdb</td>
<td>0.02</td>
<td>232.93</td>
<td>0.03</td>
<td>9.83</td>
<td>0.18</td>
<td>975.17</td>
<td>197.84</td>
<td>0.98</td>
<td>99.80</td>
<td>1.34</td>
<td>1.33</td>
</tr>
</tbody>
</table>

**Time: 10:56:36 AM**

<table>
<thead>
<tr>
<th>Device</th>
<th>rrqm/s</th>
<th>wrqm/s</th>
<th>r/s</th>
<th>w/s</th>
<th>rkB/s</th>
<th>wkB/s</th>
<th>avgrq-sz</th>
<th>avgqu-sz</th>
<th>await</th>
<th>svctm</th>
<th>%util</th>
</tr>
</thead>
<tbody>
<tr>
<td>dasda</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>dasdb</td>
<td>0.00</td>
<td>1981.55</td>
<td>0.00</td>
<td>339.81</td>
<td>0.00</td>
<td>9495.15</td>
<td>55.89</td>
<td>0.91</td>
<td>2.69</td>
<td>1.14</td>
<td>38.83</td>
</tr>
</tbody>
</table>

**Time: 10:56:37 AM**

<table>
<thead>
<tr>
<th>Device</th>
<th>rrqm/s</th>
<th>wrqm/s</th>
<th>r/s</th>
<th>w/s</th>
<th>rkB/s</th>
<th>wkB/s</th>
<th>avgrq-sz</th>
<th>avgqu-sz</th>
<th>await</th>
<th>svctm</th>
<th>%util</th>
</tr>
</thead>
<tbody>
<tr>
<td>dasda</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>dasdb</td>
<td>0.00</td>
<td>2055.00</td>
<td>0.00</td>
<td>344.00</td>
<td>0.00</td>
<td>9628.00</td>
<td>55.98</td>
<td>1.01</td>
<td>2.88</td>
<td>1.19</td>
<td>41.00</td>
</tr>
</tbody>
</table>

Recent versions are improved by reporting reads/writes separately which is great as they have vastly different characteristics.
Sysstat - PIDSTAT

- Characteristics: Easy to use extended per process statistics
- Objective: Identify processes with peak activity
- Usage: `pidstat [w] [-r] [-d]
- Package: RHEL: sysstat SLES: sysstat

- 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

### Voluntarily / Involuntary

<table>
<thead>
<tr>
<th>Time</th>
<th>PID</th>
<th>minflt/s</th>
<th>majflt/s</th>
<th>VSZ</th>
<th>RSS</th>
<th>%MEM</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>12:47:51 PM</td>
<td>985</td>
<td>0.06</td>
<td>0.00</td>
<td>15328</td>
<td>3948</td>
<td>0.10</td>
<td>smbd</td>
</tr>
<tr>
<td>12:47:51 PM</td>
<td>992</td>
<td>0.04</td>
<td>0.00</td>
<td>5592</td>
<td>2152</td>
<td>0.05</td>
<td>sshd</td>
</tr>
<tr>
<td>12:47:51 PM</td>
<td>1073</td>
<td>526.41</td>
<td>0.00</td>
<td>1044240</td>
<td>321512</td>
<td>7.89</td>
<td>Xorg</td>
</tr>
</tbody>
</table>

### Faults per process

<table>
<thead>
<tr>
<th>Time</th>
<th>PID</th>
<th>kB_rd/s</th>
<th>kB_wr/s</th>
<th>kB_ccwr/s</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>12:49:18 PM</td>
<td>330</td>
<td>0.00</td>
<td>1.15</td>
<td>0.00</td>
<td>sshd</td>
</tr>
<tr>
<td>12:49:18 PM</td>
<td>2899</td>
<td>4.35</td>
<td>0.09</td>
<td>0.04</td>
<td>notes2</td>
</tr>
<tr>
<td>12:49:18 PM</td>
<td>3045</td>
<td>23.43</td>
<td>0.01</td>
<td>0.00</td>
<td>audacious2</td>
</tr>
</tbody>
</table>

### How much KB disk I/O per process
Sysstat - mpstat

- Characteristics: Show statistics per processor
- Objective: Identify imbalanced utilization or interrupt peaks
- Usage: mpstat -A <interval>
- Package: RHEL: sysstat SLES: sysstat

- Shows
  - -u utilization
  - -I <CPU|SCPU|ALL> Interrupts

- Hints
  - Can be restricted to selected processor(s) (-P)
Sysstat – mpstat example

- As one can see there are plenty of different (s)irq sources these days
  - Ordered horizontally per type and vertically per cpu

**IRQs**

<table>
<thead>
<tr>
<th>Time</th>
<th>CPU</th>
<th>EXT/s</th>
<th>I/O/s</th>
<th>AIO/s</th>
<th>CLK/s</th>
<th>EXC/s</th>
<th>EMS/s</th>
<th>TMR/s</th>
<th>TAL/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>10:40:12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PFL/s</td>
<td>DSD/s</td>
<td>VRT/s</td>
<td>SCP/s</td>
<td>IUC/s</td>
<td>CMS/s</td>
<td>CMC/s</td>
<td>CMR/s</td>
<td>CIO/s</td>
<td>QAI/s</td>
</tr>
<tr>
<td>DAS/s</td>
<td>C15/s</td>
<td>C70/s</td>
<td>TAP/s</td>
<td>VMR/s</td>
<td>LCS/s</td>
<td>CLW/s</td>
<td>CTC/s</td>
<td>APB/s</td>
<td>ADM/s</td>
</tr>
<tr>
<td>CSC/s</td>
<td>PCI/s</td>
<td>MSI/s</td>
<td>VIR/s</td>
<td>VAI/s</td>
<td>NMI/s</td>
<td>RST/s</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
10:40:17     0     2.40     0.00     0.00     2.00     0.20     0.20     0.00     0.00
0.00     0.00     0.00     0.00     0.00     0.00     0.00     0.00     0.00
0.00     0.00     0.00     0.00     0.00     0.00     0.00     0.00     0.00
0.00     0.00     0.00     0.00     0.00     0.00     0.00     0.00     0.00
10:40:17     1     1.40     0.00     0.00     0.00     0.00     0.00     0.00     0.00
0.00     0.00     0.00     0.00     0.00     0.00     0.00     0.00     0.00
0.00     0.00     0.00     0.00     0.00     0.00     0.00     0.00     0.00
0.00
```

**SoftIRQs**

<table>
<thead>
<tr>
<th>Time</th>
<th>CPU</th>
<th>HI/s</th>
<th>TIMER/s</th>
<th>NET_TX/s</th>
<th>NET_RX/s</th>
<th>BLOCK/s</th>
<th>BLOCK_IOPOLL/s</th>
<th>TASKLET/s</th>
<th>SCHED/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>10:40:26</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HRTIMER/s</td>
<td>RCU/s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
10:40:31     0     0.00     0.60     0.00     0.00     0.00     0.00     0.00     1.00
0.00     0.00     0.60     0.00     0.00     0.00     0.00     0.00     0.00
10:40:31     1     0.00     0.00     0.00     0.00     0.00     0.00     0.00     0.00
0.00     0.00     0.00     0.00     0.00     0.00     0.00     0.00     0.00
```

CPU | IRQ type | Rate per second
--- | --- | ---
... | CPU | IRQ type | Rate per second
... | ... | ... | ...
DASD statistics

- Characteristics: Easy to use, very detailed
- Objective: Collects statistics of I/O operations on DASD devices
- Usage:
  - enable: echo on > /proc/dasd/statistics
  - show:
    - Overall: cat /proc/dasd/statistics
    - for individual DASDs: tunedasd -P /dev/dasda
- Package: n/a for kernel interface, s390-tools for dasdstat

- Shows:
  - various processing times:
    - Start: Histogram of I/O till ssch
      - Build channel program
      - wait till subchannel is free
    - Histogram of I/O between ssch and IRQ
      - Processing data transfer from/to storage server
    - Histogram between I/O and End
      - Tell block dev layer
      - Data has arrived
    - End: Histogram of I/O times
      - Tool “dasdstat” available to handle that all-in-one
DASD statistics – report

- Sample:
  - $8 \times 512b = 4KB \leq\text{ request size } < 16 \times 512b = 8KB$
  - $1 \text{ms } \leq\text{ response time } < 2 \text{ms}$

- Histogram of sizes (512B secs)
  - 0 0 9925 3605 1866 4050 4102 933 2700 2251 0 0 0 0 0 0
  - 0 0 0 0 0 0 0 1283 1249 6351 7496 3658 8583 805 7 0

- Histogram of I/O times (microseconds)
  - 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  - 0 0 0 0 0 0 0 14018 7189 2402 1031 4758 27 4 3 0

- Histogram of I/O time till ssch - look here for subchannel busy
  - 2314 283 98 34 13 5 16 275 497 8917 5567 4232 7117 60 4 0
  - 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

- Histogram of I/O time between ssch and irq - look here for slow SAN
  - 0 0 0 0 0 0 14018 7189 2402 1031 4758 27 4 3 0
  - 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

- Histogram of I/O time between irq and end
  - 2733 6 5702 9376 5781 940 1113 3781 0 0 0 0 0 0 0 0
  - 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

- # of req in chnx at enqueuing (1..32)
  - 0 2740 628 1711 1328 23024 0 0 0 0 0 0 0 0 0 0
  - 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

- Hints
  - Also shows data per sector which usually is only confusing
DASD statistics – look for subchannel busy issues

- "Build subchannel program (usually very fast)
- "Wait for a free subchannel (can be long without or too few HPAV)
- "Please be aware that the x axis is scaling by $2^n$

Time consists of:

- **Build subchannel program** (usually very fast)
- **Wait for a free subchannel** (can be long without or too few HPAV)
- Please be aware that the x axis is scaling by $2^n$
FCP statistics

- Characteristics: Detailed latency information for FCP I/O
- Objective: Collect details of I/O operations on FCP devices
- Package: n/a (Kernel interface)

Usage:
- enable
  - `CONFIG_STATISTICS=y` must be set in the kernel config file
  - debugfs is mounted at `/sys/kernel/debug/
  - For a certain LUN in directory
    `/sys/kernel/debug/statistics/zfcp-<device-bus-id>-<WWPN>-<LUN>`
    `issue echo on=1 > definition` (turn off with on=0, reset with data=reset)
- view
  - `cat /sys/kernel/debug/statistics/zfcp-<device-bus-id>-<WWPN>-<LUN>/data`

Hint
- FCP and DASD statistics are not directly comparable, because in the FCP case many I/O requests can be sent to the same LUN before the first response is given. There is a queue at FCP driver entry and in the storage server
FCP statistics

- Shows:
  - Request sizes in bytes (hexadecimal)
  - Channel latency in ns Time spent on the FCP channel (internal transfer)
  - Fabric latency in ns Time spent in the FCP fabric (outside transfer)
  - (Overall) latencies whole time spent entry/exit of the zFCP layer in ms

<table>
<thead>
<tr>
<th>SCSI stack</th>
<th>zfcp</th>
<th>I/O passes through QDIO and z/VM</th>
<th>I/O queued for transm.</th>
<th>Fabric</th>
<th>Completion handling</th>
<th>IRQ handling by zfcp</th>
<th>Completion handled by SCSI</th>
</tr>
</thead>
<tbody>
<tr>
<td>preps I/O</td>
<td>preps I/O</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>= blktrace dispatch</td>
<td>zfcp issues I/O</td>
<td></td>
<td></td>
<td>Fabric</td>
<td>resp. received by card</td>
<td>Receive IRQ</td>
<td>Received by SCSI stack</td>
</tr>
<tr>
<td>seen in blktrace</td>
<td>I/O received by card</td>
<td>I/O issued by card</td>
<td></td>
<td>Channel</td>
<td>Fabric</td>
<td>Channel</td>
<td>Overall Latency</td>
</tr>
</tbody>
</table>

Channel, Fabric, Overall Latency, Completion handled by SCSI
## FCP statistics

- On popular request – the “where to complain” color coding
  - Linux developers / Distributor in general
  - zfcp/qdio (optional also z/VM) developers
  - FCP card HW/FW stack
  - SAN

<table>
<thead>
<tr>
<th>SCSI stack preps I/O</th>
<th>zfcp preps I/O</th>
<th>I/O passes through QDIO and z/VM</th>
<th>I/O queued for transm.</th>
<th>Fabric</th>
<th>Completion handling</th>
<th>IRQ handling by zfcp</th>
<th>Completion handled by SCSI</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

seen in blktrace:
- SCSI stack issues I/O
- zfcp issues I/O
- zfcp preps I/O
- I/O received by card
- I/O issued by card
- Reponse received by card
- Receive IRQ
- SCSI stack completed

overall latency:
- channel
- fabric
- channel

seen in blktrace:
- overall latency
- completion handled by SCSI stack
- SCSI stack completed
- receive irq
- response received by card
- i/o issued by card
- i/o received by card
- zfcp issues i/o
- zfcp preps i/o
- i/o passes through qdio and z/VM
- i/o queued for transm.
- fabric
- completion handling
- irq handling by zfcp
- completion handled by scsi stack
- seen in blktrace
FCP statistics example – rather unreadable

cat /sys/kernel/debug/statistics/zfcp-0.0.1700-0x5005076303010482-0x4014400500000000/data

... request_sizes_scsi_read 0x1000 1163
request_sizes_scsi_read 0x80000 805
request_sizes_scsi_read 0x54000 47
request_sizes_scsi_read 0x2d000 44
request_sizes_scsi_read 0x2a000 26
request_sizes_scsi_read 0x57000 25
request_sizes_scsi_read 0x1e000 25
...

... latencies_scsi_read <=1 1076
latencies_scsi_read <=2 205
latencies_scsi_read <=4 575
latencies_scsi_read <=8 368
latencies_scsi_read <=16 0
...

... channel_latency_read <=16000 0
channel_latency_read <=32000 983
channel_latency_read <=64000 99
channel_latency_read <=128000 115
channel_latency_read <=256000 753
channel_latency_read <=512000 106
channel_latency_read <=1024000 141
channel_latency_read <=2048000 27
channel_latency_read <=4096000 0
...

... fabric_latency_read <=1000000 1238
fabric_latency_read <=2000000 328
fabric_latency_read <=4000000 522
fabric_latency_read <=8000000 136
fabric_latency_read <=16000000 0
...

request size 4KB, 1163 occurrences

response time <= 1ms

Channel response time <= 32μs
= all below driver

Fabric response time <= 1ms
= once leaving the card
FCP statistics example

- Some statistics are per device
- Some per adapter
  - Adapter statistics can not be reset
- Example beautifying the same data with a little script (not public yet)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>sda</td>
<td>92</td>
<td>130</td>
<td>1386</td>
<td>126.00</td>
<td>7</td>
<td>10</td>
<td>98</td>
<td>8.91</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>sdb</td>
<td>127</td>
<td>131</td>
<td>2072</td>
<td>129.50</td>
<td>7</td>
<td>10</td>
<td>140</td>
<td>8.75</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>sdc</td>
<td>126</td>
<td>140</td>
<td>2075</td>
<td>129.69</td>
<td>7</td>
<td>14</td>
<td>145</td>
<td>9.06</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>sdd</td>
<td>126</td>
<td>132</td>
<td>1160</td>
<td>128.89</td>
<td>7</td>
<td>8</td>
<td>74</td>
<td>8.22</td>
<td>9</td>
<td></td>
</tr>
</tbody>
</table>

[...]

| sdbd | n/a     | n/a     | 0       | 0.00    | n/a     | n/a     | 0       | 0.00    | 0      |                |
| sdbe  | n/a     | n/a     | 0       | 0.00    | n/a     | n/a     | 0       | 0.00    | 0      |                |

per adapter latency statistics (f - fabric; c - channel)

<table>
<thead>
<tr>
<th>adapter (subch/dev)</th>
<th>rd-cnt</th>
<th>rd-mb</th>
<th>rd-avgsz</th>
<th>wr-cnt</th>
<th>wr-mb</th>
<th>wr-avgsz</th>
<th>cmd-cnt</th>
<th>sec-active</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0.0004/0.0.1700</td>
<td>4899</td>
<td>18</td>
<td>3.76</td>
<td>11572</td>
<td>1249</td>
<td>110.52</td>
<td>240</td>
<td>17324</td>
</tr>
<tr>
<td>0.0.000c/0.0.1800</td>
<td>4901</td>
<td>16</td>
<td>3.34</td>
<td>11564</td>
<td>1265</td>
<td>112.02</td>
<td>236</td>
<td>17325</td>
</tr>
<tr>
<td>0.0.00d6/0.0.5100</td>
<td>4765</td>
<td>16</td>
<td>3.44</td>
<td>11595</td>
<td>1254</td>
<td>110.75</td>
<td>239</td>
<td>17318</td>
</tr>
<tr>
<td>0.0.00e2/0.0.5b00</td>
<td>1888</td>
<td>5</td>
<td>2.71</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
<td>160</td>
<td>17309</td>
</tr>
</tbody>
</table>
**iotop**

- **Characteristics:** simple, top like I/O monitor
- **Objective:** Check which processes are doing I/O
- **Usage:** `iotop`
- **Package:** RHEL: iotop SLES: iotop

- **Shows**
  - Read/Write per thread
  - Can accumulate (-a) for updating summaries instead of live views
    - Useful for Disk I/O tests that don't account on their own
  - Separate accounting for swap

- **Hints**
  - Can be restricted to certain processes via (-p)
  - Has a batch mode like top
### Iotop - examples

**Example I: disk I/O can be spread other than expected**

<table>
<thead>
<tr>
<th>TID</th>
<th>PRI</th>
<th>USER</th>
<th>DISK READ</th>
<th>DISK WRITE</th>
<th>SWAPIN</th>
<th>IO</th>
<th>COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>7204</td>
<td>be/4</td>
<td>qemu</td>
<td>378.16 K/s</td>
<td>0.00 B/s</td>
<td>0.00 %</td>
<td>6.16 % qemu-system-s390x -machine accel=kvm -name p1035002 -S -m</td>
<td></td>
</tr>
<tr>
<td>7231</td>
<td>be/4</td>
<td>qemu</td>
<td>350.87 K/s</td>
<td>0.00 B/s</td>
<td>0.00 %</td>
<td>6.08 % qemu-system-s390x -machine accel=kvm -name p1035002 -S -m</td>
<td></td>
</tr>
<tr>
<td>7228</td>
<td>be/4</td>
<td>qemu</td>
<td>343.08 K/s</td>
<td>19.49 K/s</td>
<td>0.00 %</td>
<td>6.00 % qemu-system-s390x -machine accel=kvm -name p1035002 -S -m</td>
<td></td>
</tr>
<tr>
<td>7174</td>
<td>be/4</td>
<td>qemu</td>
<td>340.97 K/s</td>
<td>0.00 B/s</td>
<td>0.00 %</td>
<td>6.00 % qemu-system-s390x -machine accel=kvm -name p1035002 -S -m</td>
<td></td>
</tr>
<tr>
<td>7203</td>
<td>be/4</td>
<td>qemu</td>
<td>362.57 K/s</td>
<td>0.00 B/s</td>
<td>0.00 %</td>
<td>5.96 % qemu-system-s390x -machine accel=kvm -name p1035002 -S -m</td>
<td></td>
</tr>
</tbody>
</table>

**System wide totals**

**Read / Write per process**

**Example II: even interesting for memory bound (overcommitted) loads**

<table>
<thead>
<tr>
<th>TIME</th>
<th>TID</th>
<th>PRI</th>
<th>USER</th>
<th>DISK READ</th>
<th>DISK WRITE</th>
<th>SWAPIN</th>
<th>IO</th>
<th>COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>08:13:16</td>
<td>971957.16 K/s</td>
<td>0.00 %</td>
<td>35.54 %</td>
<td>2.99 %</td>
<td>/mempighd-pf2 -s 2240 -d 1200 -p 32 -w 32 -x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>08:13:16</td>
<td>4057.11 K/s</td>
<td>0.00 %</td>
<td>23.10 %</td>
<td>2.98 %</td>
<td>/mempighd-pf2 -s 2240 -d 1200 -p 32 -w 32 -x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>08:13:16</td>
<td>6496.12 K/s</td>
<td>0.00 %</td>
<td>28.59 %</td>
<td>2.86 %</td>
<td>/mempighd-pf2 -s 2240 -d 1200 -p 32 -w 32 -x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>08:13:16</td>
<td>6563.95 K/s</td>
<td>0.00 %</td>
<td>42.70 %</td>
<td>2.78 %</td>
<td>/mempighd-pf2 -s 2240 -d 1200 -p 32 -w 32 -x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>08:13:16</td>
<td>6796.11 K/s</td>
<td>0.00 %</td>
<td>30.59 %</td>
<td>2.76 %</td>
<td>/mempighd-pf2 -s 2240 -d 1200 -p 32 -w 32 -x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>08:13:16</td>
<td>7033.54 K/s</td>
<td>0.00 %</td>
<td>30.03 %</td>
<td>2.74 %</td>
<td>/mempighd-pf2 -s 2240 -d 1200 -p 32 -w 32 -x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>08:13:16</td>
<td>5374.43 K/s</td>
<td>0.00 %</td>
<td>24.38 %</td>
<td>2.71 %</td>
<td>/mempighd-pf2 -s 2240 -d 1200 -p 32 -w 32 -x</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Lszcrypt / icastats

- Characteristics: overview of s390 crypto HW and libica usage
- Objective: am I really using my crypto hardware
- Usage: “lszcrypt -VV[V]” “cat /proc/icastats”
- Package: RHEL: s390utils-base SLES: s390-tools

```
lszcrypt -VV
request_count=443
`card02: CEX3C  hwtype=9  depth=8`
request_count=0
`card03: CEX3A  hwtype=8  depth=8`
```

```
Card/proc/icastats

<table>
<thead>
<tr>
<th>function</th>
<th># hardware</th>
<th># software</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHA-1</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>SHA-224</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SHA-256</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SHA-384</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SHA-512</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>RANDOM</td>
<td>187109</td>
<td>0</td>
</tr>
<tr>
<td>MOD EXPO</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>RSA CRT</td>
<td>93554</td>
<td>0</td>
</tr>
<tr>
<td>DES ENC</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DES DEC</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3DES ENC</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3DES DEC</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>AES ENC</td>
<td>2574106</td>
<td>0</td>
</tr>
<tr>
<td>AES DEC</td>
<td>2075854</td>
<td>0</td>
</tr>
<tr>
<td>CMAC GEN</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>CMAC VER</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
```

- Never assume your HW correctly is used until you confirmed it
  - If not going via libica (e.g. Java pkcs#11 you won't see it in icastat)
**lsqeth**

- **Characteristics:** overview of network devices
- **Objective:** check your network devices basic setup
- **Usage:** “`lsqeth -p`”
- **Package:** RHEL: s390-utils-base SLES: s390-tools

```plaintext
lsqeth -p
```

<table>
<thead>
<tr>
<th>devices</th>
<th>CHPID</th>
<th>interface</th>
<th>cardtype</th>
<th>port</th>
<th>chksum</th>
<th>prio-q'ing</th>
<th>rtr4</th>
<th>rtr6</th>
<th>lay'2</th>
<th>cnt</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0.e000/0.0.e001/0.0.e002</td>
<td>x84</td>
<td>eth1</td>
<td>OSD_10GIG</td>
<td>0</td>
<td>sw</td>
<td>always_q_2</td>
<td>n/a</td>
<td>n/a</td>
<td>1</td>
<td>64</td>
</tr>
<tr>
<td>0.0.e100/0.0.e101/0.0.e102</td>
<td>x85</td>
<td>eth2</td>
<td>OSD_10GIG</td>
<td>0</td>
<td>sw</td>
<td>always_q_2</td>
<td>n/a</td>
<td>n/a</td>
<td>1</td>
<td>64</td>
</tr>
<tr>
<td>0.0.f200/0.0.f201/0.0.f202</td>
<td>x6B</td>
<td>eth0</td>
<td>OSD_1000</td>
<td>0</td>
<td>hw</td>
<td>always_q_2</td>
<td>no</td>
<td>no</td>
<td>0</td>
<td>64</td>
</tr>
</tbody>
</table>

- **Check for layer, offload, and buffer counts**
  - More buffers are usually better especially for massive amounts of concurrent connections
Ethtool I

- **Characteristics**: overview of network device capabilities / offload settings
- **Objective**: check your network device (offload) settings
- **Usage**: “ethtool <dev>”, “ethtool -k <dev>”
- **Package**: RHEL: ethtool SLES: ethtool

```
ethtool eth1
Settings for eth1:
  Supported ports: [ FIBRE ]
  Supported link modes:  10baseT/Half 10baseT/Full
                        100baseT/Half 100baseT/Full
                        1000baseT/Half 1000baseT/Full
                        10000baseT/Full
  Supported pause frame use: No
  Supports auto-negotiation: Yes
  Advertised link modes:  10baseT/Half 10baseT/Full
                          100baseT/Half 100baseT/Full
                          1000baseT/Half 1000baseT/Full
                          10000baseT/Full
  Advertised pause frame use: No
  Advertised auto-negotiation: Yes
  Speed: 10000Mb/s
  Duplex: Full
  Port: FIBRE
  PHYAD: 0
  Transceiver: internal
  Auto-negotiation: on
  Link detected: yes
```

- **Check e.g. announced speeds**
Ethtool II

- Offload Settings via "ethtool -k <dev>"
- Changes via upper case "-K"

```
ethtool -k eth1
Features for eth1:
rx-checksumming: off [fixed]
tx-checksumming: off
  tx-checksum-ipv4: off [fixed]
tx-checksum-ipv6: off [fixed]
tx-checksum-fcoe-crc: off [fixed]
tx-checksum-sctp: off [fixed]
scatter-gather: off
  tx-scatter-gather: off [fixed]
tx-scatter-gather-fraglist: off [fixed]
tcp-segmentation-offload: off
  tx-tcp-segmentation: off [fixed]
tx-tcp-ecn-segmentation: off [fixed]
tx-tcp6-segmentation: off [fixed]
udp-fragmentation-offload: off [fixed]
generic-segmentation-offload: off [requested on]
generic-receive-offload: on
large-receive-offload: off [fixed]
rx-vlan-offload: off [fixed]
rx-vlan-offload: off [fixed]
rx-vlan-offload: off [fixed]
rx-vlan-offload: off [fixed]
rx-vlan-offload: off [fixed]
rx-vlan-offload: off [fixed]
rx-vlan-offload: off [fixed]
receive-hash-randomize: on
receive-hash-hmac: on
receive-hash-lng: on
receive-hash-opt: on
receive-hash-secure: on
receive-hash-tcp: on
receive-hash-udp: on
scatter-gather: off
udp-fragmentation-offload: off [fixed]
generic-segmentation-offload: off [requested on]
rx-gro: on
rx-gso: on
rx-gso-robust: on
ntuple-filters: off [fixed]
receive-hashing: off [fixed]
highdma: off [fixed]
rx-vlan-filter: on [fixed]
vlan-challenged: off [fixed]
tx-lockless: off [fixed]
netns-local: off [fixed]
tx-gso-robust: off [fixed]
tx-fcoe-segmentation: off [fixed]
tx-gre-segmentation: off [fixed]
tx-udp-tnl-segmentation: off [fixed]
fcoe-mtu: off [fixed]
tx-nocache-copy: off
loopback: off [fixed]
rx-fcs: off [fixed]
rx-all: off [fixed]
tx-vlan-stag-hw-insert: off [fixed]
rx-vlan-stag-hw-parse: off [fixed]
rx-vlan-stag-filter: off [fixed]
```

- In some cases external influences like OSA-layer2 prevent most offloads (the example here)
Don't miss preparation

- Of all tools preparation is clearly
  - The most important
  - The most effective

- Prepare
  - System and Workload descriptions
  - Healthy system data for comparison

- Gather
  - In case of emergency

- Report
  - How to report a Problem Description

- Solve
  - Tools to start an analysis

[Diagram of the preparation, gather, report, solve process]
Don't miss preparation

- Of all tools preparation is clearly
  - The most important
  - The most effective

- Prepare
  - System and Workload descriptions
  - Healthy system data for comparison

- Gather
  - In case of emergency

- Report
  - How to report a Problem Description

- Solve
  - Tools to start an analysis

This is like “Heisenbergs uncertainty principle”
The more time you put into preparation,
the less time you'll need to solve issues
They fundamentally are never both huge,
What do you prefer?
Don't miss preparation

▪ Of all tools preparation is clearly
  – The most important
  – The most effective

▪ Prepare
  – System and Workload descriptions
  – Healthy system data for comparison

▪ Gather
  – In case of emergency

▪ Report
  – How to report a Problem Description

▪ Solve
  – Tools to start an analysis

This is like “Heisenbergs uncertainty principle”
The more time you put into preparation,
the less time you'll need to solve issues
They fundamentally are never both huge,
What do you prefer?
.. combined with Murphy: there is always a bug
Don't miss preparation

- Of all tools preparation is clearly
  - The most important
  - The most effective

- Prepare
  - System and Workload descriptions
  - Healthy system data for comparison

- Gather
  - In case of emergency

- Report
  - How to report a Problem Description

- Solve
  - Tools to start an analysis

This is like “Heisenbergs uncertainty principle”
The more time you put into preparation, the less time you'll need to solve issues
They fundamentally are never both huge, What do you prefer?

.. combined with Murphy: there is always a bug
That means with enough preparation you'll surely get a bug that no one can fix, so you least get famous for finding the final bug
Share: Your chance to win … some love

- Thanks for staying with me, now your final test

- When should you start using all those tools?
  - Valid answers:
    - prior to error
    - tomorrow
    - in preparation
    - better soon
    - with any new hire
    - immediately
    - “silence” as you are already busy planning to work with it
Thanks (and complaints) go to

Christian Ehrhardt
End of Part II
# Agenda

## Basic
- Utilization
- Scheduling
- Page Cache
- Swapping
- top
- ps
- vmstat

## Intermediate
- General thoughts
- Sysstat
- Dasdstat
- Scsi I/O statistics
- iotop
- Lschrpt
- Icastats
- Lsqqeth
- Ethtool
- Preparation

## Advanced
- Strace
- Ltrace
- Lsof
- Lsluns
- Multipath
- Hytop
- Dstat
- Htop
- Netstat
- Socket Statistics
- Iptraf

## Master
- Perf
- Slabtop
- Blktrace
- Ziomon
- Tcpdump
- Java Health Center
- Java Garbage Collection and Memory visualizer
- Jinsight

## Elite
- Cachestat
- Smem
- Valgrind
- Irqstats
- Wireshark
- Kernel Tracepoints
- Systemtap
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]`
- Package: RHEL: strace SLES: strace

- 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 timestamp (-r)

- Hints
  - The option “-c” allows medium overhead by just tracking counters and durations
**strace - example**

```bash
strace -cf -p 26802
Process 26802 attached - interrupt to quit
^Process 26802 detached
```

<table>
<thead>
<tr>
<th>% time</th>
<th>seconds</th>
<th>usecs/call</th>
<th>calls</th>
<th>errors</th>
<th>syscall</th>
</tr>
</thead>
<tbody>
<tr>
<td>58.43</td>
<td>0.007430</td>
<td>17</td>
<td>450</td>
<td></td>
<td>read</td>
</tr>
<tr>
<td>24.33</td>
<td>0.003094</td>
<td>4</td>
<td>850</td>
<td>210</td>
<td>access</td>
</tr>
<tr>
<td>5.53</td>
<td>0.000703</td>
<td>4</td>
<td>190</td>
<td>10</td>
<td>open</td>
</tr>
<tr>
<td>4.16</td>
<td>0.000529</td>
<td>3</td>
<td>175</td>
<td></td>
<td>write</td>
</tr>
<tr>
<td>2.97</td>
<td>0.000377</td>
<td>2</td>
<td>180</td>
<td></td>
<td>munmap</td>
</tr>
<tr>
<td>1.95</td>
<td>0.000248</td>
<td>1</td>
<td>180</td>
<td></td>
<td>close</td>
</tr>
<tr>
<td>1.01</td>
<td>0.000128</td>
<td>1</td>
<td>180</td>
<td></td>
<td>mmap</td>
</tr>
<tr>
<td>0.69</td>
<td>0.000088</td>
<td>18</td>
<td>5</td>
<td></td>
<td>fdatasync</td>
</tr>
<tr>
<td>0.61</td>
<td>0.000078</td>
<td>0</td>
<td>180</td>
<td></td>
<td>fstat</td>
</tr>
<tr>
<td>0.13</td>
<td>0.000017</td>
<td>3</td>
<td>5</td>
<td></td>
<td>pause</td>
</tr>
</tbody>
</table>

100.00  0.012715  2415  225 total

Shares to rate importance:

- a lot, slow or failing calls?

Name (see man pages)
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]
- Package: RHEL: ltrace SLES: ltrace

- 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 timestamp ( -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

<table>
<thead>
<tr>
<th>% time</th>
<th>seconds</th>
<th>usecs/call</th>
<th>calls</th>
<th>function</th>
</tr>
</thead>
<tbody>
<tr>
<td>98.33</td>
<td>46.76</td>
<td>5845707</td>
<td>8</td>
<td>pause</td>
</tr>
<tr>
<td>0.94</td>
<td>0.45</td>
<td>10</td>
<td>42669</td>
<td>strncmp</td>
</tr>
<tr>
<td>0.44</td>
<td>0.21</td>
<td>25</td>
<td>8253</td>
<td>fgets</td>
</tr>
<tr>
<td>0.08</td>
<td>0.04</td>
<td>11</td>
<td>3168</td>
<td>__isoc99_sscanf</td>
</tr>
<tr>
<td>0.07</td>
<td>0.03</td>
<td>20</td>
<td>1530</td>
<td>access</td>
</tr>
<tr>
<td>0.04</td>
<td>0.02</td>
<td>10</td>
<td>1611</td>
<td>strchr</td>
</tr>
<tr>
<td>0.03</td>
<td>0.01</td>
<td>10</td>
<td>1530</td>
<td>snprintf</td>
</tr>
<tr>
<td>0.02</td>
<td>0.01</td>
<td>1163</td>
<td>9</td>
<td>fdatasync</td>
</tr>
<tr>
<td>0.02</td>
<td>0.01</td>
<td>27</td>
<td>324</td>
<td>fclose</td>
</tr>
<tr>
<td>0.02</td>
<td>0.00</td>
<td>21</td>
<td>342</td>
<td>fopen</td>
</tr>
<tr>
<td>0.01</td>
<td>0.00</td>
<td>19</td>
<td>315</td>
<td>write</td>
</tr>
<tr>
<td>0.00</td>
<td>0.00</td>
<td>10</td>
<td>54</td>
<td>strncpy</td>
</tr>
</tbody>
</table>

|     |         |            |       |                  |
| 100.00 | 47.56 | 59948     | total |

- **shares to rate importance**
- **a lot or slow calls?**
- **name (see man pages)**
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 timestamp
- 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\0\0\17\0\0\0\0\0\0\0"..., 680) = 680 <0.000007>
0.000027 write(3, "\0\0\0\0\0\0\0\0\0\17\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}, 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, 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>
ls/of

- Characteristics: list of open files plus extra details
- Objective: which process accesses which file in which mode
- Usage: lsof +fg
- Package: RHEL: lsof SLES: ls/of

- Shows
  - List of files including sockets, directories, pipes
  - User, Command, Pid, Size, Device
  - File Type and File Flags

- Hints
  - +fg reports file flags which can provide a good cross check opportunity
# lsof - example

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>PID</th>
<th>TID</th>
<th>USER</th>
<th>FD</th>
<th>TYPE</th>
<th>FILE-FLAG</th>
<th>DEVICE</th>
<th>SIZE/OFF</th>
<th>NODE</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>crond</td>
<td>16129</td>
<td></td>
<td>root</td>
<td>mem</td>
<td>REG</td>
<td></td>
<td>94,1</td>
<td>165000</td>
<td>881893</td>
<td>/usr/lib64/ld-2.16.so</td>
</tr>
<tr>
<td>crond</td>
<td>16129</td>
<td></td>
<td>root</td>
<td>0r</td>
<td>CHR</td>
<td>LG</td>
<td>1,3</td>
<td>0t0</td>
<td>2051</td>
<td>/dev/null</td>
</tr>
<tr>
<td>crond</td>
<td>16129</td>
<td></td>
<td>root</td>
<td>1u</td>
<td>unix</td>
<td>RW 0x00000001f1ba02000</td>
<td>0t0</td>
<td>106645</td>
<td>socket</td>
<td></td>
</tr>
<tr>
<td>crond</td>
<td>16129</td>
<td></td>
<td>root</td>
<td>2u</td>
<td>unix</td>
<td>RW 0x00000001f1ba02000</td>
<td>0t0</td>
<td>106645</td>
<td>socket</td>
<td></td>
</tr>
<tr>
<td>crond</td>
<td>16129</td>
<td></td>
<td>root</td>
<td>4r</td>
<td>a_inode</td>
<td>0x80000</td>
<td>0,9</td>
<td>0</td>
<td>6675</td>
<td>notify</td>
</tr>
<tr>
<td>crond</td>
<td>16129</td>
<td></td>
<td>root</td>
<td>5u</td>
<td>unix</td>
<td>RW,0x80000 0x00000001f5d3ad000</td>
<td>0t0</td>
<td>68545</td>
<td>socket</td>
<td></td>
</tr>
<tr>
<td>dd</td>
<td>17617</td>
<td></td>
<td>root</td>
<td>cwd</td>
<td>DIR</td>
<td></td>
<td>94,1</td>
<td>4096</td>
<td>16321</td>
<td>/root</td>
</tr>
<tr>
<td>dd</td>
<td>17617</td>
<td></td>
<td>root</td>
<td>rtd</td>
<td>DIR</td>
<td></td>
<td>94,1</td>
<td>4096</td>
<td>2</td>
<td>/</td>
</tr>
<tr>
<td>dd</td>
<td>17617</td>
<td></td>
<td>root</td>
<td>txt</td>
<td>REG</td>
<td></td>
<td>94,1</td>
<td>70568</td>
<td>1053994</td>
<td>/usr/bin/dd</td>
</tr>
<tr>
<td>dd</td>
<td>17617</td>
<td></td>
<td>root</td>
<td>mem</td>
<td>REG</td>
<td></td>
<td>94,1</td>
<td>165000</td>
<td>881893</td>
<td>/usr/lib64/ld-2.16.so</td>
</tr>
<tr>
<td>dd</td>
<td>17617</td>
<td></td>
<td>root</td>
<td>0r</td>
<td>CHR</td>
<td>LG</td>
<td>1,9</td>
<td>0t0</td>
<td>2055</td>
<td>/dev/urandom</td>
</tr>
<tr>
<td>dd</td>
<td>17617</td>
<td></td>
<td>root</td>
<td>1w</td>
<td>REG</td>
<td>W,DIR,LG</td>
<td>94,1</td>
<td>5103616</td>
<td>16423</td>
<td>/root/test</td>
</tr>
<tr>
<td>dd</td>
<td>17617</td>
<td></td>
<td>root</td>
<td>2u</td>
<td>CHR</td>
<td>RW,LG</td>
<td>136,2</td>
<td>0t0</td>
<td>5</td>
<td>/dev/pts/2</td>
</tr>
</tbody>
</table>

- You can filter that per application or per file
  - Fd holds fdnumber, type, characteristic and lock information
    - File descriptors can help to read strace/ltrace output
  - Flags can be good to confirm e.g. direct IO, async IO
  - Size (e.g. mem) or offset (fds), name, ...
lsluns

- Characteristics: overview of multipathing
- Objective: check your multipath setup hierarchy
- Usage: “lsluns -a”
- Package: RHEL: s390utils-base SLES: s390-tools

```
lsluns -a
adapter = 0.0.1700
  port = 0x500507630900c7c1
  lun = 0x4020402100000000 /dev/sg0 Disk IBM:2107900
  lun = 0x4020402200000000 /dev/sg1 Disk IBM:2107900
  lun = 0x4020402300000000 /dev/sg2 Disk IBM:2107900
  lun = 0x4021402100000000 /dev/sg3 Disk IBM:2107900
  lun = 0x4021402200000000 /dev/sg4 Disk IBM:2107900
  lun = 0x4021402300000000 /dev/sg5 Disk IBM:2107900
adapter = 0.0.1780
  port = 0x500507630903c7c1
  lun = 0x4020402100000000 /dev/sg17 Disk IBM:2107900
  lun = 0x4020402200000000 /dev/sg18 Disk IBM:2107900
  lun = 0x4020402300000000 /dev/sg19 Disk IBM:2107900
  lun = 0x4021402100000000 /dev/sg20 Disk IBM:2107900
  lun = 0x4021402200000000 /dev/sg23 Disk IBM:2107900
  lun = 0x4021402300000000 /dev/sg26 Disk IBM:2107900
```

- Lsluns provides a hierarchical view which often easily identifies missing paths, adapters or similar imbalances
- Adapter to WWPN associations can have concurring targets
  - Low overhead, max fallback capability, best performance, ...
Multipath -ll

- Characteristics: overview of multipathing
- Objective: check your multipath setup configuration
- Usage: “multipath -ll”
- Package: RHEL: device-mapper-multipath SLES: multipath-tools

```
multipath -ll
swap-3of6  (36005076309ffc7c10000000000002022) dm-2 IBM ,2107900
size=256G features='0' hwhandler='0' wp=rw
  `+- policy='service-time 0' prio=0 status=active
      |- 0:0:20:1075986464 sdb  8:16  active ready running
      |- 1:0:22:1075986464 sdx  65:112 active ready running
      |- 2:0:21:1075986464 sdh  8:112 active ready running
      |- 3:0:20:1075986464 sdn  8:208 active ready running
      |- 4:0:26:1075986464 sdz  65:144 active ready running
      |- 5:0:19:1075986464 sdy  65:128 active ready running
      | `- 7:0:25:1075986464 sdac 65:192 active ready running
         `- 6:0:24:1075986464 sdad 65:208 active ready running
[...]
```

- This also reports multipath.conf inconsistencies
- Check all reported parameters are what you thought them to be
  - For example (in)famous rr_min_io renaming
**Hyptop**

- **Characteristics:** Easy to use Guest/LPAR overview
- **Objective:** Check CPU and overhead statistics of your and sibling images
- **Usage:** `hyptop`
- **Package:** RHEL: s390utils-base SLES: s390-tools

- **Shows**
  - CPU load & Management overhead
  - Memory usage (only under zVM)
  - 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” checkbox in HMC
Hyptop

Why are exactly 4 CPUs used in all 6 CPU guests

All these do not fully utilize their 2 CPUs

No peaks in service guests

LPAR images would see other LPARs

memuse = resident

service guest weights
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]
- Short: dstat -vtin
- Package: RHEL: dstat SLES: n/a WWW: http://dag.wieers.com/home-made/dstat/
- Shows
  - Throughput
  - Utilization
  - Summarized and per Device queue information
  - Much more … it more or less combines several classic tools like iostat and vmstat
- Hints
  - Powerful plug-in 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

similar to vmstat
similar to iostat
(also per device)
new in live tool
HTOP

- Characteristics: Process overview with extra features
- Objective: Get a understanding about your running processes
- Usage: `htop`
- Package: RHEL: n/a  SLES: n/a  WWW: http://htop.sourceforge.net/
- 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

Common process info

Hierarchy

Accumulated Usage and IO rates

Tasks: 101, 80 thr; 60 running
Load average: 42.03 16.67 6.24
Uptime: 00:17:11
netstat

- Characteristics: Easy to use, connection information
- Objective: Lists connections
- Usage: `netstat -eeapn`
- Package: RHEL: net-tools SLES: net-tools

- Shows
  - Information about each connection
  - Various connection states

- Hints
  - Inodes and program names are useful to reverse-map ports to applications
**netstat -s**

- **Characteristics:** Easy to use, very detailed information
- **Objective:** Display summary statistics for each protocol
- **Usage:** `netstat -s`

- **Shows**
  - Information to each protocol
  - Amount of incoming and outgoing packages
  - Various error states, for example TCP segments retransmitted!

- **Hints**
  - Shows accumulated values since system start, therefore mostly the differences between two snapshots are needed
  - There is always a low amount of packets in error or resets
  - Retransmits occurring only when the system is sending data
    - When the system is not able to receive, then the sender shows retransmits
  - Use sadc/sar to identify the device
netstat -s

**Output sample:**

**Tcp:**
- 15813 active connections openings
- 35547 passive connection openings
- 305 failed connection attempts
- 0 connection resets received
- 6117 connections established
- 8160642 segments received
- 127803327 segments send out
- **288729 segments retransmitted**
- 0 bad segments received.
- 6 resets sent
Socket statistics

- Characteristics: Information on socket level
- Objective: Check socket options and weird connection states
- Usage: `ss -aempi`
- Package: RHEL: iproute-2 SLES: iproute2
- Shows
  - Socket options
  - Socket receive and send queues
  - Inode, socket identifiers

Sample output

```
ss -aempi
State    Recv-Q Send-Q Local Address:Port Peer Address:Port
LISTEN   0      128 :::ssh :::::
users:(("sshd",959,4)) ino:7851 sk:ef858000 mem:(r0,w0,f0,t0)
```

Hints

- Inode numbers can assist reading strace logs
- Check long outstanding queue elements
IPTRAF

- Characteristics: Live information on network devices / connections
- Objective: Filter and format network statistics
- Usage: iptraf
- Package: RHEL: iptraf / iptraf-ng SLES: 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
  - There are various other tools: iftop, bmon, …
    - like with net benchmarks no one seem to fit all
iptraf

- Questions that usually can be addressed
  - Connection behavior overview
  - Do you have peaks in your workload characteristic
  - Who does your host really communicate with

- Comparison to wireshark
  - Not as powerful, but much easier and faster to use
  - Lower overhead and no sniffing needed (often prohibited)
End of Part III
# Agenda

## Basic
- Utilization
- Scheduling
- Page Cache
- Swapping
- top
- ps
- vmstat

## Intermediate
- General thoughts
- Sysstat
- Dasdstat
- Scsi I/O statistics
- iotop
- Lszcrpt
- Icastats
- Lsqeth
- Ethtool
- Preparation

## Advanced
- Strace
- Ltrace
- Lsof
- Lsluns
- Multipath
- Hytop
- Dstat
- Htop
- Netstat
- Socket Statistics
- Iptraf

## Master
- Perf
- Slabtop
- Blktrace
- Ziomon
- Tcpdump
- Java Health Center
- Java Garbage Collection and Memory visualizer
- Jinsight

## Elite
- Cachestat
- Smem
- Valgrind
- Irqstats
- Wireshark
- Kernel Tracepoints
- Systemtap
Perf

- Characteristics: Easy to use profiling and kernel tracing
- Objective: Get detailed information where & why CPU is consumed
- Usage: `perf` (to begin with)
- Package: RHEL: perf SLES: perf

- Shows
  - Sampling for CPU hotspots
    - Annotated source code along hotspots
  - 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 upstream, wait for next distribution releases
Perf

- What profiling can and what it can't
  + Search hotspots 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", …
    • Other than real “sampling” this can help to search for stalls
  - Counters provide even lower overhead and report HW and Software events
Perf profiling

- Perf example how-to
  - Needs proper HW support to work well for the kernel (not yet in the field)
    - Ignore and kernel profiling data until this is available!
  - 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
  - [...]

March 3, 2015

Linux-Performance-know it all series
Perf profiling

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

<table>
<thead>
<tr>
<th># Baseline</th>
<th>Delta</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12.14%</td>
<td>+8.07%</td>
<td>[kernel.kallsyms] [k] lock_acquire</td>
</tr>
<tr>
<td>8.96%</td>
<td>+5.50%</td>
<td>[kernel.kallsyms] [k] lock_release</td>
</tr>
<tr>
<td>4.83%</td>
<td>+0.38%</td>
<td>reaim [.] add_long</td>
</tr>
<tr>
<td>4.22%</td>
<td>+0.41%</td>
<td>reaim [.] add_int</td>
</tr>
<tr>
<td>4.10%</td>
<td>+2.49%</td>
<td>[kernel.kallsyms] [k] lock_acquired</td>
</tr>
<tr>
<td>3.17%</td>
<td>+0.38%</td>
<td>libc-2.11.3.so [.] msort_with_tmp</td>
</tr>
<tr>
<td>3.56%</td>
<td>-0.37%</td>
<td>reaim [.] string_rtns_1</td>
</tr>
<tr>
<td>3.04%</td>
<td>-0.38%</td>
<td>libc-2.11.3.so [.] strncat</td>
</tr>
</tbody>
</table>
**Perf stat - preparation**

- **Activate the cpu measurement facility**
  - If not you'll encounter this
    ```
    Error: You may not have permission to collect stats.
    Consider tweaking /proc/sys/kernel/perf_event_paranoid
    Fatal: Not all events could be opened.
    ```

- **Check if its activated**
  - separate for counter and sampling
  - Basic and/or Diagnostic mode
    ```
    lscpumf -i
    CPU-measurement counter facility
    [...] Sampling facility information for cpum_sf
    [...] Authorized sampling modes:
    basic (sample size: 32 bytes)
    diagnostic (sample size: 85 bytes)
    ```
Perf stat - usage

perf stat -B --event=cycles,instructions,r20,r21,r3,r5,sched:sched_wakeup find / -iname "*foobar*"

Performance counter stats for 'find / -iname *foobar*':
3,623,031,935 cycles # 0.000 GHz
1,515,404,340 instructions # 0.42 insns per cycle
1,446,545,776 PROBLEM_STATE_CPU_CYCLES
  757,589,098 PROBLEM_STATE_INSTRUCTIONS
  705,740,759 L1I_PENALTY_CYCLES
  576,226,424 L1D_PENALTY_CYCLES
  40,675 sched:sched_wakeup
6.156288957 seconds time elapsed

- Events
  - Cycles/Instructions globally
  - Note: counters are now readable, but aliases can still be used
    - e.g. r20 = PROBLEM_STATE_CPU_CYCLES
    - List of all existing events lscpumf -C
    - counters available to you lscpumf -c
  - Not only HW events, you can use any of the currently 163 tracepoints
Slabtop

- Characteristics: live profiling of kernel memory pools
- Objective: Analyze kernel memory consumption
- Usage: `slabtop`
- Package: RHEL: procps SLES: procps

- Shows
  - Active / Total object number/size
  - Objects per Slab
  - Object Name and Size
  - Objects per Slab

- Hints
  - -o is one time output e.g. to gather debug data
  - Despite slab/slob/slub in kernel its always slabtop
Slabtop - example

<table>
<thead>
<tr>
<th>OBJS</th>
<th>ACTIVE</th>
<th>USE</th>
<th>OBJ SIZE</th>
<th>SLABS</th>
<th>OBJ/SLAB</th>
<th>CACHE SIZE</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>578172</td>
<td>578172</td>
<td>100%</td>
<td>0.19K</td>
<td>13766</td>
<td>42</td>
<td>110128K dentry</td>
<td></td>
</tr>
<tr>
<td>458316</td>
<td>458316</td>
<td>100%</td>
<td>0.11K</td>
<td>12731</td>
<td>36</td>
<td>509244K sysfs_dir_cache</td>
<td></td>
</tr>
<tr>
<td>368784</td>
<td>368784</td>
<td>100%</td>
<td>0.61K</td>
<td>7092</td>
<td>52</td>
<td>226944K proc_inode_cache</td>
<td></td>
</tr>
<tr>
<td>113685</td>
<td>113685</td>
<td>100%</td>
<td>0.10K</td>
<td>2915</td>
<td>39</td>
<td>11660K buffer_head</td>
<td></td>
</tr>
<tr>
<td>113448</td>
<td>113448</td>
<td>100%</td>
<td>0.55K</td>
<td>1956</td>
<td>58</td>
<td>62592K inode_cache</td>
<td></td>
</tr>
<tr>
<td>111872</td>
<td>44251</td>
<td>39%</td>
<td>0.06K</td>
<td>1748</td>
<td>64</td>
<td>6992K kmalloc-64</td>
<td></td>
</tr>
<tr>
<td>54688</td>
<td>50382</td>
<td>92%</td>
<td>0.25K</td>
<td>1709</td>
<td>32</td>
<td>13672K kmalloc-256</td>
<td></td>
</tr>
<tr>
<td>40272</td>
<td>40239</td>
<td>99%</td>
<td>4.00K</td>
<td>5034</td>
<td>8</td>
<td>161088K kmalloc-4096</td>
<td></td>
</tr>
<tr>
<td>39882</td>
<td>39882</td>
<td>100%</td>
<td>0.04K</td>
<td>391</td>
<td>102</td>
<td>1564K ksm_stable_node</td>
<td></td>
</tr>
<tr>
<td>38505</td>
<td>36966</td>
<td>96%</td>
<td>0.62K</td>
<td>755</td>
<td>51</td>
<td>24160K shmem_inode_cache</td>
<td></td>
</tr>
<tr>
<td>37674</td>
<td>37674</td>
<td>100%</td>
<td>0.41K</td>
<td>966</td>
<td>39</td>
<td>15456K dm_rq_target_io</td>
<td></td>
</tr>
</tbody>
</table>

- How is kernel memory managed by the slab allocator used
  - Named memory pools or Generic kmalloc pools
  - Active/total objects and their size
  - Growth/shrinks of caches due to workload adaption
BLKTRACE

- Characteristics: High detail info of the block device layer actions
- Objective: Understand what's going with your I/O in the kernel and devices
- Usage: blktrace -d [device(s)]
- Then: blkparse -st [commontracefilepart]
- Package: RHEL: blktrace SLES: blktrace
- 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?

- Blocktrace 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
The “blktrace” tool – block device events (simplified)

- **App / A / X**: Application
- **Q**: Queue
- **mergeable**: Mergeable request
- **M / F**: Merge
- **N**: Not mergeable
- **Y**: Merge with an existing request

**Actions**:

- **G**: Generate a new request
- **I**: Inserted
- **P**: Plug
- **Q**: Queued
- **U**: Unplug
- **D**: Dispatch
- **C**: Complete

**States**:

- **B**: Bounced
- **F**: Front merge
- **G**: Get request
- **M**: Back merge
- **P**: Plug queue and wait for follow-on request
- **T**: Timer unplug
- **U**: Unplug on upper limit, time reached or task switch
- **X**: Split

**Additional Info**

- **A**: Remap
- **C**: Complete
- **D**: Issued
- **F**: Front merge
- **G**: Get request
- **I**: Inserted
- **M**: Back merge
- **P**: Plug
- **Q**: Queued
- **T**: Timer unplug
- **U**: Unplug
- **X**: Split

Additional info from dasdstat or scsi sysfs statistics required to get the complete picture.
LKtrace

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
## blktrace

### 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 I/O
- The summary below shows the difference after a fix

<table>
<thead>
<tr>
<th></th>
<th>Total initially</th>
<th></th>
<th>Total after Fix</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reads Queued:</td>
<td>560,888, 2,243MiB</td>
<td>Writes Queued: 226,242, 904,968KiB</td>
<td></td>
</tr>
<tr>
<td>Read Dispatches:</td>
<td>544,701, 2,243MiB</td>
<td>Write Dispatches: 159,318, 904,968KiB</td>
<td></td>
</tr>
<tr>
<td>Reads Requeued:</td>
<td>0</td>
<td>Writes Requeued: 0</td>
<td></td>
</tr>
<tr>
<td>Reads Completed:</td>
<td>544,716, 2,243MiB</td>
<td>Writes Completed: 159,321, 904,980KiB</td>
<td></td>
</tr>
<tr>
<td>Read Merges:</td>
<td>16,187, 64,748KiB</td>
<td>Write Merges: 61,744, 246,976KiB</td>
<td></td>
</tr>
<tr>
<td>IO unplugs:</td>
<td>149,614</td>
<td>Timer unplugs: 2,940</td>
<td></td>
</tr>
</tbody>
</table>

|                      | 734,315, 2,937MiB |                      | 300,188, 1,200MiB |
|                      | 214,972, 2,937MiB |                      | 215,176, 1,200MiB |
|                      | 0               |                      | 0                    |
|                      | 214,971, 2,937MiB |                      | 215,177, 1,200MiB   |
|                      | 519,343, 2,077MiB |                      | 73,325, 293,300KiB  |
|                      | 337,130         |                      | 11,184               |
ziomon

- Characteristics: in depth zfcp based I/O analysis
- Objective: Analyze your FCP based I/O
- Usage: “ziomon” → “ziorep*”
- Package: RHEL: s390utils(-ziomon) SLES: s390-tools

- Be aware that ziomon can be memory greedy if you have very memory constrained systems
- The has many extra functions please check out the live virtual class of Stephan Raspl
  - Replay: http://ibmstg.adobeconnect.com/p7zvdjz0yye/
TCPDump

- Characteristics: dumps network traffic to console/file
- Objective: analyze packets of applications manually
- Usage: “tcpdump ...
- Package: RHEL: tcpdump SLES: tcpdump

```
tcpdump host pserver1

tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
[...]
^C
31 packets captured
31 packets received by filter
0 packets dropped by kernel
```

- Not all devices support dumping packets in older distribution releases
  - Also often no promiscuous mode
- Check flags or even content if your expectations are met
- -w flag exports captured unparsed data to a file for later analysis in libpcap format
  - Also supported by wireshark
- Usually you have to know what you want to look for
Java Performance in general

- “Too” many choices
  - There are many Java performance tools out there

- Be aware of common Java myths often clouding perception

- Differences
  - Profiling a JVM might hide the Java methods
  - Memory allocation of the JVM isn't the allocation of the Application
Java - Health Center

- Characteristics: Lightweight Java Virtual Machine Overview
- Objective: Find out where memory is leaked, sub-optimally cached, ...
- Usage: IBM Support Assistant (Eclipse)
- Package: RHEL: n/a SLES: n/a WWW: ibm.com/developerworks/java/jdk/tools/healthcenter
  Java Agents integrated V5SR10+, V6SR3+, usually no target install required

- Shows
  - Memory usage
  - Method Profiling
  - I/O Statistics
  - Class loading
  - Locking

- Hints
  - Low overhead, therefore even suitable for monitoring
  - Agent activation -Xhealthcenter:port=12345
  - Can trigger dumps or verbosegc for in-depth memory analysis
Example of method profiling

### Health Center - example

#### Method profile - JVM Support Assistant Workspace

**Support Assistant**

- **Status**
- **Classes**
- **Environment**
- **Garbage Collection**
- **Locking**
- **Memory**
- **Profiling**

#### Connection

- 1921687211097
- 54 MB received. Last updated 13:04:32. Some data was dropped because it was produced faster than the client could consume it. Around 9% of the data was lost.

#### Analysis and Recommendations

- The method `com.ibm.tmcc.demo.ComputingResourcesConsumer.generateCpuLoad()` is consuming approximately 96% of the CPU cycles. It may be a good candidate for optimization.

### Method profiling table

<table>
<thead>
<tr>
<th>Method</th>
<th>Self (%)</th>
<th>Total (%)</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>com.ibm.tmcc.demo.ComputingResourcesConsumer.generateCpuLoad()</code></td>
<td>95.8</td>
<td>95.8</td>
<td><code>org.apache.xml.dtm.ref.dom2tm.DOM2DTM.addNode@95.8</code></td>
</tr>
</tbody>
</table>

#### Invocation paths

- `com.ibm.cds.CDSHandleFileENTRY` (100%)
- `TMCCDemoServlet.doGet` (100%)
- `TMCCDemoServlet.handleHttpRequest` (100%)
- `ComputingResourcesConsumer.generateCpuLoad()` (100%)

---

**Linux on System z Performance Evaluation**

© 2015 IBM Corporation
Java - Garbage Collection and Memory Visualizer

- Characteristics: in-depth Garbage Collection analysis
- Objective: Analyze JVM memory management
- Usage: IBM Support Assistant (Eclipse)
- Package: RHEL: n/a SLES: n/a WWW: ibm.com/developerworks/java/jdk/tools/gcmv
  reads common verbosegc output, so usually no target install required

- Shows
  - Memory usage
  - Garbage Collection activities
  - Pauses
  - Memory Leaks by stale references

- Hints
  - GCMV can also compare output of two runs
  - Activate verbose logs -verbose:gc -Xverbosegclog:<log_file>
Garbage Collection and Memory Visualizer

- Most important values / indicators are:
  - Proportion of time spent in gc pauses (should be less than 5%)
  - For gencon: global collections << minor collections
Java - Jinsight

- Characteristics: zoomable call stack
- Objective: Analyze method call frequency and duration
- Usage: `jinsight_trace -tracemethods <yourProgram> <yourProgramArgs>`
- Package: RHEL: n/a SLES: n/a WWW: IBM alphaworks

- Shows
  - Call Stack and time

- Hints
  - Significant slowdown, not applicable to production systems
  - No more maintained, but so far still working
Jinsight Execution View

- Threads in columns, select one to zoom in
Many horizontal stages mean deep call stacks
Long vertical areas mean long method execution
Rectangles full of horizontal lines can be an issue
End of Part IV
## Agenda

<table>
<thead>
<tr>
<th>Basic</th>
<th>Intermediate</th>
<th>Advanced</th>
<th>Master</th>
<th>Elite</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Utilization</td>
<td>- General thoughts</td>
<td>- Strace</td>
<td>- Perf</td>
<td>- Cachestat</td>
</tr>
<tr>
<td>- Scheduling</td>
<td>- Sysstat</td>
<td>- Ltrace</td>
<td>- slabtop</td>
<td>- Smem</td>
</tr>
<tr>
<td>- Page Cache</td>
<td>- Dasdstat</td>
<td>- Lsof</td>
<td>- Blktrace</td>
<td>- Valgrind</td>
</tr>
<tr>
<td>- Swapping</td>
<td>- Scsi I/O statistics</td>
<td>- Lsluns</td>
<td>- Ziomon</td>
<td>- Irqstats</td>
</tr>
<tr>
<td>- top</td>
<td>- iotop</td>
<td>- Multipath</td>
<td>- Tcpdump</td>
<td>- Wireshark</td>
</tr>
<tr>
<td>- ps</td>
<td>- Lszcrpt</td>
<td>- hytop</td>
<td>- Java Health Center</td>
<td>- Kernel</td>
</tr>
<tr>
<td>- vmstat</td>
<td>- icastats</td>
<td>- Dstat</td>
<td>- Java Garbage Collection and</td>
<td>- Tracepoints</td>
</tr>
<tr>
<td></td>
<td>- Lsqeth</td>
<td>- Htop</td>
<td>Memory visualizer</td>
<td>- Systemtap</td>
</tr>
<tr>
<td></td>
<td>- Ethtool</td>
<td>- Netstat</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Preparation</td>
<td>- Socket Statistics</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Iptraf</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Cachestat

- Characteristics: Simple per page views of caching
- Objective: Detect what parts of a file are in page cache
- Usage: Write – or search for example code
- Package: n/a (pure code around the mincore system call)

- Shows
  - How much of a file is in cache

- Hints
  - We are now going from unsupported to non existent packages
  - Still the insight can be so useful, it is good to know
Cachestat usage

```bash
./cachestat -v ../Music/mysong.flac
```

pages in cache: 445/12626 (3.5%) [filesize=50501.0K, pagesize=4K]

cache map:

```
0: |x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|
32: |x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|
64: |x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|
[...]
320: |x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|
352: |x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|
384: |x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|
416: |x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|
448: |
480: |
[...]
12576: |
12608: |
```

- Here I show how much of a file is in cache while playing a song
  - You'll see readahead here
  - You'll also see the last block is almost always read in this case
sme

- Characteristics: Memory usage details per process/mapping
- Objective: Where is userspace memory really used
- Usage: smem -tk -c "pid user command swap vss uss pss rss"
- smem -m -tk -c "map count pids swap vss uss rss pss avgrss avgpss"

- Package: RHEL: n/a SLES: n/a WWW http://www.selenic.com/smem/
- Shows
  - Pid, user, Command or Mapping, Count, Pid
  - Memory usage in categories vss, uss, rss, pss and swap

- Hints
  - Has visual output (pie charts) and filtering options as well
  - No support for huge pages or transparent huge pages (kernel interface missing)
smem – process overview

```
smem -tk -c "pid user command swap vss uss pss rss"
```

<table>
<thead>
<tr>
<th>PID</th>
<th>User</th>
<th>Command</th>
<th>Swap</th>
<th>VSS</th>
<th>USS</th>
<th>PSS</th>
<th>RSS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1860</td>
<td>root</td>
<td>/sbin/agetty -s sclp_line0</td>
<td>0</td>
<td>2.1M</td>
<td>92.0K</td>
<td>143.0K</td>
<td>656.0K</td>
</tr>
<tr>
<td>1861</td>
<td>root</td>
<td>/sbin/agetty -s ttysclp0 11</td>
<td>0</td>
<td>2.1M</td>
<td>92.0K</td>
<td>143.0K</td>
<td>656.0K</td>
</tr>
<tr>
<td>493</td>
<td>root</td>
<td>/usr/sbin/atd -f</td>
<td>0</td>
<td>2.5M</td>
<td>172.0K</td>
<td>235.0K</td>
<td>912.0K</td>
</tr>
<tr>
<td>1882</td>
<td>root</td>
<td>/sbin/udevd</td>
<td>0</td>
<td>2.8M</td>
<td>128.0K</td>
<td>267.0K</td>
<td>764.0K</td>
</tr>
<tr>
<td>1843</td>
<td>root</td>
<td>/usr/sbin/crond -n</td>
<td>0</td>
<td>3.4M</td>
<td>628.0K</td>
<td>693.0K</td>
<td>1.4M</td>
</tr>
<tr>
<td>514</td>
<td>root</td>
<td>/bin/dbus-daemon --system -</td>
<td>0</td>
<td>3.2M</td>
<td>700.0K</td>
<td>771.0K</td>
<td>1.5M</td>
</tr>
<tr>
<td>524</td>
<td>root</td>
<td>/sbin/rsyslogd -n -c 5</td>
<td>0</td>
<td>219.7M</td>
<td>992.0K</td>
<td>1.1M</td>
<td>1.9M</td>
</tr>
<tr>
<td>2171</td>
<td>root</td>
<td>./hhptest</td>
<td>0</td>
<td>5.7G</td>
<td>1.0M</td>
<td>1.2M</td>
<td>3.2M</td>
</tr>
<tr>
<td>1906</td>
<td>root</td>
<td>-bash</td>
<td>0</td>
<td>103.8M</td>
<td>1.4M</td>
<td>1.5M</td>
<td>2.1M</td>
</tr>
<tr>
<td>2196</td>
<td>root</td>
<td>./hhptest</td>
<td>0</td>
<td>6.2G</td>
<td>2.0M</td>
<td>2.2M</td>
<td>3.9M</td>
</tr>
<tr>
<td>1884</td>
<td>root</td>
<td>sshd: root@pts/0</td>
<td>0</td>
<td>13.4M</td>
<td>1.4M</td>
<td>2.4M</td>
<td>4.2M</td>
</tr>
<tr>
<td>2203</td>
<td>root</td>
<td>/usr/bin/python /usr/bin/sm</td>
<td>0</td>
<td>109.5M</td>
<td>6.1M</td>
<td>6.2M</td>
<td>6.9M</td>
</tr>
</tbody>
</table>

- **How much of a process is:**
  - Swap - Swapped out
  - VSS - Virtually allocated
  - USS - Really unique
  - RSS - Resident
  - PSS - Resident accounting a proportional part of shared memory
sMEM – mAPPings overview

sMEM -m -tk -c "map count pids swap vss uss rss pss avgrss avgpss"

<table>
<thead>
<tr>
<th>Map</th>
<th>Count</th>
<th>PIDs</th>
<th>Swap</th>
<th>VSS</th>
<th>USS</th>
<th>RSS</th>
<th>PSS</th>
<th>AVGRSS</th>
<th>AVGPPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>[stack:531]</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>8.0M</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>[vdso]</td>
<td>25</td>
<td>25</td>
<td>0</td>
<td>200.0K</td>
<td>0</td>
<td>132.0K</td>
<td>0</td>
<td>5.0K</td>
<td>0</td>
</tr>
<tr>
<td>/dev/zero</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>2.5M</td>
<td>4.0K</td>
<td>4.0K</td>
<td>4.0K</td>
<td>4.0K</td>
<td>4.0K</td>
</tr>
<tr>
<td>/usr/lib64/sasl2/libsasldb.so.2.0.23</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>28.0K</td>
<td>4.0K</td>
<td>4.0K</td>
<td>4.0K</td>
<td>4.0K</td>
<td>4.0K</td>
</tr>
<tr>
<td>/bin/dbus-daemon</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>404.0K</td>
<td>324.0K</td>
<td>324.0K</td>
<td>324.0K</td>
<td>324.0K</td>
<td>324.0K</td>
</tr>
<tr>
<td>/usr/sbin/sshd</td>
<td>6</td>
<td>2</td>
<td>0</td>
<td>1.2M</td>
<td>248.0K</td>
<td>728.0K</td>
<td>488.0K</td>
<td>364.0K</td>
<td>244.0K</td>
</tr>
<tr>
<td>/bin/systemd</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>768.0K</td>
<td>564.0K</td>
<td>564.0K</td>
<td>564.0K</td>
<td>564.0K</td>
<td>564.0K</td>
</tr>
<tr>
<td>/bin/bash</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1.0M</td>
<td>792.0K</td>
<td>792.0K</td>
<td>792.0K</td>
<td>792.0K</td>
<td>792.0K</td>
</tr>
<tr>
<td>[stack]</td>
<td>25</td>
<td>25</td>
<td>0</td>
<td>4.1M</td>
<td>908.0K</td>
<td>976.0K</td>
<td>918.0K</td>
<td>39.0K</td>
<td>36.0K</td>
</tr>
<tr>
<td>/lib64/libc-2.14.1.so</td>
<td>75</td>
<td>25</td>
<td>0</td>
<td>40.8M</td>
<td>440.0K</td>
<td>9.3M</td>
<td>1.2M</td>
<td>382.0K</td>
<td>48.0K</td>
</tr>
<tr>
<td>/lib64/libcrypto.so.1.0.0j</td>
<td>8</td>
<td>4</td>
<td>0</td>
<td>7.0M</td>
<td>572.0K</td>
<td>2.0M</td>
<td>1.3M</td>
<td>501.0K</td>
<td>321.0K</td>
</tr>
<tr>
<td>[heap]</td>
<td>16</td>
<td>16</td>
<td>0</td>
<td>8.3M</td>
<td>6.4M</td>
<td>6.9M</td>
<td>6.6M</td>
<td>444.0K</td>
<td>422.0K</td>
</tr>
<tr>
<td>&lt;anonymous&gt;</td>
<td>241</td>
<td>25</td>
<td>0</td>
<td>55.7G</td>
<td>20.6M</td>
<td>36.2M</td>
<td>22.3M</td>
<td>1.4M</td>
<td>913.0K</td>
</tr>
</tbody>
</table>

- How much of a mapping is:
  - Swap - Swapped out
  - VSS - Virtually allocated
  - USS - Really unique
  - RSS - Resident
  - PSS - Resident accounting a proportional part of shared memory
  - Averages as there can be multiple mappers
Example of a memory distribution Visualization (many options)
But before thinking of monitoring be aware that the proc/#pid/smaps interface is an expensive one
Valgrind

- Characteristics: in-depth memory analysis
- Objective: Find out where memory is leaked, sub-optimally cached, ...
- Usage: valgrind [program]
- Package: RHEL: valgrind SLES: valgrind

- 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

Diagram:
- Binary
  - Translation into IR
  - Instrumentation
  - Translation to machine code
  - New binary
  - System call wrapper
- Libraries
- Valgrind
- Kernel
- System call interface
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
  - Cache.grind: 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)
Valgrind - Good to know

- No need to recompile, but
  - Better results with debug info
  - Gcc option -O0 might result in more findings (the compiler might hide some errors)
  - Gcc option -fno-builtin might result in more findings

- --trace-children=yes will also debug child processes
- Setuid programs might cause trouble
  - Valgrind is the process container (→ no setuid)
  - Possible solution: remove setuid and start as the right user, check documentation for other ways

- The program will be slower
  - 5-30 times slower for memcheck
 IRQ Statistics

- Characteristics: Low overhead IRQ information
- Objective: Condensed overview of IRQ activity
- Usage: `cat /proc/interrupts` and `cat /proc/softirqs`
- Package: n/a (Kernel interface)

- Shows
  - Which interrupts happen on which cpu
  - Where softirqs and tasklets take place

- Hints
  - Recent Versions (SLES11-SP2) much more useful due to better naming
  - 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 focused 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

<table>
<thead>
<tr>
<th></th>
<th>CPU0</th>
<th>CPU1</th>
<th>CPU2</th>
<th>CPU3</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXT:</td>
<td>21179</td>
<td>24235</td>
<td>22217</td>
<td>22959</td>
</tr>
<tr>
<td>I/O:</td>
<td>1542959</td>
<td>340076</td>
<td>356381</td>
<td>325691</td>
</tr>
<tr>
<td>CLK:</td>
<td>15995</td>
<td>16718</td>
<td>15806</td>
<td>16531</td>
</tr>
<tr>
<td>EXC:</td>
<td>255</td>
<td>325</td>
<td>332</td>
<td>227</td>
</tr>
<tr>
<td>EMS:</td>
<td>4923</td>
<td>7129</td>
<td>6068</td>
<td>6201</td>
</tr>
<tr>
<td>TMR:</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>TAL:</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PFL:</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DSD:</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>VRT:</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SCP:</td>
<td>6</td>
<td>63</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>IUC:</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>CPM:</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>CIO:</td>
<td>163</td>
<td>310</td>
<td>269</td>
<td>213</td>
</tr>
<tr>
<td>QAI:</td>
<td>1541773</td>
<td>338857</td>
<td>354728</td>
<td>324110</td>
</tr>
<tr>
<td>DAS:</td>
<td>1023</td>
<td>909</td>
<td>1384</td>
<td>1368</td>
</tr>
</tbody>
</table>

[EXT] Clock Comparator
[EXT] External Call
[EXT] Emergency Signal
[EXT] CPU Timer
[EXT] Timing Alert
[EXT] Pseudo Page Fault
[EXT] DASD Diag
[EXT] Virtio
[EXT] Service Call
[EXT] IUCV
[EXT] CPU Measurement
[I/O] Common I/O Layer Interrupt
[I/O] QDIO Adapter Interrupt
[I/O] DASD

[...] 3215, 3270, Tape, Unit Record Devices, LCS, CLAW, CTC, AP Bus, Machine Check
**IRQ Statistics II**

- Also softirqs can be tracked which can be useful to
  - check if tasklets execute as intended
  - See if network, scheduling and I/O behave as expected

<table>
<thead>
<tr>
<th></th>
<th>CPU0</th>
<th>CPU1</th>
<th>CPU2</th>
<th>CPU3</th>
</tr>
</thead>
<tbody>
<tr>
<td>HI</td>
<td>498</td>
<td>1522</td>
<td>1268</td>
<td>1339</td>
</tr>
<tr>
<td>TIMER</td>
<td>5640</td>
<td>914</td>
<td>664</td>
<td>643</td>
</tr>
<tr>
<td>NET_TX</td>
<td>15</td>
<td>16</td>
<td>52</td>
<td>32</td>
</tr>
<tr>
<td>NET_RX</td>
<td>18</td>
<td>34</td>
<td>87</td>
<td>45</td>
</tr>
<tr>
<td>BLOCK</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>BLOCK_IOPOLL</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>TASKLET</td>
<td>13</td>
<td>10</td>
<td>44</td>
<td>20</td>
</tr>
<tr>
<td>SCHED</td>
<td>8055</td>
<td>702</td>
<td>403</td>
<td>445</td>
</tr>
<tr>
<td>HRTIMER</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>RCU</td>
<td>5028</td>
<td>2906</td>
<td>2794</td>
<td>2564</td>
</tr>
</tbody>
</table>
Wireshark

- Characteristics: Analyzes captured network traffic
- Objective: In depth analysis of handshakes, missing replies, protocols, ...
- Usage: Dump in libpcap or pcap-ng format (tcpdump, dumpcap) then analyze on remote system via “wireshark”
- Package: RHEL: wireshark SLES: wireshark

- No “direct” invocation on System z usually
  - e.g. on RH6 there is not even a wireshark binary
- Scrolling huge files on Remote X isn't fun anyway
  - Capturing tools are available

- Custom columns and profiles are important to visualize what you want to look for
- For more details you might start at
  - The share sessions of Mathias Burkhard
    https://share.confex.com/share/121/webprogram/Session13282.html
  - Official documentation http://www.wireshark.org/docs/wsug_html/
Wireshark example

1. Dump via “tcpdump -w” or wireshark’s “dumpcap”

2. analyze on remote system via “wireshark”

tcpdump host pserver1 -w traceme

tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes

^C40 packets captured
40 packets received by filter
0 packets dropped by kernel

[scp to my system]
wireshark traceme
Tracepoints (Events)

- Characteristics: Complex interface, but a vast source of information
- Objective: In kernel latency and activity insights
- Usage: Access /tracing mount point
- Package: n/a (Kernel interface)

- Shows
  - Timestamp 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 postprocessing
  - There are much more things you can handle with tracepoints check out
    Kernel Documentation/trace/tracepoint-analysis.txt (via perf stat)
    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, Userspace, Kernel or all of them
  - Workload was a simple 1 connection 1 byte←→1 byte load
  - Call “\texttt{perf list}” for a list of currently supported tracepoints

- We used the following tracepoints

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Tracepoint</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>netif_receive_skb</td>
<td>low level receive</td>
</tr>
<tr>
<td>P</td>
<td>napi_poll</td>
<td>napi work related to receive</td>
</tr>
<tr>
<td>Q</td>
<td>net_dev_queue</td>
<td>enqueue in the stack</td>
</tr>
<tr>
<td>S</td>
<td>net_dev_xmit</td>
<td>low level send</td>
</tr>
</tbody>
</table>
Tracepoints – example II/III

-(Simplified) Script

• # full versions tunes buffer sizes, checks files, ...

```bash
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

```bash
#          _=CPU#
#          /_ irqs-off
#         | /_ need-resched
#        || /_ hardirq/softirq
#       ||| |_/ preempt-depth
#      |||| / delay
# cmd    pid |||| time | caller
#  \ /   |||| \ \ /
<...>-24116 0..s. 486183281us+: net_dev_xmit: dev=eth5 skbaddr=000000075b7e3e8 len=67 rc=0
<idle>-0 0..s. 486183303us+: netif_receive_skb: dev=eth5 skbaddr=00000007ecc6e00 len=53
<idle>-0 0.Ns. 486183306us+: napi_poll: napi poll on napi struct 000000007d2479a8 fordevdevice eth
<...>-24116 0..s. 486183311us+: net_dev_queue: dev=eth5 skbaddr=000000075b7e3e8 len=67
<...>-24116 0..s. 486183317us+: net_dev_xmit: dev=eth5 skbaddr=000000075b7e3e8 len=67 rc=0
```
Tracepoints – example III/III

- **Example postprocessed**

<table>
<thead>
<tr>
<th></th>
<th>SUM</th>
<th>COUNT</th>
<th>AVERAGE</th>
<th>MIN</th>
<th>MAX</th>
<th>STD-DEV</th>
</tr>
</thead>
<tbody>
<tr>
<td>P2Q:</td>
<td>8478724</td>
<td>1572635</td>
<td>5.39</td>
<td>4</td>
<td>2140</td>
<td>7.41</td>
</tr>
<tr>
<td>Q2S:</td>
<td>12188675</td>
<td>1572638</td>
<td>7.65</td>
<td>3</td>
<td>71</td>
<td>4.89</td>
</tr>
<tr>
<td>S2R:</td>
<td>38562294</td>
<td>1572636</td>
<td>24.42</td>
<td>1</td>
<td>2158</td>
<td>9.08</td>
</tr>
<tr>
<td>R2P:</td>
<td>4197486</td>
<td>1572633</td>
<td>2.57</td>
<td>1</td>
<td>43</td>
<td>2.39</td>
</tr>
<tr>
<td>SUM:</td>
<td>63427179</td>
<td>1572635</td>
<td>40.03</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>SUM</th>
<th>COUNT</th>
<th>AVERAGE</th>
<th>MIN</th>
<th>MAX</th>
<th>STD-DEV</th>
</tr>
</thead>
<tbody>
<tr>
<td>P2Q:</td>
<td>7191885</td>
<td>1300897</td>
<td>5.53</td>
<td>4</td>
<td>171</td>
<td>1.31</td>
</tr>
<tr>
<td>Q2S:</td>
<td>10622270</td>
<td>1300897</td>
<td>8.17</td>
<td>3</td>
<td>71</td>
<td>5.99</td>
</tr>
<tr>
<td>S2R:</td>
<td>32078550</td>
<td>1300898</td>
<td>24.66</td>
<td>2</td>
<td>286</td>
<td>5.88</td>
</tr>
<tr>
<td>R2P:</td>
<td>3707814</td>
<td>1300897</td>
<td>2.85</td>
<td>1</td>
<td>265</td>
<td>2.59</td>
</tr>
<tr>
<td>SUM:</td>
<td>53600519</td>
<td>1300897</td>
<td>41.20</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Confirmed that ~all of the 1.2 ms were lost inside Linux (not in the fabric)
- Confirmed that it was not at/between specific function tracepoints
  - Eventually it was an interrupt locality issue causing bad caching
Systemtap

- Characteristics: tool to “tap” into the kernel for analysis
- Objective: analyze in kernel values or behavior that otherwise would be inaccessible or require a modification/recompile cycle
- Usage (mini example): “stap -v -e 'probe vfs.read {printf("read performed\n"); exit()}’"
- Package: RHEL: systemtap + systemtap-runtime SLES: systemtap
  Also requires kernel debuginfo and source/devel packages

- Procedural and C-like language based on two main constructs
  - Probes – “catching events”
    • On functions, syscalls or single statements via file:linenumber
  - Functions – “what to do”
    • Supports local and global variables
    • Program flow statements if, loops, …
- Tapsets provide pre written probe libraries

- Fore more check out “Using SystemTap on Linux on System z” from Mike O’Reilly
  https://share.confex.com/share/118/webprogram/Handout/Session10452/atlanta.pdf
There would be even more tools to cover ...

- Further tools - (no slides yet)
  - **Collectl** – full system monitoring
  - **Ftrace** – kernel function tracing
  - **Lttng** – complex latency tracing infrastructure (packages start to appear in Fedora 19)
  - **Nicstat, ktap, stap, ...**
End of Part V
Thanks (and complaints) go to

Christian Ehrhardt
Questions?

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

Martin Schwidefsky
Schönaicher Strasse 220
71032 Böblingen, Germany
Phone +49 (0)7031-16-2247
schwidefsky@de.ibm.com