Linux on System z performance hints and tips

Eberhard Pasch
epasch@de.ibm.com

Session 2591
Trademarks

The following are trademarks of the International Business Machines Corporation in the United States and/or other countries.

- DB2®
- DB2 Connect
- DB2 Universal Database
- e-business logo
- IBM®
- IBM eServer
- IBM logo®
- Informix®
- System z
- Tivoli®
- WebSphere®
- z/VM®
- zSeries®
- z/OS®
- ECKD
- Enterprise Storage Server®
- FICON
- FICON Express
- HiperSocket
- OSA
- OSA Express

* Registered trademarks of IBM Corporation

The following are trademarks or registered trademarks of other companies.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Java and all Java-related trademarks and logos are trademarks of Sun Microsystems, Inc., in the United States and other countries.

SET and Secure Electronic Transaction are trademarks owned by SET Secure Electronic Transaction LLC.

* All other products may be trademarks or registered trademarks of their respective companies.

Notes:

Performance is in Internal Throughput Rate (ITR) ratio based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user’s job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput improvements equivalent to the performance ratios stated here.

IBM hardware products are manufactured from new parts, or new and serviceable used parts. Regardless, our warranty terms apply.

All customer examples cited or described in this presentation are presented as illustrations of the manner in which some customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics will vary depending on individual customer configurations and conditions.

This publication was produced in the United States. IBM may not offer the products, services or features discussed in this document in other countries, and the information may be subject to change without notice. Consult your local IBM business contact for information on the products or services available in your area.

All statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.

Information about non-IBM products is obtained from the manufacturers of those products or their published announcements. IBM has not tested those products and cannot confirm the performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Prices subject to change without notice. Contact your IBM representative or Business Partner for the most current pricing in your geography.
Agenda

- **Tuning**
  - Application
    - C/C++
  - Middleware
    - Java
  - Linux
    - Networking
    - DASD
  - Virtualization
  - Hardware / Setup

- **Monitoring**
  - Oprofile
  - SCSI
Optimize your stack in the right direction

- Diminishing effect of tuning efforts
  - Application design
  - Application implementation
  - Middleware
  - Operating system
  - Virtualization layer
  - Hardware
Impact of newer software releases

Virtualization Performance - Throughput Comparison

- Hardware:
  System z9<sup>TM</sup> 2094-S18
  8-way 1.65 GHz

- Software upgraded
  - z/VM  5.2 → 5.3
  - Java  1.4 → 1.5
  - WebSphere Application server  6.0.2 → 6.1.0.11
  - DB2  8.2 → 9.1

Keep your system current!

The newer software levels provides a significant improvement in throughput!
Optimizing C/C++ code

- **Use -O3 optimization as default**
  - no debugging options
  Further optimization:
  - architecture dependent options
    - `-march` = values `<G5, z900, z990> <z9-109 with gcc-4.1> <z10 with patched gcc 4.3>`
    - `-mtune` = values `<G5, z900, z990> <z9-109 with gcc-4.1> <z10 with patched gcc 4.3>`
  - inline assembler functions

- **Next step: application design**
  - dynamic or static linking
  - Avoid `-fPIC` for executables
  - right use of inlined C / C++ functions

- **Fine Tuning: additional general options on a file by file basis**
  - `-funroll-loops -ffast-math`
Results of changing compiler options

- Using `-O3` instead of no optimization cuts runtime up to 50%
GCC Cross Compile Performance on System z

**Compile Times**
Cross Compile for m68k Linux Kernel

<table>
<thead>
<tr>
<th>Number of CPU's</th>
<th>Compile time in seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>X3920 (5,00%)</td>
</tr>
<tr>
<td></td>
<td>z10 (10,00%)</td>
</tr>
<tr>
<td>2</td>
<td>X3920 (15,00%)</td>
</tr>
<tr>
<td></td>
<td>z10 (30,00%)</td>
</tr>
<tr>
<td>4</td>
<td>X3920 (25,00%)</td>
</tr>
<tr>
<td></td>
<td>z10 (50,00%)</td>
</tr>
<tr>
<td>8</td>
<td>X3920 (40,00%)</td>
</tr>
<tr>
<td></td>
<td>z10 (50,00%)</td>
</tr>
</tbody>
</table>

**Relative Performance Improvements**
z10 compared to X3920

<table>
<thead>
<tr>
<th>Number of CPU's</th>
<th>Performance Improvement in Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Difference Percent</td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>
Java basics

- Try to use latest Java version
  - Up to 20% release to release improvements
  - True as well for newer service releases (SR)

- Make sure that you've got enabled JIT
  - Verify Java output and look for “JIT enabled: jitc”

- Don't use Java in batch mode:
  - If you do 100 calls “java -jar myprogram.jar” you compile myprogram 100 times
    - can take more CPU power than the program itself
    - the JIT compiler can't do its optimization work
  - Instead pull the loop inside the Java program and call “java -jar myprogram100.jar” once
Java heap size

- **Useful parameters**
  - Setting heap size: `-Xms (minimal), -Xmx (maximal), use min=max`
  - `-verbose:gc` -- monitor GC

- **Max heap <= available memory**
  - Avoid paging - Linux and VM
  - remember: heap memory will be used eventually!

- **Larger heap size usually implies better performance**
  - in 31bit SLES8, SLES9 & SLES10 use `/proc/<pid>/mapped_base` to define heaps up to 1.7 GB
  - In 31bit RHEL4 environments use flex-mmap mechanism
    - Watch out for prelinked applications!
    - Works also in 31bit emulation on 64 bit distros
Mapped_base HowTo

- Only available for Novell distribution SLES8,9,10 (31 bit)
- PID is the process ID of the process you want to change
  - In bash `$$` gives you the current process, from any process `/proc/self/...` works as well
- Display memory map of any PID by `cat /proc/PID/maps`
- Check the mapped_base value by `cat /proc/PID/mapped_base`
- Change value to e.g. 256 Mb by `echo 268435456 > /proc/PID/mapped_base`
Java Results 64-bit

- Java improvements through newer JVM and JIT
- Improvements through new hardware
- 64-bit Java is production ready
- Java 6: new option -Xcompressedref (stay tuned)
Java tuning exercise for z10 – workload specific

- Split workload so that multiple 31 bit JVMs can be used
- `-Xms1600m -Xmx1600m` : maximize heap, same value
- `-Xlp` : use large pages for the heap (new z10 feature)
- `-Xgcpolicy:gencon` : change garbage collection policy to treat long and short lived objects differently, results in shorter pause times
- `-Xmo800m -Xmn800m` : explicitly set the size for old and new objects (nursery too small by default)
- `-Xnoloa` : don't use special large object area in the heap

Further reading:
z10 Performance: Java workload

- System z versus System p

Throughput

- 31-bit JVM z9
- 31-bit JVM z10
- 32-bit JVM p570

CPU clock speeds
- z9 - 1.7 GHz
- z10 - 4.4 GHz
- p570 - 4.7 GHz
Networking performance

- Which connectivity to use:
  - External connectivity:
    - Use new 10 GbE cards with MTU 8992
    - Attach OSA directly to Linux guest image
  - Internal connectivity:
    - Hipersockets for LPAR-LPAR communication
    - VSwitch for guest-guest communication
- For really busy network devices consider to
  - use channel bonding
    - Increase the number of inbound buffers in the qeth driver
      - Device has to be offline
      - `# echo <number> > /sys/bus/ccwgroup/drivers/qeth/<device_bus_id>/buffer_count`
- Channel bonding for HA creates only a small overhead
- Choose your MTU size carefully
  - Avoid fragmentation, lots of small packages can drive up CPU utilization
How to improve disk performance

- **Hardware choices**
  - Use SCSI instead of ECKD
  - Use FICON instead of ESCON
    - 4Gb FICON > 2Gb FICON > 1Gb FICON

- **Utilize your hardware**
  - Use “striped” logical volumes from different ranks
  - Consider using HiperPAV
  - Carefully set up your storage system
  - With D8000 – new option to stripe on storage server (see Session 2590)
Effect of dasdfmt block size on throughput and capacity

- Use 4k block size on ECKD DASDs whenever possible!

<table>
<thead>
<tr>
<th>dasdfmt blocksize</th>
<th>disk space</th>
</tr>
</thead>
<tbody>
<tr>
<td>512b</td>
<td>3.5G</td>
</tr>
<tr>
<td>1024b</td>
<td>4.7G</td>
</tr>
<tr>
<td>2048b</td>
<td>6G</td>
</tr>
<tr>
<td>4096b</td>
<td>6.8G</td>
</tr>
</tbody>
</table>

Graph showing MB/s for different block sizes and read/write types.
“On Demand Timer” patch

- Linux uses HZ based timer interrupts
- Timer interrupts for idle guests create unnecessary overhead
- Starting with SLES8: enable & disable on the fly
  - `/proc/sys/kernel/hz_timer`
  - `1` = timer interrupts occurring every 10 ms
  - `0` = timer interrupts generated on demand only
- Included in SLES9, SLES10 and RHEL4, RHEL5 s390/s390x distributions
CMM

- 2 methods available:
  - VMRM-CMM (VM Resource Manager – Cooperative Memory Management) aka CMM1
    - Resource manager controls the size of the guests
  - CMMA (Collaborative Memory Management Assist) aka CMM2
    - Linux indicates which pages don't need to be saved
- Both methods show performance improvements when z/VM hits a system memory constraint.
CMM1 scenario

- Large Oracle guests, total used Linux memory = 2x of z/VM central storage, OLTP workload
- Advantages with CMM1
- Guests did not suffer from smaller page cache

![Throughput for 10 guests](image)
CMM2 scenario

- **Workload**
  - 15 guests, touching all their memory, all z/VM storage used. A guest orders now 150MB, 500MB, 1.5GB of memory. We measure the duration of this operation

- **Result**
  - In case of sudden memory claims CMM2 is the best choice

### Duration of claiming Memory

<table>
<thead>
<tr>
<th>Memory in MiB to be claimed</th>
<th>Duration avg. [sec] w/o cmm</th>
<th>Duration avg. [sec] w/ cmm2</th>
<th>Duration avg. [sec] w/ cmm1</th>
</tr>
</thead>
<tbody>
<tr>
<td>150</td>
<td>2.53</td>
<td>9.38</td>
<td>0.2</td>
</tr>
<tr>
<td>500</td>
<td>11.96</td>
<td>13.5</td>
<td>0.62</td>
</tr>
<tr>
<td>1500</td>
<td>49.32</td>
<td>279.38</td>
<td>13.5</td>
</tr>
</tbody>
</table>

### Improvement factor for claiming Memory (normalized)

<table>
<thead>
<tr>
<th>Memory in MiB to be claimed</th>
<th>Improvement factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>150</td>
<td>2</td>
</tr>
<tr>
<td>500</td>
<td>6</td>
</tr>
<tr>
<td>1500</td>
<td>18</td>
</tr>
</tbody>
</table>
# of CPUs per Linux image

- Use as few virtual CPUs as possible

- For LPAR definitions:
  - # all virtual CPUs : # real CPUs \(\leq 4:1\)

- For z/VM:
  - # of guest CPUs \(\leq\) # of CPUs for VM (LPAR)
  - Don't define more CPUs than you really need!

- You don't get done more by defining more CPUs!

- Automatic adaption with cpuhotplugd (see session 2590)
Linux command 'top' – the snapshot tool

- Adds new field “CPU steal time”
  - Is time Linux wanted to run, but the hypervisor was not able to schedule CPU
  - Is included in SLES10 and RHEL5
Sysstat – the 'long' term data collection

- Contains four parts
  - sadc: data gatherer - stores data in binary file
  - Sar: reporting tool - reads binary file and converts it to readable output
  - Mpstat: processor utilization
  - Iostat: I/O utilization

- “steal time” included starting version 7.0.0

- Install the sysstat package and configure it depending on your distribution (crontab)
  - by default data is collected in /var/log/sa

- More info at: http://perso.orange.fr/sebastien.godard and with “man sar” on your system
Oprofile – the Open Source sampling tool

- Oprofile offers profiling of all running code on Linux systems, providing a variety of statistics.
  - By default, kernel mode and user mode information is gathered for configurable events

- System z hardware currently does not have support for hardware performance counters, instead timer interrupt is used
  - Enable the hz_timer(!)

- The timer is set to whatever the jiffy rate is and is not user-settable

- Novell / SUSE: oprofile is on the SDK CDs

- More info at:
  - http://oprofile.sourceforge.net/docs/
Oprofile – short how-to


  ```
  sysctl -w kernel.hz_timer=1
  gunzip /boot/vmlinux-2.6.5-7.201-s390x.gz
  ```

  ```
  opcontrol --vmlinux=/boot/vmlinux-2.6.5-7.201-s390x
  opcontrol --start
  ```

  `<DO TEST>`

  ```
  opcontrol --shutdown
  opreport
  ```
### Oprofile – output example

CPU: CPU with timer interrupt, speed 0 MHz (estimated)
Profiling through timer interrupt

<table>
<thead>
<tr>
<th>vma</th>
<th>samples</th>
<th>%</th>
<th>app name</th>
<th>symbol name</th>
</tr>
</thead>
<tbody>
<tr>
<td>80002840</td>
<td>5862</td>
<td>34.8970</td>
<td>mcf_base.z_Linux</td>
<td>price_out_impl</td>
</tr>
<tr>
<td>800012c8</td>
<td>5221</td>
<td>31.0811</td>
<td>mcf_base.z_Linux</td>
<td>refresh_potential</td>
</tr>
<tr>
<td>80003cb4</td>
<td>4398</td>
<td>26.1817</td>
<td>mcf_base.z_Linux</td>
<td>primal_bea_mpp</td>
</tr>
<tr>
<td>80003b60</td>
<td>408</td>
<td>2.4289</td>
<td>mcf_base.z_Linux</td>
<td>sort_basket</td>
</tr>
<tr>
<td>0001a67c</td>
<td>345</td>
<td>2.0538</td>
<td>vmlinux</td>
<td>default_idle</td>
</tr>
<tr>
<td>800013d8</td>
<td>138</td>
<td>0.8215</td>
<td>mcf_base.z_Linux</td>
<td>flow_cost</td>
</tr>
<tr>
<td>800033bc</td>
<td>98</td>
<td>0.5834</td>
<td>mcf_base.z_Linux</td>
<td>update_tree</td>
</tr>
<tr>
<td>800020f8</td>
<td>88</td>
<td>0.5239</td>
<td>mcf_base.z_Linux</td>
<td>dual_feasible</td>
</tr>
<tr>
<td>800036a4</td>
<td>72</td>
<td>0.4286</td>
<td>mcf_base.z_Linux</td>
<td>primal_iminus</td>
</tr>
<tr>
<td>8000323c</td>
<td>40</td>
<td>0.2381</td>
<td>mcf_base.z_Linux</td>
<td>write_circulations</td>
</tr>
<tr>
<td>80002720</td>
<td>24</td>
<td>0.1429</td>
<td>mcf_base.z_Linux</td>
<td>insert_new_arc</td>
</tr>
</tbody>
</table>
SCSI statistics

- In SLES9 and SLES10 SCSI statistics can be collected.

- If debugfs is mounted at /sys/kernel/debug/, all the statistics data collected can be found at /sys/kernel/debug/statistics/.

- The names of these subdirectories consist of:
  - zfcp-<device-bus-id> for an adapter and
  - zfcp-<device-bus-id>-<WWPN>-<LUN> for a LUN.

- Each subdirectory contains two files:
  - data file
  - definition file

- `echo on=1 > definition` enables the data gathering.

- `echo on=0 > definition` disables the data gathering. By default data gathering is off.

- `echo data=reset > definition` resets the counters to 0.
SCSI statistics example

```
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
request_sizes_scsi_read 0x63000 24
request_sizes_scsi_read 0x6f000 19
request_sizes_scsi_read 0x12000 19
...
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
...
```
Comparing SCSI and ECKD request sizes

- Similar request sizes for sequential and random I/O

Request sizes (IOzone 16 processes)
Comparing SCSI and ECKD latencies (1)

- SCSI sequential write latencies are longer

![Graph comparing SCSI and ECKD latencies](image-url)
Comparing SCSI and ECKD latencies (2)

- SCSI sequential read latencies are shorter
Visit us!

- Linux on zSeries Tuning Hints and Tips

- Linux-VM Performance Website:

- WAS 6.1 tuning guide
Questions
Backup – older but still valid topics for reference
/proc/dasd/statistics (1)

- Linux can collect performance stats on DASD activity as seen by Linux(!)
- Turn on with
  `echo on > /proc/dasd/statistics`
- Turn off with
  `echo off > /proc/dasd/statistics`
- To reset: turn off and then on again
- Can be read for the whole system by
  `cat /proc/dasd/statistics`
- Can be read for individual DASDs by
  `tunedasd -P /dev/dasda`
/proc/dasd/statistics (2)

- Collects statistics (mostly processing times) of IO operations
- Each line represents a histogram of times for a certain operation
- Operations split up into the following:

  - Histogram of I/O till ssch
  - Histogram of I/O between ssch and IRQ
  - Histogram between I/O and End

### /proc/dasd/statistics (3)

Tue Jan 18 20:52:50 EST 2005
21155901 dasd I/O requests
with 433275376 sectors (512B each)

<table>
<thead>
<tr>
<th>__&lt;4</th>
<th>___8</th>
<th>__16</th>
<th>__32</th>
<th>__64</th>
<th>128</th>
<th>256</th>
<th>512</th>
<th>1k</th>
<th>__2k</th>
<th>__4k</th>
<th>__8k</th>
<th>_16k</th>
<th>_32k</th>
<th>_64k</th>
<th>128k</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>3774298</td>
<td>838941</td>
<td>352193</td>
<td>232188</td>
<td>43222</td>
<td>30563</td>
<td>16163</td>
<td>1403</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>51</td>
<td>0</td>
<td>24686</td>
<td>204678</td>
<td>524222</td>
<td>2803252</td>
<td>500319</td>
<td>537993</td>
<td>249088</td>
<td>316175</td>
<td>111592</td>
<td>15932</td>
<td>1005</td>
<td>26</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3498191</td>
<td>51615</td>
<td>86168</td>
<td>21601</td>
<td>2756</td>
<td>1927</td>
<td>4348</td>
<td>22793</td>
<td>177758</td>
<td>138465</td>
<td>955964</td>
<td>214188</td>
<td>61200</td>
<td>42284</td>
<td>9075</td>
<td>621</td>
</tr>
<tr>
<td>14</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>41819</td>
<td>517428</td>
<td>890743</td>
<td>3323127</td>
<td>21897</td>
<td>23329</td>
<td>103966</td>
<td>280533</td>
<td>79777</td>
<td>6056</td>
<td>282</td>
<td>10</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4531949</td>
<td>633301</td>
<td>75411</td>
<td>41903</td>
<td>4984</td>
<td>791</td>
<td>516</td>
<td>48</td>
<td>40</td>
<td>3</td>
<td>3</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

# of req in chanq at enqueuing (1..32)

<table>
<thead>
<tr>
<th>__&lt;4</th>
<th>___8</th>
<th>__16</th>
<th>__32</th>
<th>__64</th>
<th>128</th>
<th>256</th>
<th>512</th>
<th>1k</th>
<th>__2k</th>
<th>__4k</th>
<th>__8k</th>
<th>_16k</th>
<th>_32k</th>
<th>_64k</th>
<th>128k</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>3658672</td>
<td>277906</td>
<td>128989</td>
<td>97542</td>
<td>1125789</td>
<td>27</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
How to collect z/VM monitor data


- 5 basic steps
  - Create monitor DCSS
  - Setup userid to issue monwrite command
  - Start and configure monitor
  - Start monwrite
  - Stop monwrite and save data
How to insert Linux data in z/VM monitor stream

- Enable your guest for inserting data into the monitor stream
  - set APPLMON option to user direct

- Insert Linux modules
  - modprobe appldata_mem
  - modprobe appdata_os
  - modprobe appldata_net_sum

- Turn on monitoring
  - echo 1 > /proc/sys/appldata/timer
  - echo 1 > /proc/sys/appldata/mem
  - echo 1 > /proc/sys/appldata/os
  - echo 1 > /proc/sys/appldata/net_sum

- Details can be found in chapter 15 of Device Drivers, Features, and Commands (SC33-8281-02)
z/VM 2 GB considerations

- **Solution:** upgrade z/VM to 5.2 or 5.3 level
- **Read at**
- **Old workarounds**
  - **Cooperative Memory Management**
  - fixed I/O buffers with kernel 2.6 and ECKD
  - distribute your guests to multiple z/VMs
  - Move large guest to LPAR
Problem:
- with many guests in z/VM it can happen that CP is busy executing diagnose instructions for the guest

What's behind it:
- in a so-called spin lock, Linux guests give their CPU share back to the hipervisor using DIAG 44
- Hipervisor can be overloaded

Solution:
- Linux tries to get a lock n times before issuing a DIAG
- Value of n is adjustable in /proc/sys/kernel/spin_retry (default 1000)
- Included in latest SLES9 + SLES10 + RHEL4 + RHEL5
Tick based CPU Time inaccuracy

kernel context
user context

user context

kernel context
user context

user context

kernel context
user context

timer ticks

user context

kernel context

user context

timer ticks

user context

kernel context

user context

2/100 s
7/100 s
1/100 s
8/100 s
4/100 s
5/100 s
Tick based CPU accounting on virtual systems

Kernel context
User context

Timer ticks
Kernel context
User context

Timer ticks
Kernel context
User context
New Virtual CPU time accounting

Host action: stop start stop start stop start
Guest action: stpt stpt stpt stpt stpt stpt stpt stpt

kernel context
user context

kernel time
user time
steal time

1/100 s
5/100 s
3/100 s
5/100 s
1/100 s
3/100 s

stpt = Store CPU Timer