Jens Freimann

19/08/2015

Pushing the limits: 1000 guests per host and beyond

Jens Freimann

19/08/2015

Pushing the limits: 1000 guests per host and beyond

(Q)Emus predominately travel in pairs, and while they can form large flocks, this is an atypical social behaviour that arises from the common need to move towards a new food source. - Wikipedia

[2]

Agenda

What is this about?

The test setup

What we tested

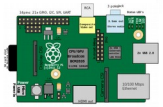
Challenges

Conclusion

What is this about?

KVM runs on...

[4]



[5]

What is this about?

- Finding limits of our machine/architecture and KVM/QEMU/Libvirt
- How far can we go given a sufficient host configuration and where does it break?
  - How many guests?
  - How many block and network devices?
  - Which resources are necessary?
  - Wh configuration settings are needed to support large configurations?
- Testing the extreme limits, not necessarily recommended to run in production
- No performance measurements

[6]

Agenda

What is this about?

The test setup

What we tested

Challenges

Conclusion

Test setup

Host:

System Type:	z13 (Type: 2964 Model 701 NE1)
Central Storage:	2 TB
CPUs:	64 (shared)

Guests:

RHEL 7.1 & SuSE 12

Memory:	1024 MB
CPUs:	2
Disk:	Raw image files on SCSI disks
Network:	Linux bridged

[7]

Test setup

~# cat /proc/cpuinfo

vendor_id : IBM/S390

processors : **64**

bogomips per cpu: 20325.00

features : esan3 zarch stfle msa ldisp eimm dfp edat etf3eh highgprs te

cache0 : level=1 type=Data scope=Private size=128K line_size=256 associativity=8

cache1 : level=1 type=Instruction scope=Private size=96K line_size=256 associativity=6

cache2 : level=2 type=Data scope=Private size=2048K line_size=256 associativity=8

cache3 : level=2 type=Instruction scope=Private size=2048K line_size=256 associativity=8

cache4 : level=3 type=Unified scope=Shared size=65536K line_size=256 associativity=16

cache5 : level=4 type=Unified scope=Shared size=491520K line_size=256 associativity=30

processor 0: version = 00, identification = 11C667, machine = 2964

processor 1: version = 00, identification = 11C667, machine = 2964

processor 2: version = 00, identification = 11C667, machine = 2964

…

processor 63: version = 00, identification = 11C667, machine = 2964

Agenda

What is this about?

The test setup

What we tested

Challenges

Conclusion

What we tested

- Start **4096** guests
- Assign **4096** virtio-block devices to a guest
- Use **16384** virtio-block devices per host
- Assign **1024** virtio-net devices to a guest
- Assign large amount of ressources to the host,
  e.g. use more than **8**TB RAM for the host
- Concurrent live guest migrations of > **64** guests
- ...

Agenda

What is this about?

The test setup

What we tested

Challenges

Conclusion

General: memory dumps can take very long

- Amount of memory in servers has increased faster than bandwidth of memory and storage
- On s390 we have the possibility to take memory dumps even in very early or late stages
- Not kdump but IBM stand-alone dump tools
- Downside: no filtering of memory and only 1 CPU available
- Writing a memory dump of a multi-TB system can take days

- For kvm **guests** with huge memory sizes guest debugging with gdb is a workaround

Host: adding network devices

- 2048 macvtap devices:
  - found problem in **s390 qeth driver** code
  - Each macvtap device is assigned its own MAC address
  - List of MAC addresses in hardware needs to be **re-populated every time we add a macvtap device**
  - Causes long delays (76.5 seconds for 50 devices) and eventually leads to stalls
  - Working on **fix for qeth** driver

- System limits hit:
  - Linux bridges limited to **1024 ports** → add more bridges

  - Max. number of IPv6 routes
    IPv6: Maximum number of routes reached, consider increasing route/max
    → Set **ipv6.sysctl.ip6_rt_max_size** = 8192

Attaching 4096 disks to the host

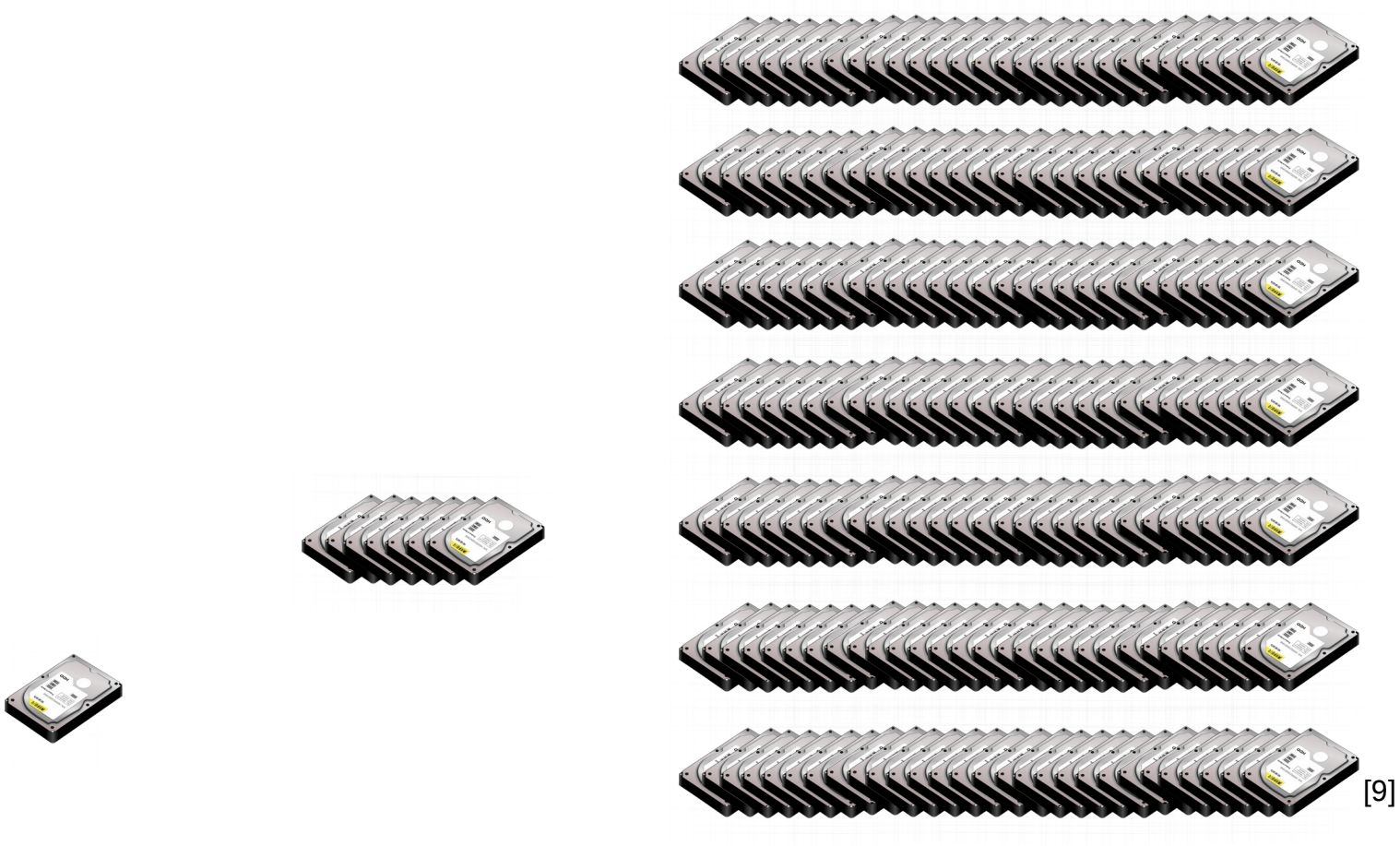
- System limits hit:
  - Max. number of inotify watches
    systemd-udevd: ***inotify_add_watch****(7, /dev/dm-1784, 10) failed:* ***No space left on device***
    → set fs.inotify.max_user_watches=<some big number>
  - Number of file descriptors
    - ...

Counting file descriptors...

- Mostly eventfds, one per virtqueue
- For example: a virtio-net device is using
  - one virtqueue for TX/RX (depending on multiqueue or not)
  - one virtqueue for CTRL
  - /dev/tap
  - /dev/vhost-net + 3 irqfds
  - 8 FDs in total
- Add 1000 virtio-net devices
- We need many!

[8]

Use many virtio-block devices



[9]

Attach 4096 disks to a guest
- What could possibly go wrong, right?

- Tested this a while ago, but limit was 1024 due to select()
- Change from select() to poll() between QEMU 1.4 – 1.5
- Target: 4096, started with: 1000
- Guest startup time with 1000 disks attached: 13 minutes
- Problem: Calling bdrv_drain_all() in each virtio_reset() call for every BlockDriverState
- Fix: Call aio_poll() per AioContext instead, save many aio_poll()'s

  → down to 16 seconds for 1000 disks
- Upstream commit:

  commit f406c03c093f1451ac0ba7fde31eeb78e5e5e417
  Author: Alexander Yarygin <yarygin@linux.vnet.ibm.com>
  Date: Wed Jun 10 14:38:17 2015 +0300

  block: Let bdrv_drain_all() to call aio_poll() for each AioContext
- 4096 disks is possible with this fix

Using 16384 virtio-block disks

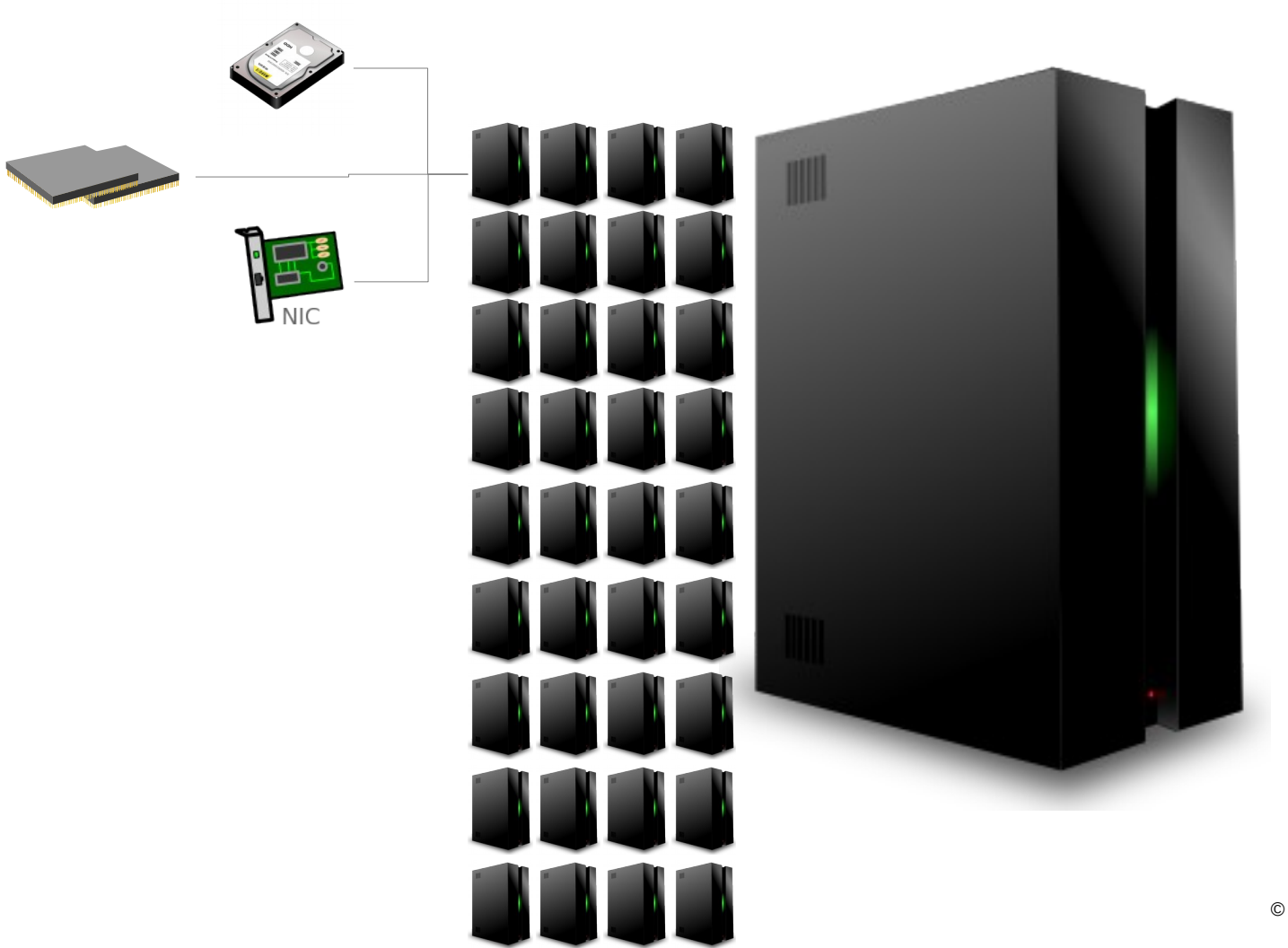
- Started 16 guests, each assigned 1024 virtio-block devices

  qemu-system-s390x: -drive file=/guestimages/data3/mydomain.img,if=none,id=drive-virtio-disk0,format=raw,serial=skel,cache=none,aio=native: could not open disk image /guestimages/data3/zs93k1g08166.img: ***Could not refresh total sector count: Bad file descriptor***

- Discovered that /proc/sys/fs/aio-nr is the same as /proc/sys/fs/aio-max-nr
- Not enough aio contexts available
- Fix by increasing **fs.aio-max-nr**

Starting 4096 guests

… because it's such a nice even number



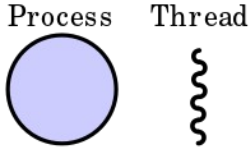
Starting 4096 guests

… because it's such a nice even number

Many system limits had to be tweaked:

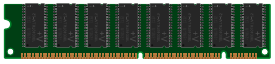
journal: Failed to find user record for uid '107': ***Too many open files***
→ create file /etc/systemd/system/libvirtd.service.d/openfiles.conf and add **LimitNOFILE=4096**
Number of file descriptors (of course!)

Increase max PTYs
journal: internal error: process exited while connecting to monitor: 2015-04-04T15:41:07.159443Z
qemu-system-s390x: -chardev pty,id=charconsole0: ***Failed to create chardev***
sysctl.conf: kernel.pty_max = 8192



Max processes/threads exceeded

- pthread_create_returns -EAGAIN → insufficient resources
- **LimitNPROC=1048576** in libvirtd.service
- In qemu.conf **max_process=8192**



Not enough memory → increase to 2TB
kernel: [sched_delayed] sched: RT throttling activated
followed by multiple page allocation failures.
kernel: top: ***page allocation failure****: order:4, mode:0x1040d0*

Sources: [14],[15],[16]

Starting 4096 guests

systemd says hello :)

- On systems with systemd installed there is a static service called systemd-machined
- Libvirt calls systemd-machine::CreateMachine() for every guest started
- Message handling in older versions (at least until v208) of systemd was inefficient

  CreateMachine: Did not receive a reply. Possible causes include: the remote application did not send a reply, the message bus security policy blocked the reply, the reply timeout expired, or the network connection was broken.

- CreateMachine() timed out because of buggy message handling in systemd
- Fixed with systemd-219 by rewriting message loop

[10]

Starting 4096 guests

Guests running

```
top - 11:03:47 up 1 day, 59 min,  4 users,  load average: 133.44, 400.06, 312.86
Tasks: 8569 total,  24 running, 8545 sleeping,   0 stopped,   0 zombie
%Cpu(s): 60.7 us,  9.8 sy,  0.0 ni, 19.2 id,  0.0 wa,  0.7 hi,  7.0 si,  2.6 st
KiB Mem:  21102485+total, 14060241+used, 70422444+free,   418040 buffers
KiB Swap: 10240000+total,        0 used, 10240000+free.  1124296 cached Mem
  scroll coordinates: y = 1/8569 (tasks), x = 1/12 (fields)
    PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
  39910 qemu      20   0 1788656 490400   8028 S 135.5  0.0   9:33.44 qemu-system-s39
  39732 qemu      20   0 1739188 509052   8032 S 131.7  0.0   9:55.96 qemu-system-s39
  39952 qemu      20   0 1713584 512480   8024 S 131.4  0.0   9:36.17 qemu-system-s39
  39936 qemu      20   0 1727924 486156   8028 S 130.8  0.0   9:30.80 qemu-system-s39
  39685 qemu      20   0 1699568 483452   8024 S 130.1  0.0  10:17.34 qemu-system-s39
  39874 qemu      20   0 1635056 486732   8028 S 130.1  0.0   9:26.30 qemu-system-s39
  39718 qemu      20   0 1649392 486188   8028 S 129.8  0.0  10:00.58 qemu-system-s39
  40058 qemu      20   0 1708464 510620   8028 S 129.8  0.0   9:20.78 qemu-system-s39
  39981 qemu      20   0 1758960 485816   8028 S 129.5  0.0   9:14.50 qemu-system-s39
```

Starting 4096 guests

Guests running

```
[root@zs93k1 ~]# virsh list --all | wc -l
8796
[root@zs93k1 ~]# virsh list | wc -l
3985
```

```
[root@zs93k1 ~]# free -h
             total       used       free     shared    buffers     cached
Mem:          2.0T       1.3T       661G       268M       410M       1.1G
-/+ buffers/cache:       1.3T       662G
Swap:         976G         0B       976G
```

Running CPU and network stress tools

Starting 4096 guests

a running guest

```
top - 03:17:40 up 18:29,  1 user,  load average: 1.00, 1.01, 1.05
Tasks:  73 total,   1 running,  72 sleeping,   0 stopped,   0 zombie
%Cpu(s): 35.8 us,  0.0 sy,  0.0 ni, 50.0 id,  0.0 wa,  0.0 hi,  0.0 si, 14.2 st
KiB Mem:    917576 total,   269456 used,   648120 free,    34100 buffers
KiB Swap:        0 total,        0 used,        0 free.   177384 cached Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
 2093 root      20   0 1064712  19020   6116 S 72.00 2.073 567:23.50 java
    1 root      20   0    6104   4004   2076 S 0.000 0.436   0:02.08 systemd
    2 root      20   0       0      0      0 S 0.000 0.000   0:00.00 kthreadd
    3 root      20   0       0      0      0 S 0.000 0.000   0:00.02 ksoftirqd+
    5 root       0 -20       0      0      0 S 0.000 0.000   0:00.00 kworker/0+
    6 root      20   0       0      0      0 S 0.000 0.000   0:00.11 kworker/u+
    7 root      rt   0       0      0      0 S 0.000 0.000   0:00.00 migration+
    8 root      20   0       0      0      0 S 0.000 0.000   0:00.00 rcu_bh
    9 root      20   0       0      0      0 S 0.000 0.000   0:00.13 rcu_sched
   10 root      rt   0       0      0      0 S 0.000 0.000   0:00.00 migration+
   11 root      20   0       0      0      0 S 0.000 0.000   0:00.02 ksoftirqd+
   13 root       0 -20       0      0      0 S 0.000 0.000   0:00.00 kworker/1+
   14 root       0 -20       0      0      0 S 0.000 0.000   0:00.00 khelper
   15 root      20   0       0      0      0 S 0.000 0.000   0:00.00 kdevtmpfs
   16 root       0 -20       0      0      0 S 0.000 0.000   0:00.00 netns
   17 root       0 -20       0      0      0 S 0.000 0.000   0:00.00 writeback
   18 root       0 -20       0      0      0 S 0.000 0.000   0:00.00 kintegrit+

zs93k1g85009:~ # ps aux| grep java
root      2093 83.8  2.0 1064712 19020 ?       Sl   Aug04 567:44 java -classpath /products/muncher/Muncher.jar com.ibm.wlm.tools.Muncher
root      2963  0.0  0.0   2732   624 ttysclp0 R+   03:18   0:00 grep --color=auto java
zs93k1g85009:~ #
```

Starting 4096 guests

strict overcommit and glibc per-thread arenas

- We use many threads in QEMU, for vcpu threads, io threads,...
- Glibc has per-thread memory pools (arenas)
- Arenas are created per thread but are limited on the basis of cores the system has
- MALLOC_ARENA_MAX is calculated by
  ((NUMBER_OF_CPU_CORES) * (sizeof(long) == 4 ? 2 : 8))
  - 8 memory pools per core * 64 MB arena size
  - For example with 64 cores i.e. = 64 * 8 = 512 arenas
  - 512 arenas * 64 MB per arena = 32 GB
  - Uses lots of address space

Starting 4096 guests

strict overcommit and glibc per-thread arenas (continued)

- …
- usually kernel memory accounting treads this correctly, but...
- … with strict overcommit the entire memory is accounted as “in use”
- malloc() fails much earlier
- This will get worse as number of cores increase!
- Capped by MALLOC_MAX_ARENA
- Control by setting <qemu:env name='MALLOC_ARENA_MAX' value='16'/>

Migrating many guests concurrently

Migrating many guests concurrently – TCP port range for migration

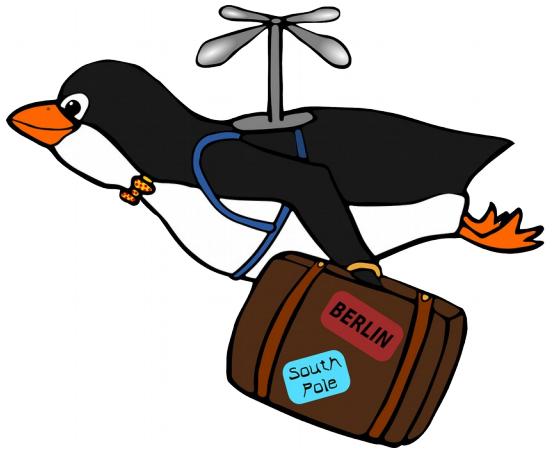
- There's a default port range for migration in libvirt
- When we're running out of ports:
  error: internal error: Unable to find an unused port in range 'migration' (49152-49215)
- Ports can be specified as part of the virsh command line to circumvent this
- ...or configurable with migration_port_min and migration_port max in qemu.conf

[11]

Migrating many guests concurrently – max. SSH connections

- sshd has a default limit of 10 sessions
- When testing migration of > 64 guests we had to increase this limit
- Change /etc/ssh/sshd_config and set MaxSessions to a higher number

 # Authentication
...
#MaxSessions 128



[12]

QEMU: Memslot limitation

```
/*
 * Some of the bitops functions do not support too long bitmaps.
 * This number must be determined not to exceed such limits.
 */
#define KVM_MEM_MAX_NR_PAGES ((1UL << 31) - 1)
```

- ((1UL << 31) -1) * 4k pages → 8TB
- Limit is per memslot
- Currently we have only one memslot for RAM.
- Would need to implement support for multiple memslots in s390x

Processor Memory

Model	Minimum	Maximum
N30	64 GB	2.5 TB**
N63	64 GB	5.0 TB
N96	64 GB	7.5 TB
NC9	64 GB	10.0 TB
NE1	64 GB	10.0 TB

Host: system with 9TB memory

commit c47386fcb81833e266b0e955f127c92c195dcdc2

Author: Martin Schwidefsky <schwidefsky@de.ibm.com>

Date: **Wed May 13 14:33:22 2015** +0200

7cded342c: **s390/mm: correct return value of pmd_pfn**

Git commit 152125b7a882df36a55a8eadbea6d0edf1461ee7
"s390/mm: implement dirty bits for large segment table entries"
broke the pmd_pfn function, it changed the return value from
'unsigned long' to 'int'. This **breaks all machine configurations with memory above the 8TB line**.

Cc: stable@vger.kernel.org # 3.17+

Signed-off-by: Martin Schwidefsky <schwidefsky@de.ibm.com>

s390x/kvm: Irqchip route reallocation

- We create a new irqfd per virtqueue
- Every time a new irqfd is created we reallocate the irqchip routing table in the kernel
- With many devices and many virtqueues this causes delays
- Interface of KVM_SET_GSI_ROUTING allows to add a bunch of routing entries in one go. However currently it is not used that way. One entry is added after another.
- For every entry that's to be added we do ioctl(KVM_SET_GSI_ROUTING) and then in the kernel we allocate a new table, re-populate it and add the new entry.
- Fixed by adding all routes to the routing table in QEMU first and then do the ioctl to commit the routing table to the kernel

commit 35d4a70da363e3909beb5d74f35649db25a6004d
Author: Jens Freimann <jfrei@linux.vnet.ibm.com>
Date: Mon Jul 27 16:53:27 2015 +0200

s390x/kvm: make setting of in-kernel irq routes more efficient

KVM: IRQchip limit

- The in-kernel irqchip is limited to 1024 pins
- For s390 already extended to 4096
- When limit exceeded devices fall back to qemu internal irqfd handling
- Problem: one irqfd is set up per virtqueue (mapping to queue indicator), so queues per irqchip are exhausted fast
  - virtio-net devices possible = 4096 / 3 virtqueues = 1365
- Ideas:
  - Bump up number of pins even further
  - Tweak irqfd interface to support a payload containing an offset to the summary indicators.
    - Then we would not have to set up routes for every single virtqueue

Agenda

What is this about?

The test setup

What we tested

Challenges

Conclusion

Conclusion

- Linux (and KVM) scales quite well, most often a matter of adjusting ulimits
  - ipv6.sysctl.ip6_rt_max_size
  - sshd: *#MaxSessions 128*
  - libvirtd: min_port_range, max_port_range
  - systemd: LimitNOFILE
  - sysctl: kernel.pty.max, fs.aio-max-nr, fs.inotify.max_user_watches
- One problem in systemd which is already fixed upstream
- In-kernel IRQchip limits virtio devices
- Apart from some smaller issues, no major code findings in Libvirt/QEMU/KVM
- No fundamental design problems in our stack Libvirt/QEMU/KVM

Questions ?

Sources: [13]

Thank you!

Legal Statement

This work represents the view of the author and does not necessarily represent the view of IBM.

IBM, IBM(logo), z/Architecture, zSeries, Enterprise Systems Architecture/390, ESA/390, Enterprise Systems Architecture/370, ESA/370 and System/360 are trademarks and/or registered trademarks of International Business Machines Corporation in the United States, other countries, or both.

Linux is a registered trademark of Linus Torvalds.
Other company, product and service names may be trademarks or service marks of others.

Sources

- [1] https://pixabay.com/static/uploads/photo/2013/07/12/14/10/explosion-147909_640.png
- [2] http://uncyclopedia.wikia.com/wiki/File:EmuArmy.jpg
- [3] https://www.flickr.com/photos/107244436@N07/10807513426/
- [4] https://en.wikipedia.org/wiki/IBM_700/7000_series#/media/File:IBM_Electronic_Data_Processing_Machine_-_GPN-2000-001881.jpg
- [5] https://en.wikipedia.org/wiki/IBM_zEnterprise_System#/media/File:Glowing_IBM_z13_and_clock_-_cropped.JPG

  , Agiorgio
- [6] https://www.flickr.com/photos/80497449@N04/10531620296/, Nicolas Raymond
- [7] IBM z13 Image Library
- [8] https://pixabay.com/en/explosion-detonation-blast-burst-147909/

Sources

- [9] https://openclipart.org/detail/131323/hard-disk
- [10] http://ngc891.blogdns.net/?p=383
- [11] https://en.wikipedia.org/wiki/Ad%C3%A9lie_penguin#/media/File:Automated_weighbridge_for_Ad%C3%A9lie_penguins_-_journal.pone.0085291.g002.png
- [12] https://www.flickr.com/photos/66944824@N05/17171489773
- [13] https://www.flickr.com/photos/colinkinner/2200500024
- [14] https://upload.wikimedia.org/wikipedia/commons/thumb/9/9a/Process_vs._thread.svg/200px-Process_vs._thread.svg.png
- [15] https://openclipart.org/download/178295/RAM-Board.svg
- [16] https://openclipart.org/detail/212873/terminal