Case study of ftrace: Interrupt

David Chang
dchang@suse.com

Case study I (Interrupt)

Bsc#969524 - Worse network performance (xen kernel vs default kernel)

- SLE11-SP3
- Assigned To: David Bond
- wasL3:44945

Synopsis

- problem with network performance on the virtual servers with XEN kernel
- the response time of default kernel are better than XEN kernel
  - default kernel: the average ping response time is 0.197 ms
  - XEN kernel: the average ping response time is 7,230 ms
- **It can not be reproduced by our side**

Bsc#969524 - Worse network performance (xen kernel vs default kernel)

- SLE11-SP3
- problem with network performance on the virtual servers with XEN kernel
- the response time of default kernel are better than XEN kernel
  - default kernel: the average ping response time is 0.197 ms
  - XEN kernel: the average ping response time is 7,230 ms
- **It can not be reproduced by our side**

trace.sh

```
echo "Starting trace..."
# sp4 trace-cmd doesn't know -C
echo "global" > /sys/kernel/debug/tracing/trace_clock
cat /sys/kernel/debug/tracing/trace_clock &> "$output/trace-cmd.out"

# use explicit form of &>> to appease sp4 bash
trace-cmd record -p nop -e all -b 500000 -o "$output/trace.dat" -s
1000000 >>"$output/trace-cmd.out" 2>&1 &
```

- -p run command with plugin enabled
- -e run command with event enabled
- -b change kernel buffersize (in kilobytes per CPU)
- -o data output file [default trace.dat]
- -s sleep interval between recording (in usecs) [default: 1000]

Output of trace.sh

root@linux-gslf:/tmp# **./trace.sh**
Recording initial state...
Starting trace...
Please use the following shell to run the test case. When done, exit the shell to let this script complete.
root@linux-gslf:/tmp# **ping** -c10 192.168.70.20
PING 192.168.70.20 (192.168.70.20) 56(84) bytes of data.
64 bytes from 192.168.70.20: icmp_seq=1 ttl=64 time=4.06 ms
64 bytes from 192.168.70.20: icmp_seq=2 ttl=64 time=0.278 ms
…
--- 192.168.70.20 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 8998ms
rtt min/avg/max/mdev = 0.218/0.642/4.067/1.142 ms
root@linux-gslf:/tmp# exit
Recording final state...
Output available in "/tmp/output-20160617-215252+0000.tar.bz2"

Logs analysis

64 bytes from sc169.sscc.ae.aena.es (89.1.6.245): icmp_seq=8 ttl=64 time=0.379 ms
64 bytes from sc169.sscc.ae.aena.es (89.1.6.245): icmp_seq=9 ttl=64 time=**41.8 ms**
64 bytes from sc169.sscc.ae.aena.es (89.1.6.245): icmp_seq=10 ttl=64 time=0.301 ms

- At the trace:

starttime syscall(args) = retval # duration, cpu
2459408.969389 sendmsg(3, 7ff2edba7300, 800) = 64 # 0.000019, 001
2459408.969410 recvmsg(3, 7ffd2a6d8290, 0) = 84 # **0.029262**, **001 -> 000**

  – the recvmsg call for this echo request took around 29ms and initially ran on cpu1 but was migrated to cpu0

More trace log

idle-0 [000] 2459408.**941071**: **irq_handler_entry**: **irq=96 name=hpvsa0**

[...]

idle-0 [000] 2459408.**997356**: **irq_handler_exit**: irq=96 ret=handled

idle-0 [000] 2459408.997359: irq_handler_entry: irq=88 name=timer

idle-0 [000] 2459408.997361: hrtimer_cancel: hrtimer=0xffff880fbfc07860

idle-0 [000] 2459408.997362: hrtimer_expire_entry: hrtimer=0xffff880fbfc07860 now=246403123296212 6 function=tick_sched_timer/0x0

idle-0 [000] 2459408.997366: softirq_raise: vec=1 [action=TIMER]

idle-0 [000] 2459408.997368: softirq_raise: vec=7 [action=SCHED]

idle-0 [000] 2459408.997370: hrtimer_expire_exit: hrtimer=0xffff880fbfc07860

[…]

- irq 96 for the **hpvsa driver ran for 56ms** which is an inordinate amount of time for an irq handler to run and **which delayed all other irqs on cpu0**, including those for tg3 (eth4). Hence the reception of the icmp echo reply was delayed.

Result

- The problem is that the irq handler for the hpvsa driver running in dom0 sometimes takes very long to complete
- Ths hpvsa module in this case:

  Kernel Status -- Tainted: P X

  TAINT: (P) Proprietary module has been loaded

  TAINT: (X) Modules with external support loaded

  module=hpvsa license=HP supported=external
- HPE suggest updating hpvsa to latest version
- No response from the customer in the end

Case study II (Interrupt)

bsc#924919 - find or ls on nfs-mounted directory slows down on "getdents" (SLES 12)

- SLE12
- Assigned To: Benjamin Poirier
- wasL3:42782

Synopsis

- NFS clients running SLES have a terrible performance doing a "ls -R" or "find" on a nfs-mounted directory
  - it may hang for up to 10 seconds
  - Using strace on the above calls shows that the hangs are at "getdents" calls
- The SLE12 and SLE11 client are identical machines (same CPU, same ethernet etc.)
- a SLES 11sp3 nfs client mounting the test directory from the server with nfsv4 doesn't have these hang

Analysis of NFS

- the actual difference between SLE11 and SLE12 is the buffer size used by find.
- This buffer size is chosen by glibc and is based on the block size reported by the filesystem. This in turn is based (for NFS) on the 'wsize' value which is 1048576, exactly what 'strace' shows.
- mount with "wsize=32768" you should get the same performance as SLE11
  - Result: No different

- In turns out to be a network issue from tcpdump trace (e1000e)
  - In the SLE11 trace, all the replies are a single IP packet
  - In the SLE12 trace, it is sometimes 2 packets

Testing with tg3 adapter

- Redid the tests several times, tg3 is faster and even faster than the SLE11 client
  - sle12 with e1000e and wsize=100: 1:18
  - sle12 with **tg3** and wsize=100: **0:09**
  - sle11 with e1000e and wsize=100: 0:12
- The problem must have been introduced in the e1000e driver between SLE11 sp3 and SLE12

Need more informaiton for network

- Try to reproduce the issue
- Examine previous tcpdump trace log
- Request supportconfig
  - More information about network hardware and configuration
- Compare with GRO enable/disable (receive)
  - Reproduce the issue after disabling gro on the client
  - ethtool -K eth0 gro off

What is GRO (Generic Receive Offload)

- Combines received packets to a single larger packet and passes them to the higher up network stack in order to increase performance (reduce cpu overhead)
- Improving LRO (Large Receive Offload)
  - Limited to TCP/IPv4
  - Packet merging doesn't preserve all state
  - Problematic with packet forwarding(>MTU) and bridge
- Protocol-independent
- The criteria for which packets can be merged is greatly restricted
- Merged packets can be resegmented losslessly

Using NIC offload

- **ethtool -k**|--show-features|--show-offload **DEVNAME**

  Get state of protocol offload and other features

- **ethtool -K**|--features|--offload **DEVNAME FEATURE on|off** ...

  Set protocol offload and other features

```
linux-kyyb:/home/dchang # ethtool -k eth0
Features for eth0:
[...]
tcp-segmentation-offload: on
        tx-tcp-segmentation: on
        tx-tcp-ecn-segmentation: off [fixed]
        tx-tcp6-segmentation: on
udp-fragmentation-offload: off [fixed]
generic-segmentation-offload: on
generic-receive-offload: on
large-receive-offload: off [fixed]
rx-vlan-offload: on
tx-vlan-offload: on
[...]

linux-kyyb:/home/dchang # ethtool -K eth0 gro off
[...]
generic-receive-offload: off
[...]
```

More finding

- The e1000e trace shows the problem clearly:
  - there is an interrupt firing towards the end of napi processing
  - but it does not cause a napi_schedule().
  - This interrupt is almost certainly for the first instance of the tcp segment which ends up staying the nic queues until the retransmission is received
  - and a new interrupt causes both copies of the data to be received.

Receive data of a NIC

- NIC receives data from the network.
- NIC uses DMA to write the network data to RAM
- NIC raises an interrupt
- Device driver’s registered interrupt handler is executed
- NIC cleared interrupt, so that it can generate interrupt for new packet
- NAPI softirq poll loop is started with a call to napi_schedule

trace.sh

```
cd /sys/kernel/debug/tracing
echo 0 > tracing_on
echo nop > current_tracer
if [ $(cat per_cpu/cpu*/stats | awk '/^entries/ && $2 != 0' | wc -l) -gt 0 ]; then
  echo "Dropping unconsummed trace buffers..."
fi
echo > trace
echo 100000 > buffer_size_kb
echo global > trace_clock
echo "napi:*" > set_event
echo net:netif_receive_skb >> set_event
if ! (echo "e1000e:*" >> set_event 2>/dev/null || echo "tg3:*" >> set_event
2>/dev/null); then
  echo "Warning: could not activate driver tracepoints" >/dev/stderr
fi
```

trace.log

4980.739166 e1000e_intr 0
!!! intr while napi poll is running
Presumably, this happens after interrupts have been re-enabled at the
end of e1000e_poll.

4980.739187 napi_poll napi poll on napi struct ffff880797f80c10 for device eth0, done 2 / 64

frame 97 is waiting in queue for 200ms until tcp retransmission

4980.941330 e1000e_intr 0
4980.941332 napi_schedule napi schedule on napi struct ffff880797f80c10 for device eth0

4980.941335 dev_gro_receive gro receive of skb ffff880799215180 (type 0x86dd seq 1859408914) on napi struct ffff880797f80c10 (gro_count 0) for device eth0
4980.941336 dev_gro_result: gro result of skb ffff880799215180 completed (null) same_flow 0 free 0 flush 2048
4980.941336 napi_skb_finish skbaddr= ffff880799215180 ret= 3
4980.941337 netif_receive_skb dev=eth0 skbaddr=ffff880799215180 len=1128
frame 97 in pcap

Resolution

- Interrupts are firing at unexpected times
- Interrupt automasking for e1000e adapters in msi-x mode does not work.
  - This is because one of the registers controlling this feature was not programmed correctly by the driver
- Turning gro off does not really prevent the problem but instead reduces this critical window by ~10x
- Instead, operating the adapter in INTx or MSI mode (instead of MSI-X, by using the IntMode module option) is a proper workaround.

- Patches submitted to upstream
  - 4d432f6 e1000e: Remove unreachable code
  - 16ecba5 e1000e: Do not read ICR in Other interrupt
  - a61cfe4 e1000e: Do not write lsc to ics in msi-x mode
  - 0a8047a e1000e: Fix msi-x interrupt automask


SUSE®
We adapt. You succeed.