Optimizing igb and ixgbe network driver scaling performance

Alexander Duyck,
LAN Access Division,
Intel Corp.
Agenda

- The state of the igb and ixgbe drivers then and now
- Configuring the kernel and system for best performance
- Getting to the root cause of the performance improvements
- Where we still might have room to improve
The state of igb & ixgbe over a year ago

- Problem: igb was over 40% faster than the ixgbe.
- Solution: Refactor ixgbe to more closely match igb.
The state of igb & ixgbe now

- Mission accomplished!
- How did we get from there to here?
- Could we be masking over some other issues?
- Where do we go from here?
Configuring the kernel for best performance

- Disable any and all config options that add to the size of the sk_buff w/o any benefit to your testing
  - IPv6, IPSEC, IOAT, Netfilter, Qos support, and network actions
  - Result should be an sk_buff that fits in 3 cache lines
  - May be unrealistic but we are testing the drivers, not the stack

- Disable IOMMU
  - Generates significant DMA map/unmap overhead
  - May also be disabled via kernel parameter
Configuring the system for best performance

- Evenly load the nodes and memory channels with memory
  - The test system had 2 nodes with 3 channels of memory each.
  - I loaded 1 2G DIMM of memory on each channel for a total of 12GB
- Evenly distribute device interrupts on CPUs
  - `set_irq_affinity.sh` script included with ixgbe driver can now handle this task
The test configuration

- System running dual Xeon X5680 @ 3.33Ghz
  - Running 2.6.35.14 kernel
  - Connected back to back with Spirent Smartbits 6000c containing a XLW-3720a card
  - Tests typically ran with 64 simultaneous UDP streams
- 3 Basic tests
  - 3 Queue pktgen
  - 3 Queue receive & drop at ip_rcv
  - 8 Queue bidirectional single port routing
- Why select only 3/8 queues?
  - They had not reached line rate in any tests I was running
  - Routing showed that the stack consumed about 25% of the total
  - Thus I end up with 3 CPUs RX, 3 CPUs TX, and 2 CPUs stack
Results, Round 1

IXGBE Performance

- Pktgen 3 thread
- Routing 8 Thread
- RX Drop 3 thread
Something doesn't seem right..

- 2.1.4
  - Dropped support for RSS w/ UDP ports
    - This changed the work distribution
- 3.0.14
  - Removed trans_start from Tx path
    - This change was pushed upstream over a year prior
- 3.2.10
  - Removed last_rx from Rx path
    - Another change that made it upstream over a year prior
Results, Round 2

IXGBE Performance

<table>
<thead>
<tr>
<th>Version</th>
<th>2.0.75.7</th>
<th>2.0.84.11</th>
<th>2.1.4</th>
<th>3.0.14</th>
<th>3.1.17</th>
<th>3.2.10</th>
<th>3.3.9</th>
<th>3.4.24</th>
<th>3.5.11</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pkts</td>
<td>6000000</td>
<td>7000000</td>
<td>8000000</td>
<td>9000000</td>
<td>10000000</td>
<td>11000000</td>
<td>12000000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Performance root cause

- 3.1.17
  - Set TXDCTL.PTHRESH to 32, allowing hardware to prefetch descriptors in groups of 8.
- 3.2.10
  - Combined all hotpath items in adapter struct into a single read-mostly cacheline
- 3.5.11
  - Enabled SRRCTL.DROP_EN when RX multiqueue is enabled and flow control is disabled
Cutting the memory overhead

- Combine all adapter fields accessed in hot-path into single cache-line to prevent cache pollution
- Configure hardware to batch descriptor reads & writes
Knowing when to drop a packet

- Rx FIFO is yet another buffer that can introduce delays
  - Prone to head of line blocking in multiqueue configurations
  - Can only move as fast as the quickest ring
- SRRCTRL.DROP_EN drops packets from Rx FIFO
  - Only drops packets when no buffers are available on ring to DMA to
  - Allows faster rings to keep processing while slower rings drop packets
  - Reduces overall dropped packet rate
  - Mutually exclusive with features like flow control and DCB
Results, Round 3

IXGBE Performance

Packets Per Second

Version

Pktgen 3 thread  Routing 8 Thread  RX Drop 3 thread
Performance root cause

- 2.0.84.11
  - Performance regression due to alignment change of `ixgbe_clean_tx_irq` from 64 byte to 16 byte

- 3.1.17
  - Combined `ixgbe_tx_map` and `ixgbe_tx_queue` calls into a single function
  - Fused all NAPI cleanup into `ixgbe_poll`
  - General cleanup of TX and RX path

- 3.5.11
  - Store values in “first” `tx_buffer_info` struct sooner
  - Avoid unnecessary modification of TX descriptor in cleanup
Reduce & reuse to cut memory usage

- Store values in the Tx buffer_info structure instead of in the stack
- Make Tx/Rx cleanup paths leave descriptors rings untouched until new buffers are available
- Reduce memory reads in Tx path by separating read-mostly and write-mostly parts of the ring structure
- Allocate memory on local node to reduce memory access time
- Allocate sk_buff such that skb->head is fully used; sizes 512, 1.5K, 3K, 7K, & 15K are optimal
- Use only ½ of a page to allow for page reuse via page user count
Reducing code complexity

ixgbe-2.0.75.7
ixgbe_msix_clean_tx
ixgbe_msix_clean_rx
ixgbe_msix_clean_many
ixgbe_poll
ixgbe_clean_txonly
ixgbe_clean_rxonly
ixgbe_clean_rxtx_many

ixgbe-3.5.11
ixgbe_msix_clean_rings
ixgbe_poll

• Avoid unnecessary duplication of effort in maintenance.
• By only having one ixgbe_poll call the compiler can optimize by in-lining all of the various functions used by the call.
Where do we still have room to improve?

- At this point ixgbe consumes only about 15% of the total CPU utilization.

- Duplicate overhead in ixgbe_poll and __alloc_skb due to cache misses while zeroing & reading skb header.

- 90% of spin lock overhead appears to be due to Qdisc lock taken in sch_direct_xmit.

- The top functions consuming CPU time are:
  - _raw_spin_lock: 20.14%
  - ixgbe_poll: 7.76%
  - eth_type_trans: 6.58%
  - __alloc_skb: 4.69%
  - ip_rcv: 4.68%
  - ixgbe_xmit_frame_ring: 4.36%
  - ip_forward: 3.48%
  - ip_route_input_common: 3.37%
  - dev_queue_xmit: 3.30%
  - kfree: 3.24%
  - __netif_receive_skb: 3.03%
  - kmem_cache_free: 2.81%
  - kmem_cache_alloc_node: 2.70%
  - kmem_cache_alloc_node_notrace: 2.56%
  - ixgbe_alloc_rx_buffers: 1.93%
  - __phys_addr: 1.93%
  - memcpy: 1.73%
  - skb_release_data: 1.73%
  - ixgbe_select_queue: 1.32%
  - dev_hard_start_xmit: 1.30%
  - swiotlb_dma_mapping_error: 1.26%
  - ip_finish_output: 1.14%
  - __kmalloc_node: 1.08%
  - local_bh_enable: 1.03%
What if we delay skb init?

Routing Performance

__alloc_skb split in two

Routing Performance

<table>
<thead>
<tr>
<th>Function</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>_raw_spin_lock</td>
<td>20.84%</td>
</tr>
<tr>
<td>ixgbe_poll</td>
<td>8.56%</td>
</tr>
<tr>
<td>ixgbe_xmit_frame_ring</td>
<td>5.38%</td>
</tr>
<tr>
<td>ip_rcv</td>
<td>4.40%</td>
</tr>
<tr>
<td>eth_type_trans</td>
<td>3.90%</td>
</tr>
<tr>
<td>kfree</td>
<td>3.85%</td>
</tr>
<tr>
<td>dev_queue_xmit</td>
<td>3.83%</td>
</tr>
<tr>
<td>ip_route_input_common</td>
<td>3.45%</td>
</tr>
<tr>
<td>kmem_cache_free</td>
<td>2.98%</td>
</tr>
<tr>
<td>__netif_receive_skb</td>
<td>2.89%</td>
</tr>
<tr>
<td>kmem_cache_alloc_node_notrace</td>
<td>2.83%</td>
</tr>
<tr>
<td>kmem_cache_alloc_node</td>
<td>2.68%</td>
</tr>
<tr>
<td>ip_forward</td>
<td>2.25%</td>
</tr>
<tr>
<td>__phys_addr</td>
<td>2.24%</td>
</tr>
<tr>
<td>memcpy</td>
<td>1.79%</td>
</tr>
<tr>
<td>ixgbe_alloc_rx_buffers</td>
<td>1.78%</td>
</tr>
<tr>
<td>is_swiotlb_buffer</td>
<td>1.57%</td>
</tr>
<tr>
<td>dev_hard_start_xmit</td>
<td>1.52%</td>
</tr>
<tr>
<td>init_skb_lite</td>
<td>1.36%</td>
</tr>
<tr>
<td>skb_release_data</td>
<td>1.36%</td>
</tr>
<tr>
<td>swiotlb_map_page</td>
<td>1.33%</td>
</tr>
<tr>
<td>ip_finish_output</td>
<td>1.14%</td>
</tr>
<tr>
<td>__alloc_skb_lite</td>
<td>1.11%</td>
</tr>
<tr>
<td>local_bh_enable</td>
<td>1.10%</td>
</tr>
</tbody>
</table>
What if we didn't have a Qdisc?

Routing Performance
With and Without Qdisc

Routing Performance
With and Without Qdisc

<table>
<thead>
<tr>
<th>Function</th>
<th>ixgbe-no-q</th>
<th>ixgbe-3.5.11</th>
</tr>
</thead>
<tbody>
<tr>
<td>ixgbe_poll</td>
<td>22.26%</td>
<td></td>
</tr>
<tr>
<td>ip_rcv</td>
<td>5.95%</td>
<td></td>
</tr>
<tr>
<td>__alloc_skb</td>
<td>5.12%</td>
<td></td>
</tr>
<tr>
<td>ixgbe_xmit_frame_ring</td>
<td>4.97%</td>
<td></td>
</tr>
<tr>
<td>__netif_receive_skb</td>
<td>4.54%</td>
<td></td>
</tr>
<tr>
<td>ip_route_input_common</td>
<td>3.78%</td>
<td></td>
</tr>
<tr>
<td>kfree</td>
<td>3.69%</td>
<td></td>
</tr>
<tr>
<td>kmem_cache_alloc_node</td>
<td>3.62%</td>
<td></td>
</tr>
<tr>
<td>ip_forward</td>
<td>3.48%</td>
<td></td>
</tr>
<tr>
<td>kmem_cache_free</td>
<td>3.26%</td>
<td></td>
</tr>
<tr>
<td>kmem_cache_alloc_node_notrace</td>
<td>2.78%</td>
<td></td>
</tr>
<tr>
<td>eth_type_trans</td>
<td>2.69%</td>
<td></td>
</tr>
<tr>
<td>dev_queue_xmit</td>
<td>2.49%</td>
<td></td>
</tr>
<tr>
<td>__phys_addr</td>
<td>2.23%</td>
<td></td>
</tr>
<tr>
<td>ixgbe_alloc_rx_buffers</td>
<td>2.14%</td>
<td></td>
</tr>
<tr>
<td>memcpy</td>
<td>1.91%</td>
<td></td>
</tr>
<tr>
<td>__raw_spin_lock</td>
<td>1.69%</td>
<td></td>
</tr>
<tr>
<td>skb_release_data</td>
<td>1.68%</td>
<td></td>
</tr>
<tr>
<td>dev_hard_start_xmit</td>
<td>1.51%</td>
<td></td>
</tr>
<tr>
<td>swiotlb_map_page</td>
<td>1.35%</td>
<td></td>
</tr>
<tr>
<td>ip_finish_output</td>
<td>1.21%</td>
<td></td>
</tr>
<tr>
<td>__kmalloc_node</td>
<td>1.14%</td>
<td></td>
</tr>
<tr>
<td>swiotlb_dma_mapping_error</td>
<td>1.07%</td>
<td></td>
</tr>
<tr>
<td>is_swiotlb_buffer</td>
<td>1.01%</td>
<td></td>
</tr>
</tbody>
</table>