Motivation

Network I/O has high CPU cost

- TX: **350%** cost of linux
- RX: **310%** cost of linux

CPU cost for TCP connection at 1 Gbps
(xen-unstable (03/16/2007); PV Linux guest; X86 - 32bit)
Outline

• Performance Analysis for network I/O RX path (netfront/netback)
• Network I/O RX optimizations
• Network I/O TX optimization
Performance Analysis
For Network I/O
RX Path
Experimental Setup

• Machines: HP Proliant DL580 (client and server)
  – P4 Xeon 2.8 Ghz, 4 CPU (MT disabled), 64 GB, 512 KB L2, 2MB L3
  – NIC: Intel E1000 (1 Gbps)

• Network configuration
  – Single switch connecting client and server

• Server configuration
  – Xen unstable (c.s.14415 – March 16, 2007) (default xen0/xenU configs)
  – Single guest (512MB), dom0 also with 512MB

• Benchmark
  – Simple UDP micro benchmark (1gbps, 1500 bytes packets)
• Cost of data copy is significant both in Linux and Xen
  – Xen has the cost of an additional data copy
• Xen guest kernel alone uses more CPU than linux
• Most cost for Xen code is in dom0
Major Xen overhead is: grant table & dom page map/unmap

Copy grant: several expensive atomic operations (lock instr.prefix):
- Atomic cmpxchg operation for updating status field (grant in use)
- Increment/decrement grant usage counters (multiple spinlock open)
**Dom0 Kernel Cost**

- Bridge/network is large component in dom0 cost (Xen summit 2006)
  - Can be reduced if netfilter bridge config option is disabled
- Xen new code: Netback, hypercall, swiotlb
- Higher interrupt overhead in Xen: extra code in evtchn.c
- Additional high cost functions in Xen (accounted in “other”)
  - `spin_unlock_irqrestore()`, `spin_trylock()`
• What do we need to do to disable bridge netfilter by default?
  – Should we add a netfilter hook in netback?
Overhead in guest

- Netfront: 5 times more expensive than e1000 driver in Linux
- Memory op (mm): 2 times more expensive in Xen (?)
- grant table:
  - high cost of atomic cmpxchg operation to revoke grant access
- “Other”: spin_unlock_irqrestore(), spin_trylock() (same as dom0)
Source of Netfront Cost

- Netback copy packet data into netfront page fragments
- Netfront copies first 200 bytes of packet from fragment into main socket buffer data area
- Large netfront cost is due to this extra data copy
Opportunities for Improvement on RX path
Reduce RX head copy size

- No need to have all headers in main SKB data area
- Copy only Ethernet header (14 bytes)
- Network stack stack copies more data as needed
Move grant data copy into guest CPU

- Dom0 grant access to data and guest copy it using copy grant
- Cost of 2\textsuperscript{nd} copy to user buffer is reduced as data is already in the guest CPU cache (assumes cache is not evicted due to user process delaying read)
- Additional benefit: Improves dom0 (driver domain) scalability as more work is done at the guest side
- Data copy is more expensive in guest (alignment problem)
Grant copy align problem

- copy is expensive when destination start is not at word boundary
- Fix: Copy also 2 prefix bytes → source and destination now aligned
Fixing grant copy alignment

- Grant copy in guest becomes more efficient than in current Xen
- Grant copy in dom0: destination is word aligned but
  - Source is not word aligned
  - Can also be improved by copying additional prefix data
    - Either 2 or (2+16) bytes
Fixing alignment in current Xen

- Source alignment reduces copy cost
- Source and dest. at same buffer offset has better performance
  - Reason (?): maybe because same offset in cache?
- Copy cost in dom0 is still more expensive than copy in guest.
  - Different cache behavior
Dom0 copy has more L2 cache misses than guest copy
- Dom0 copy has lower cache locality
- Guest post multiple pages on IO ring.
  - All pages in ring must be used before the same page can be reused

For guest copy, pages are allocated on demand and reused more often improving cache locality
Possible grant optimizations

• Define new simple copy grant:
  – Allow only one copy operation at a time
  – No need to keep grant usage counters (remove lock)
• avoid cost of atomic cmpxchg operations
  – Separate fields used for enabling grant and usage status
• Avoid incrementing/decrementing page ref counters
  – Use an RCU scheme for page deallocation (lazy deallocation)
Potential savings in grant modifications

- Results are optimistic
  - Still need to implement grant modifications
  - Results are based on eliminating current operations
Coalescing netfront RX interrupts

- NIC (e1000) already coalescing HW interrupts (~10 packets/int)
- Batching packets can provide additional benefit
  - 10% for 32 packets
  - But adds extra latency
  - Dynamic coalescing scheme should be beneficial
Coalescing effect on Xen cost

Xen in guest context

- Except grant and domain page map/unmap, all other Xen costs are amortized by larger batches
  - An additional reason for optimizing grant
Combining all RX optimizations

- Cost of network I/O for RX can be significantly reduced
  - From ~250% to ~70% overhead (compared to linux)
- Largest improvement comes for moving grant copy to guest CPU
Optimization
for TX path
Lazy page mapping on TX

- Dom0 only needs to access packet headers
- No need to map guest pages with packet payload
  - NIC device access memory directly through DMA
- Avoid mapping guest page on packet TX
  - Copy packet headers using I/O ring
  - Modified grant operation returns machine address (for DMA) but does not map page in dom0.
- Provide page fault handler to deal with cases in which dom0 needs to access payload
  - Packet to dom0/domU; netfilter rules
Benefit of lazy TX page mapping

- Performance improvement for TX optimization
  - ~10% for large TX
  - ~8% for TCP RX due to ACKs
- Some additional improvement may be possible with grant optimizations
Questions ?