Interrupt Coalescing in Xen with Scheduler Awareness

Michael Peirce & Kevin Boos
Outline

- Background
- Hypothesis
- vIC-style Interrupt Coalescing
- Adding Scheduler Awareness
- Evaluation
Background

Xen split block drivers
Background: Xen block drivers

Xen hypervisor

dom0

blkback
driver

blkfront

domU guest
Background: ring buffers
Background: interrupt event channels
Focus on blkback

interrupt: requests pending

interrupt: responses pending
Hypothesis
Hypothesis

1) Coalescing interrupts in Xen will increase throughput of block devices at minor latency cost (vIC)
   ○ fewer interrupts reduces CPU overhead

2) Scheduler awareness will improve upon existing coalescing policies by reducing latency
   ○ less coalescing towards end of timeslice
   ○ minimal reduction in throughput
Conventional Interrupt Coalescing

VMware vIC
VMware-style Coalescing (vIC)

- Interrupt coalescing is absent in Xen
- Added conventional coalescing based on VMware’s vIC
- Interrupt delivery ratio based on configurable parameters:
  - IOPS threshold
  - CIF threshold
  - (Epoch period)

- Implemented in dom0’s kernel, in xen_blkback module
  - On each block_io completion event, decide whether to deliver interrupt
Default Interrupt Delivery (no coalescing)

- Device interrupts from Hypervisor
- 10 interrupts

Timeline:
- 20ms
- 30ms
Increasing Disk Throughput in vIC

Core 1
Guest 1 Timeslice

Core 2
Guest 2 Timeslice

Dom0

Device interrupts from Hypervisor

5 interrupts

20ms 30ms
Scheduler Awareness
Latency Problems in vIC

Device interrupts from Hypervisor
Reducing Latency

- **Core 1**
  - **Guest 1 Timeslice**
  - Device interrupts from Hypervisor

- **Core 2**
  - Device interrupts from Hypervisor
  - **Dom0**

- **Guest 2 Timeslice**

20ms to 30ms time span
Hybrid approach: vIC + scheduler awareness

- Should we use a separate interrupt delivery policy based on scheduler info alone?
  - No, too coarse-grained and unintelligent

- Use scheduler info to configure vIC’s parameters & ratio

- **Hard guarantee** that interrupts will be delivered right at the very end of a timeslice
  - “end of timeslice” cutoff is configurable
Exposing scheduler info from hypervisor

● Easy way: add hypercall to retrieve scheduler info
  ○ Pros: easy to implement, info generated on demand
  ○ Cons: high overhead, long latencies → stale info

● Hard way: shared memory region with dom0
  ○ Pros: info is fresh, available immediately
  ○ Cons: info is updated constantly, very difficult to implement
Implementing shared scheduler info

- Xen allocates a shared page for each domain when it boots
  - boot info, arch-specific details, interrupt masks/bit vectors

- Added scheduler info to shared page
  - One per domain (except idle & dom0)
  - Only visible in dom0
  - Updated in hypervisor’s `schedule()`

- Much difficulty with time synchronization
Scheduler Awareness Implementation Diagram

Scheduler Awareness Implementation Diagram
Scheduler Awareness Policy

We choose to deliver an interrupt when:

$$\text{remaining time in timeslice} < \frac{1}{(\text{Ratio} \times \text{IOPS})}$$
Evaluation
Evaluation Setup

- Default credit scheduler enabled
- dom0 pinned to two CPU cores, reserved for dom0 only
- All guests pinned to the same single core
  - Eliminates effects of migration
  - Imitates guest CPU contention on high-density servers

- Tools to generate disk workload:
  - Copy files with dd, small block size to create more I/O requests
  - Custom interrupt injection tool
Evaluation Questions

- Can we achieve higher throughput with minimal latency?
- Can we achieve the same increased throughput as vIC with less latency?
Throughput Measurement

- Copy files using dd tool with small block size of 8 & 512 bytes
  - Measure execution time of 1GB file transfer
Throughput Results

One guest performing I/O, others hogging CPU
Throughput Results

All guests performing I/O, all guests hogging CPU
Latency Measurement

- Instrumented frontend block driver in the guest kernel
  - Assign (guest-specific) unique ID to each request
  - Start timer when request is submitted
  - End timer when response is received
Latency Results

![Latency Results Graph]

- **No Coalescing**
- **vIC**
- **Scheduler Aware**

Latency (ms)

- 2 guests
- 4 guests
- 2 guests both I/O

30
Conclusion
Concluding Remarks

- As expected, interrupt coalescing does increase throughput
- Scheduler awareness reduces latency while maintaining the increased throughput
- Overall effects are less significant than expected
  - Need more demanding test environment

- Future work: change beginning of timeslice behavior

- Our experience developing on Xen was mediocre
  - Tedious, slow, constant reboots
  - Multiple independent code bases (dom0, xen, domU)
  - Limited debug logs, no post-crash log
  - Toolset support and networking is a nightmare