





## \* **M**Ware<sup>®</sup>

# Optimizing Storage Performance with Calibrated Interrupts

OSDI 2021

Presenter: Igor Smolyar

\* Denotes co-first authors with equal contribution

#### Motivation: rapid increase in storage performance



## Motivation 1: faster I/O interface--software and hardware

CPU-bound async workload, io depth 256



## Motivation 2: SSDs latency improvements

Device-bound sync workload, io depth 1



interrupt



Interrupts are expensive. Firing an interrupt on each completion hampers performance.

Must Coalesce!

#### Interrupt coalescing

No coalescing: interrupt on each request



Coalescing problem: latency/throughput trade-off Which request is latency sensitive?

## Eternal question: when to generate an interrupt?

- Difficult question!
- Can heuristics help?
  - Sense the workload and act accordingly
- Let's see how heuristics work in NICs

#### Eternal question: when to generate an interrupt?

## Heuristics are suboptimal

20

15

10

5

 $\bigcirc$ 

diff [µsec]

18

Mellanox CX-5 TCP RR lat. [µsec] Intel XL710 TCP RR latency [µsec] 200 80 120 70 66 100 latency [µsec] 160 15 14 60 80 120 9.8 40 60 28 2930 80 23 23 23 40 1.3 1.5 0.61.5 20 40 20 0.3 0  $\bigcirc$ 0 4KB 16KB 64KB 256B 1KB 4KB 16KB 64KB 64B 256B 1KB 64B default diff no coalescing adaptive heuristics

Devices resort to use heuristics, which is suboptimal for performance.

Problem: Semantic Gap

Missing Semantics: when to generate an interrupt?

#### Our Solution

Software has the knowledge when it needs an interrupt. I/O requests should convey this knowledge to the device.

For NVMe devices we calibrate\* interrupts with Urgent and Barrier flags.

\* To calibrate: to adjust precisely for a particular function

## Semantics: Urgent flag for latency sensitive request #1



Request #1 latency

## Performance benefits of Urgent

Mixed workload, two threads: sync (urgent) and async (throughput)



\* Cinterrupts adaptive coalescing, superior to NVMe coalescing

#### Urgent improves latency.

#### Can we also optimize throughput?

#### Semantics: Barrier flag means end of a batch

Two threads, each submits batch of 3 requests Cinterrupts: mark with Barrier the last request in the batch



## Performance benefits of Barrier

Async workload, two threads submit in batches



\* Cinterrupts adaptive coalescing, superior to NVMe coalescing

#### Barrier optimizes throughput.

Generates interrupt exactly when batch is ready.

## Implementation: passing flags with Linux kernel support

- RWF\_URGENT and RWF\_BARRIER syscall flags
- Applications use explicitly in
  - preadv2/pwritev2
  - io\_submit
- For legacy apps, kernel sets default annotations
  - mark sync or blocking with Urgent
  - mark async with Barrier

## More in the paper

- More macro & micro benchmarks
- Cinterrupts adaptive coalescing
- Emulation details
- Cinterrupts for networking
- and more...



Optimizing Storage Performance with Calibrated Interrupts Dan Tsafrir# <sup>‡</sup>VMware Research

ever, creates a trade-off between request latency and the interrupt rate. For the workloads we inspected, CPU utilization in rups rate, For the workloaus we inspected, LFO unitation in-creases by as much as 55% without coalescing (Figure 12(d)), creates by do react as 50.00 to month content of the generation while under even the minimum amount of coalescing, request more mass tree on entrances of concerning, request latency increases by as much as  $10\times$  for small requests, due samency mencanes by as maken as 10% for small lequence, one to large timeouts. Interrupt coalescing is disabled by default in Linux, and real deployments use alternatives (§2). This paper addresses the challenge of dealing with expo-

nemially increasing interrupt rates without sacrificing latency, We initially implemented adaptive coalescing for NVMe, a dy we unreastly superiorized analytic contracting for it wave, a ty-namic, device-side-only approach that tries to adjust batching based on the workload, but find that it still adds unnecess sarry latency to requests (§3.2). This led to our core insight that device-side heuristics, such as our adaptive coalescing tion to statistic tension to our magnetic tension of a chemic cannot achieve optimal latency because the device scheme, cannot active opumul takency recause the ovvice lacks the semantic context to infer the requester's intent; is the request latency sensitive or part of a series of asynchronous request tarency-sensitive or pair or a series or asynchronous requests that the requester completes in parallel? Sending this responses that the responses subgroups are provided by the semantic gap and vital information to the device bridges the semantic gap and enables the device to interrupt the requester when appropriate. We call this technique calibrating interrupts (or simply, we can use technique tomorrising the bits to requests sent to cinterrupts), achieved by adding two bits to requests sent to concernances, above you assuing two trues to requests sent to the device. With calibrated interrupts, hardware and software an user of interrupt generation and avoid interrupt storms while still delivering completions in a timely manner (§3). Because cinterrupts modifies how storage devices generate According it requires modifications to the device. interrupes, supporting a requires mountcausin to use use vie. However, these are minimal changes that would only require a firmware change in most devices. We build an emulator a introvence change in most devices. The source day estimation for cinterrupts in Linux 5.0.8, where requests run on real NVMe hardware, but hardware interrupts are emulated by

Cinterrupts is only as good as the semantics that are sent to the device. We show that the Linux kernel can naturally to the overse, we show that the Listens serves can naturally annotate all I/O requests with default calibrations, simply by inspecting the system call that originated the I/O request by more some system call to expose a system call (§4.1). We also modify the kernel to expose a system call (3+3), we and money no writes to exprace a synem van interface that allows applications to override these defaults. In microbenchmarks, cinterrupts matches the latency of in incronencontext structures matches the same system state-of-the-art interrupt-driven approaches while spending <sup>2</sup>To calibrate: to adjust precisely for a particular function [53].



Lack of semantics when to generate an interrupt is a problem for high-speed I/O devices

Cinterrupts closes this Semantic Gap for NVMe devices with Urgent and Barrier flags

Cinterrupts improves workloads performance even in a dynamic environment

#### Thank You!

Have questions? Send us an email. Amy Tai: taiam@vmware.com Igor Smolyar: igors@cs.technion.ac.il Michael Wei: mwei@vmware.com Dan Tsafrir: dan@cs.technion.ac.il