IODA: A Host/Device Co-Design for Strong Predictability Contract on Modern Flash Storage

Huaicheng Li☆*, Martin L. Putra☆, Ronald Shi☆, Xing Lin*, Gregory R. Ganger*, Haryadi S. Gunawi ☆

The 28th ACM Symposium on Operating Systems Principles (SOSP’21)

☆University of Chicago, *Carnegie Mellon University, *NetApp
IODA: A Host/Device Co-Design for Strong Predictability Contract on Modern Flash Storage

“IODA close to ideal!”

“Small but powerful”
“Attack of GC” – Unpredictability in SSDs

Host

Controller

NAND Flash

SSD

Flash Firmware

Garbage Collection (GC)

100us

100x slower due to GC

10ms

GC is Invisible to the Host
“The Tail Menace” in Flash Arrays

Host

RAID

Better

SSD0 SSD1 SSD2 SSD3

Percentage

100us

R = RAID

R₀
“The Tail Menace” in Flash Arrays

A slow SSD makes the entire flash array slow!
“A New Hope” – NVMe Predictable Latency Mode

**NVMe Predictable Latency Mode (PLM)**

- Predictable/Busy *Time Window (TW)*
- Device status *query & toggling*

**A major leap**

**But insufficient**

- *Coarse-grained* device-level predictability
- "Soft-contract" breaking predictability
- Requiring *complex* status tracking
- ...

*How to leverage NVMe PLM and enhance it for predictable latencies?*
The IODA Story

- Goal: *Tail-free* flash array system on top of *slightly-extended* PLM interface

- Design Principles:
  - *Simple* policies for efficiency
  - *Minimal changes* for easy deployment

- IODA Approach/Techniques:
  - *Per-I/O* latency predictability
  - *Busy Remaining Time (BRT)* Exposure
  - *Time Window (TW)* Formulation
  - An end-to-end design exploiting above extensions
Background & Motivation

IODA Overview

IODA Design
- Predictable latency flagged I/Os
- Busy remaining time
- Time window formulation
- Relaxed TW for better write amplification

Evaluation

Summary
When to issue the parity reads?

(1) **Wait** for timeout → Best threshold? *Tricky*

(2) **Always** Proactive (always send full-stripe) → *Inefficient*

Semantic gap between the Host and SSD to communicate the “busyness”
IOD₁: Predictable Latency Flagged I/Os

“Fail-if-Slow”: the SSD should fast-fail an I/O if it contends with GC

“Seems your submission targets a crowded area, early-rejection!”
Host

RAID5

$\text{R}_0 \oplus \text{R}_1 \oplus \text{R}_2 \rightarrow \text{R}$

fast-fail

100us

SSD0 SSD1 SSD2 SSD3

GC

100us

~100us

VS

~10ms
The Effectiveness of “Fail-if-Slow” Interface

Cut tails up to $\sim99^{th}$ percentile
A Case Against Proactive Reconstruction

Semantic Gap: the host doesn’t know how long SSD “busyness” will last

End up waiting for the busiest SSD
Busy Remaining Time (BRT) Exposure

💡 “Fail-if-Slow”: the SSD should fast-fail an I/O if it contends with GC

💡 Piggybacking **BRT** to reconstruct data from less busy SSDs

```
flag=true
```

```
Fast-Fail
```

```
“BRT: 60ms”
```
The Effectiveness of “BRT” Interface

Can we do better?
**IODA Busy Latency Windows**

"Fail-if-Slow": the SSD should fast-fail an I/O if it contends with GC

**TW Coordination**: SSDs take turns to perform GCs

**IODA: Always Predictable Latencies!**

| SSD#0 | Busy | Predictable | Predictable | Predictable |
| SSD#1 | Predictable | Busy | Predictable | Predictable |
| SSD#2 | Predictable | Predictable | Busy | Predictable |
| SSD#3 | Predictable | Predictable | Predictable | Busy |

How long should TW be?
**IODA Time Window (TW) Formulation**

SSD free space $\geq$ User load

$TW \leq S_p / ((N_{ssd} \times B_{burst}) - B_{gc})$

$TW \leq \frac{R_p \times S_t}{(N_{ssd} \times \text{Min}(B_{pcie}, \text{Max}(\frac{N_{dwpd} \times (1 - R_p) \times S_t}{8\text{hours/day}}))) - \left(\frac{(1 - R_o) \times N_{ch} \times S_{pg} \times N_{pg}}{(t_r + t_w + 2 \times t_{cpu}) \times R_o \times N_{pg} + t_e}\right)}$

$B_{burst}$: User load

$B_{gc}$: GC reclamation speed

$S_p$: Over-provisioning space

TW Upper Bound
IODA closes the gap between Base and NoGC.

TPCC Read Latency CDF: No Tails!
More in the paper!

- IODA TW analysis
  - 6 SSD models
  - Relaxed TW
  - TW vs. WAF tradeoffs

- Implementation
  - Platforms: FEMU + OpenChannel-SSD
  - Kernel: Linux Software-RAID + NVMe

- More evaluation results
  - 9 datacenter block traces + 21 real applications
  - IODA vs. 7 State-of-the-art approaches
  - IODA on OpenChannel-SSD
  - IODA throughput and write latency
  - …
IODA Stack and Evaluation Setup

- **User**
  - Storage Workloads
  - 9 datacenter I/O traces
  - 6 FileBench Workloads
  - 15 Data Intensive Applications

- **Kernel**
  - Software-RAID
  - NVMe Driver
  - SQ CQ

- **SSDs**
  - OpenChannel-SSD
  - FEMU

**Metric:** Read tail latencies

**vs. State-of-the-art**
- Preemption
- Coordination
- Speculation
- Suspension
- Partitioning
- SLO-aware
- Tiny-Tail
IODA Evaluation

Predictable Latency Flag + Reconstruction

Predictable Latency Flag + Busy Remaining Time

Predictable Latency Flag + Time Window

IODA is close to Ideal!
IODA Results: \((95^{th} - 99.99^{th})\)

Up to 75x improvement over Base

IODA is more deterministic and efficient in cutting tail latencies!

VS.

- Preemption
- Coordination
- Speculation
- Suspension
- Partitioning
- SLO-aware
- Tiny-Tail
IODA doesn’t sacrifice the array’s aggregate bandwidth
IODA Takeaways

- A **Co-Design** Approach for Performance Predictability
  - Proactive reconstruction via *fast-fail* interface
  - *BRT* for improved latencies
  - *TW* formulation to program the window length
  - Cross-device synchronization

Thank you!

*I’m on the job market.*

IODA: [https://github.com/huaicheng/IODA](https://github.com/huaicheng/IODA)