eZNS: An Elastic Zoned Namespace for Commodity ZNS SSDs

Jaehong Min, Frank Zhao, Ming Liu, and Arvind Krishnamurthy

PAUL G. ALLEN SCHOOL OF COMPUTER SCIENCE & ENGINEERING

WISCONSIN UNIVERSITY OF WISCONSIN–MADISON

Background

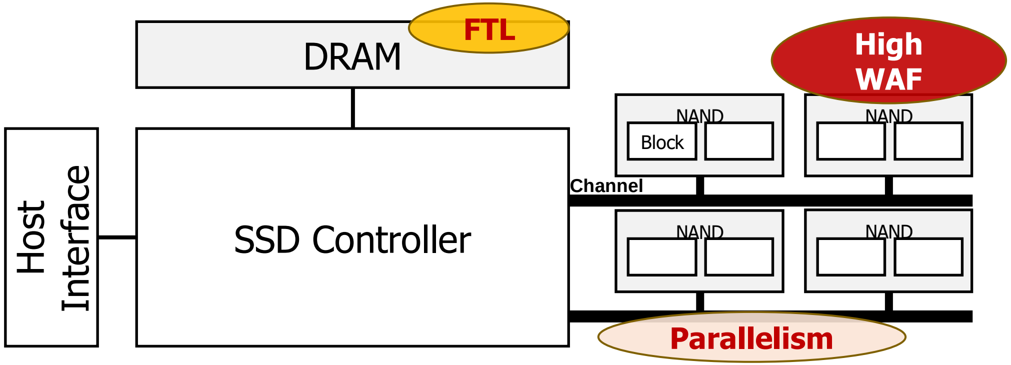
Conventional SSD Architecture

High-bandwidth with parallelism

A large DRAM to maintain FTL

Multi-tenancy incurs frequency Garbage Collection

- High WAF (Write Amplification Factor)
- I/O Interference due to the housekeeping

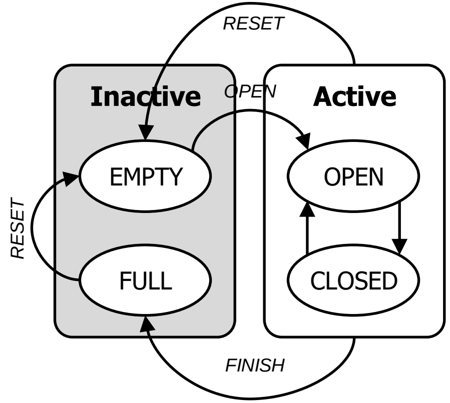


ZNS (Zoned Name Space) SSD

A point of compromise between
Open-Channel SSD and Conventional SSD

What is the ZONE?

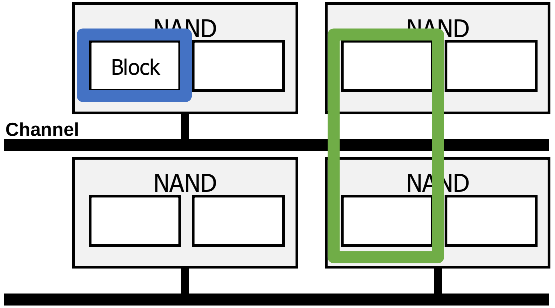
- Append-only, No random write
- Erase as a whole
- Zone is only writable in the ***Active*** states



ZONE State Diagram

Where is the zone placed?

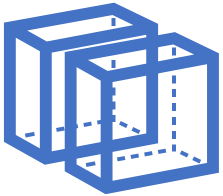
- Small-zone : A single NAND erasure block
- Large-zone : Striping across multiple blocks



ZONE Placement

‣ Focus on small-zone SSDs due to the multi-tenancy requirement

The unique features of ZNS SSD



Isolation

ZNS places data in an isolated block

No FTL, No garbage collection

Utilization

No need for over-provisioning area

No internal operations

Outline of the talk

Characterization

- Does isolated data placement imply performance isolation?
- Does ZNS deliver high performance utilization?

Our Design

- eZNS: An elastic ZNS interface
- Improve the performance in both isolation and utilization

Evaluation

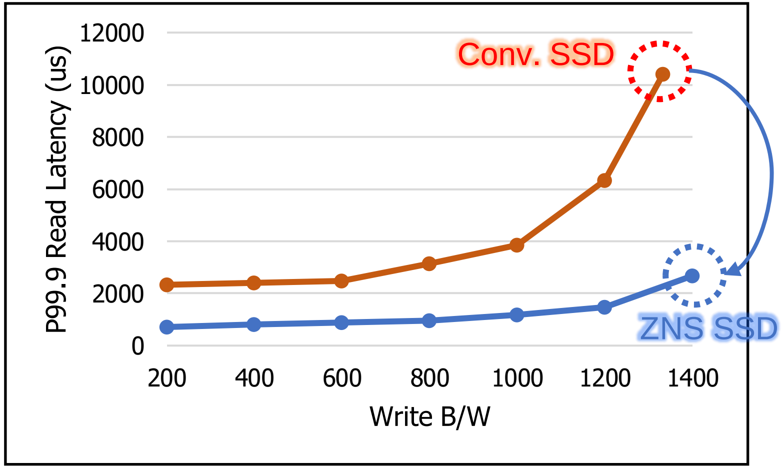
- Microbenchmarks
- RocksDB over ZenFS*

* ZNS: Avoiding the Block Interface Tax for Flash-based SSDs (ATC 21')

Anticipated Promises for Performance in ZNS

Performance Isolation

- ZNS SSD isolates write streams in a zone
- Significant improvement in read tail latency



Better tail latencies than Conv-SSD

Will the promises be upheld in real-world workloads?

Low per-zone B/W brings severe interference

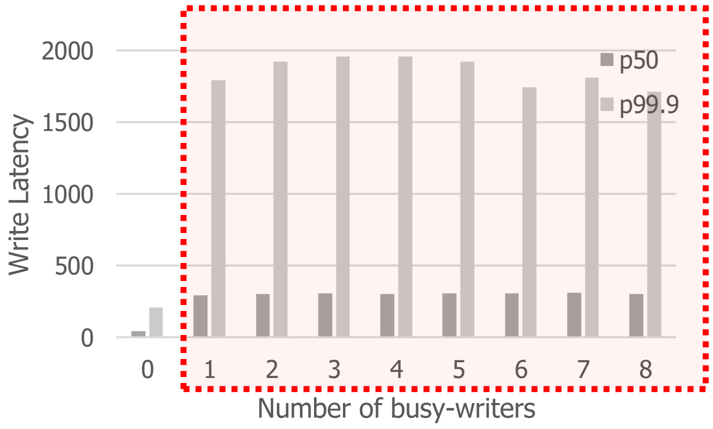
While ZNS isolates at the zone level, there could be contention at other levels of the SSD (e.g., dies and **write buffers**)

Conventional SSD

- Minimal impact before the max B/W

ZNS SSD

- A busy-writer take all write buffers

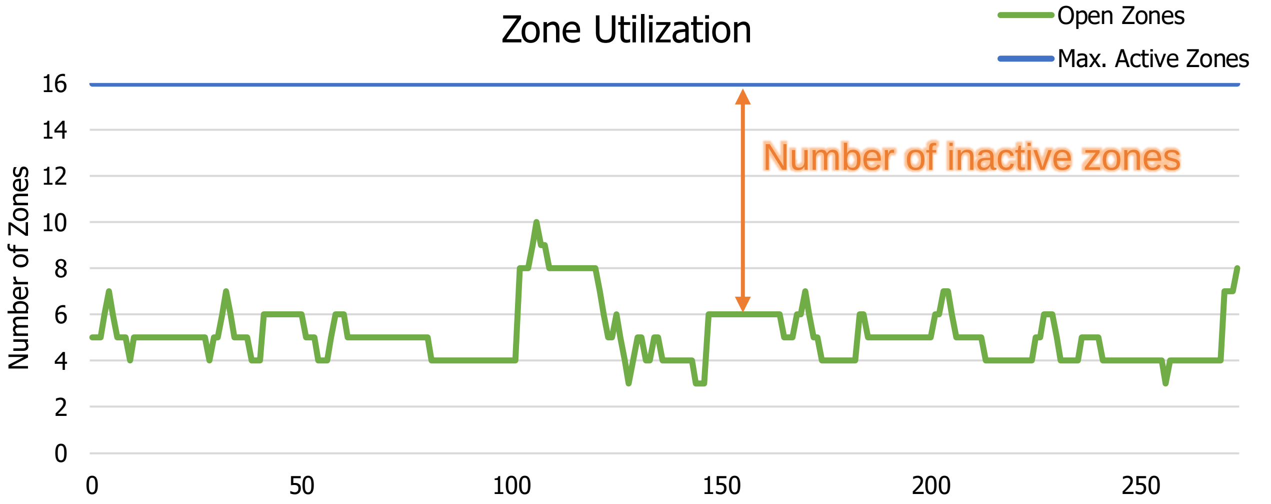


Maintaining high zone-utilization is not easy

It's challenging for applications to fully utilize active zones

- Multi-tenancy in ZNS leads to wasted or congested resources

Waste valuable active zones and yield low utilizations



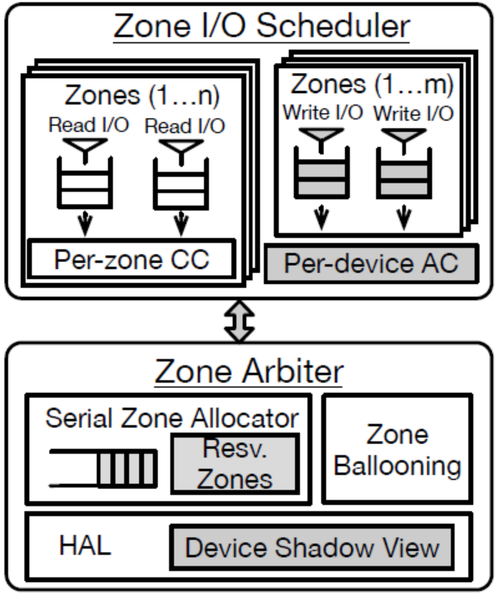
RocksDB w/ ZenFS*

Measured zone activity in 1sec window while running db_bench 'fillrandom' workload.

eZNS (Elastic ZNS)

A software layer that provides a **logical zone** abstraction

- Maximize the devices utilization in an adaptive manner
- Reduce inter-tenant interference/congestion



- **Zoned I/O scheduler** to minimize interference
  - Per-zone READ congestion control
  - Per-device WRITE admission control
- **Centralized Zone Arbiter** to maximize utilization
  - Collision-avoiding zone allocator
  - Application-aware dynamic resource manager

Challenges

➢ #1 Low performance utilization (App-agnostic zone striping)

➢ #2 I/O Interference/Congestion (Tenant-agnostic scheduling)

➢ #3 Overlapped zone allocation (Device-agnostic placement)

Proposed Solutions

✓ Logical Zone Ballooning

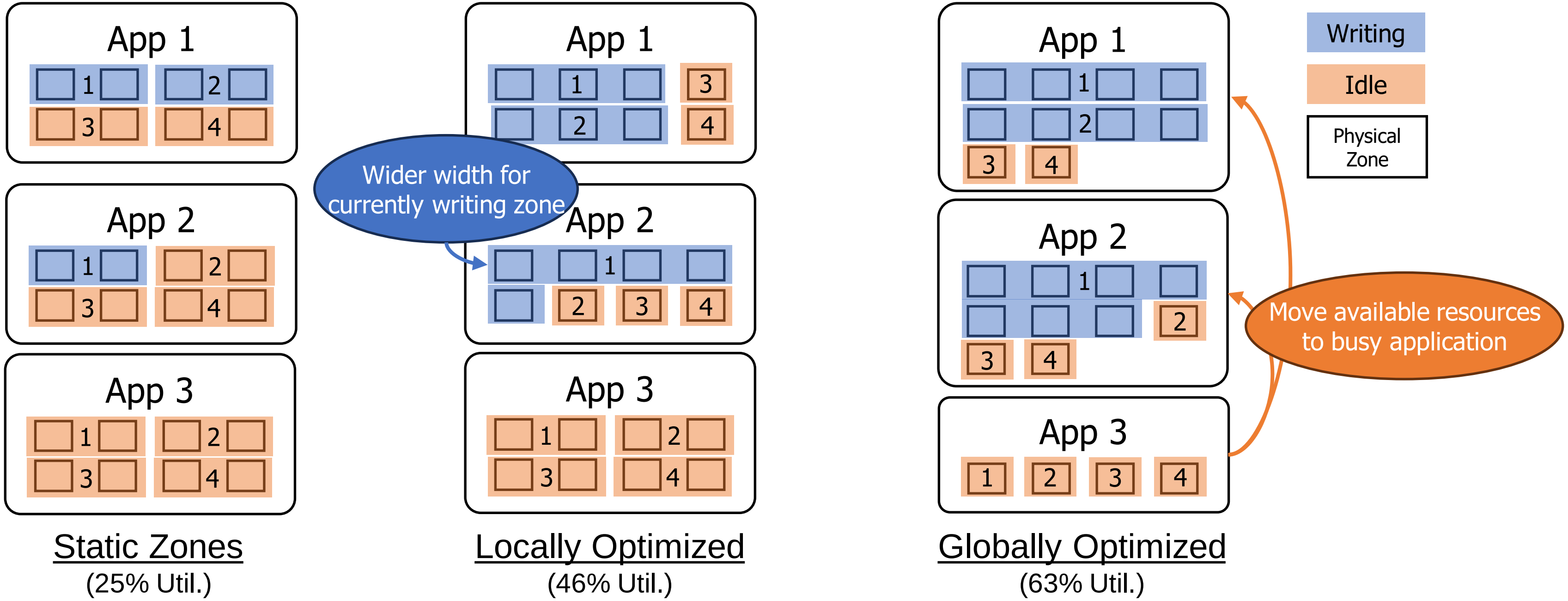
✓ Congestion/Admission Control

✓ Serial Zone Allocator

Challenge #1: App-agnostic zone striping

ZNS lacks a support for flexible interface

The optimal zone striping requires a global view



Zone Ballooning : **essentials** and **spares**

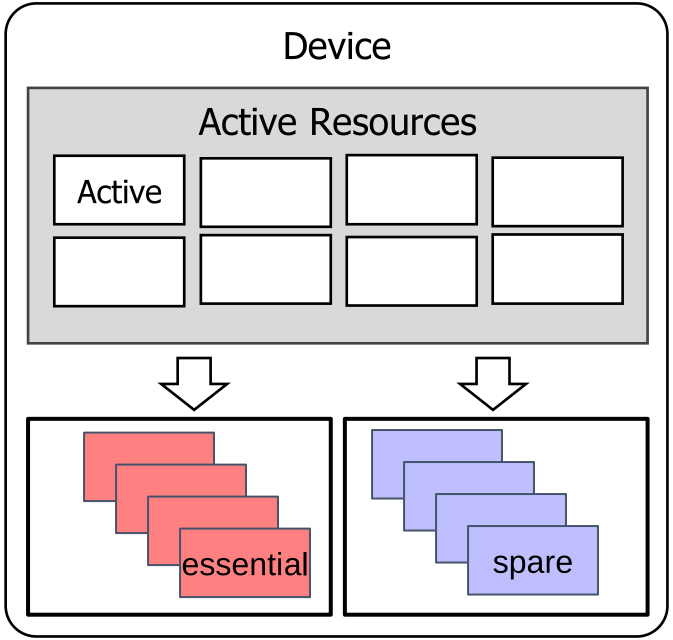
Divide active zones into two groups:

Essentials

- Exclusive resources
- Guarantee number of active zones for app
- Sufficient to achieve device utilization

Spares

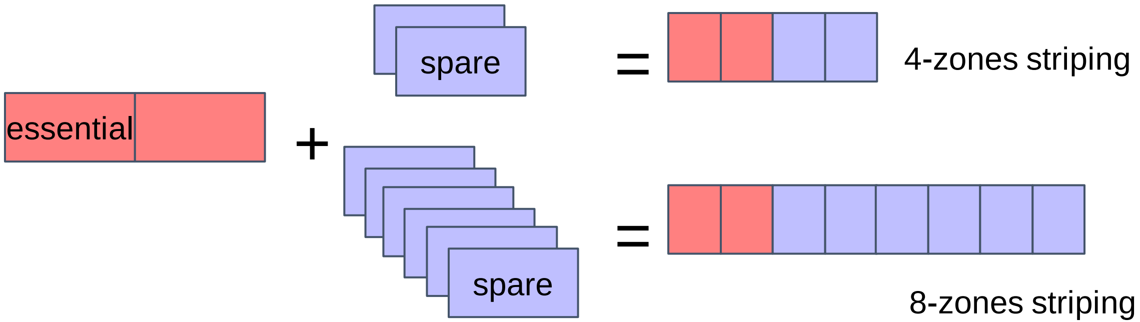
- Dynamic resources
- Temporarily boost the striping width
- Lend across namespaces (typically, apps)



Zone Ballooning: Local Overdrive

When a namespace has available spares,
a new stripe becomes an *Overdrive zone*

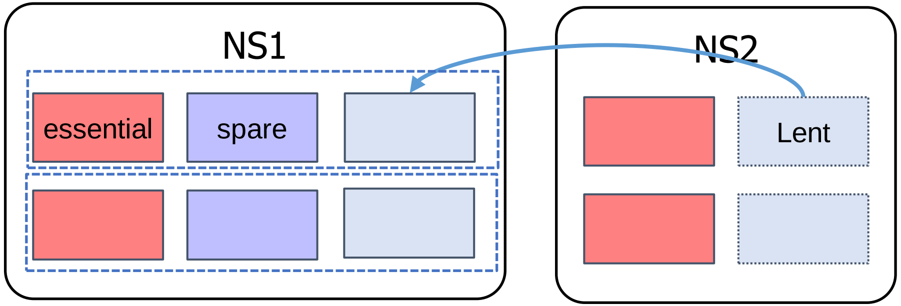
- Namespaces monitor the average number of active zones
- It widens the stripe width by adding spares to the default width



Zone Ballooning: Global Overdrive

A centralized Zone Arbiter monitors per-namespace utilization

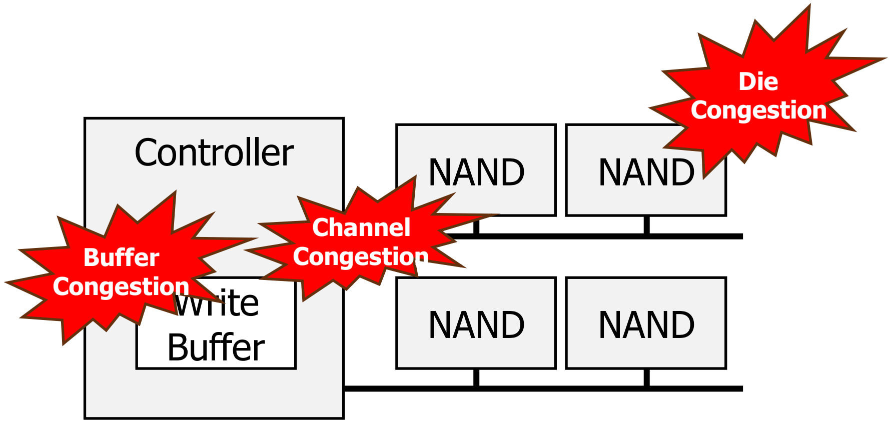
- A namespace which has no write activity is marked as “inactive”
- Redistribute unused spares in the “inactive” NS to other NS-es



Challenge #2: Tenant-agnostic scheduling

Little performance isolation and lack of fairness guarantees

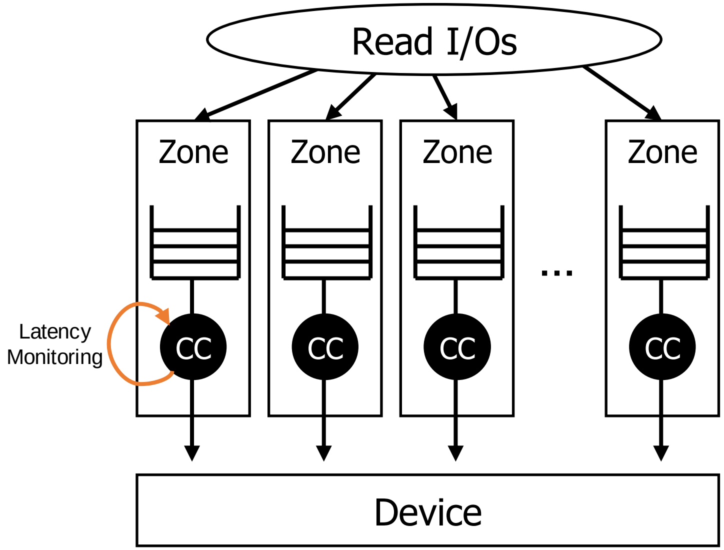
- Channel/Die congestion
- Write buffer congestion



I/O Scheduler: Per-zone Read congestion control

Delay-based CC for per-zone read scheduling

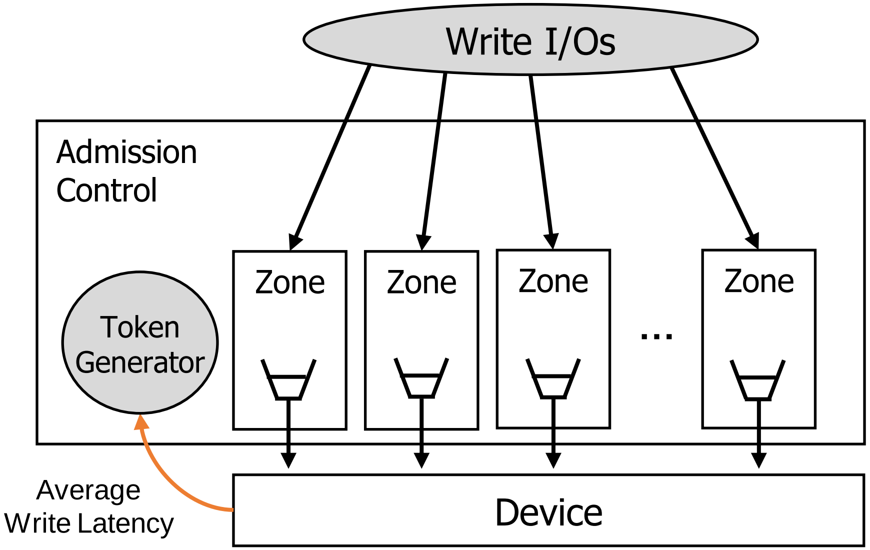
- Detect congestion via device read latency measurement within a zone
- The maximum latency threshold determines the congestion signal



I/O Scheduler: Per-device Write admission control

Write congestion occurs at the shard buffer

- The equal admission rate for all zone ensures fair resource allocation
- eZNS utilizes the average write latency to determine the admission rate



Evaluation

Evaluation Setup

eZNS is implemented as a thin layer in the SPDK framework

- Tenants connect to eZNS via NVMe over RDMA

Our testbed SSD

- Commodity Small-zone SSD

Parameters	Specification
Capacity	3,816 GB
Zone Capacity	96 MB
Maximum Active Zones	256
Number of Channels	16
Number of Dies	128 (8 dies per channel)

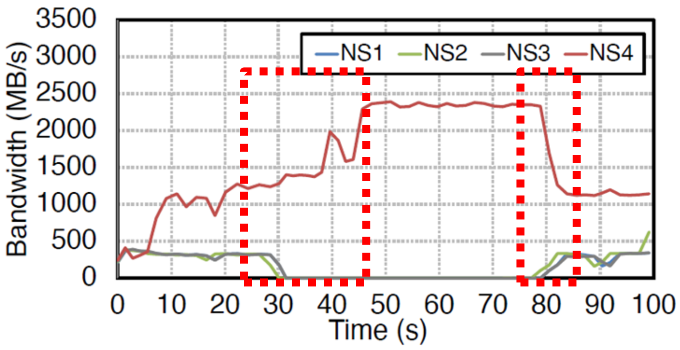
Zone Ballooning: Global Overdrive

Namespace Configuration

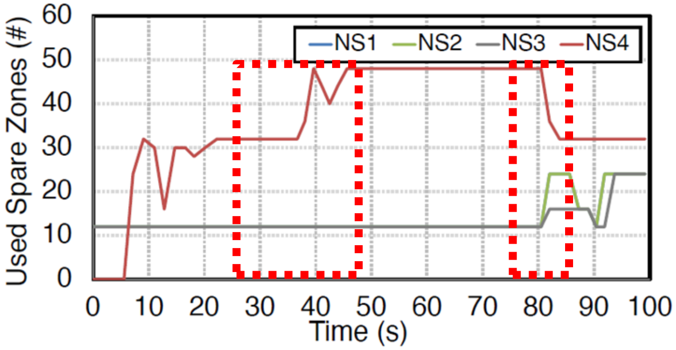
- 4 namespace with 16 active logical zones each

Moving spares boosts the write bandwidth (30~40 sec)

Lent spares are immediately returned (80 sec)



I/O Bandwidth

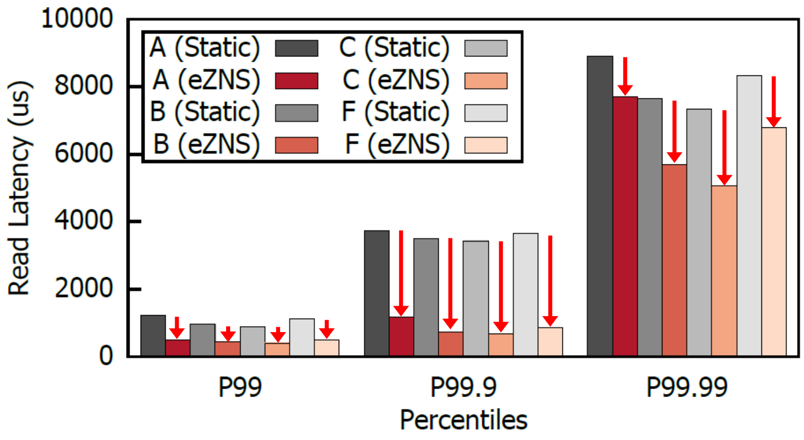


Device Utilization

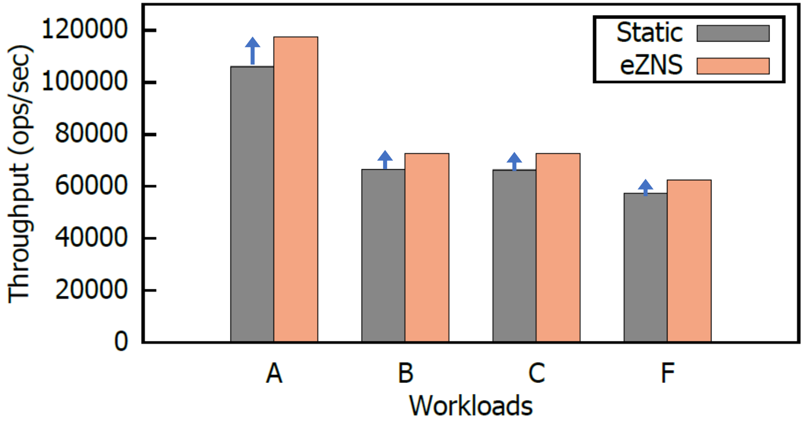
RocksDB w/ ZenFS : YCSB

eZNS improves the tail latency and throughput significantly

- YCSB workloads running on namespaces over **eZNS** and **Static-zone**
- A: Update-heavy, B: Read mostly, C: Read-only, F: Read-Modify-Write



Improve P99.9 latency by avg. 76.3%



Increase the throughput by avg. 9.5%

Summary

ZNS opens a new way of using SSDs, but has challenges

- Zone striping needs to be aware of the app characteristics and device utilization
- Zone striping must avoid overlapped allocation
- Zone incurs severe congestion due to narrower bandwidth

We design eZNS to provide an adaptive and high-performing interface

- Logical Zone Ballooning → Improves Utilization
- Read Congestion Control & Write Admission Control → Improves Isolation
- Serialized Zone Allocation → Eliminate Overlapped Allocations

eZNS significantly improves application performance in multi-tenancy

Source code will be available at https://github.com/jaehongm/eZNS soon

Thank you!