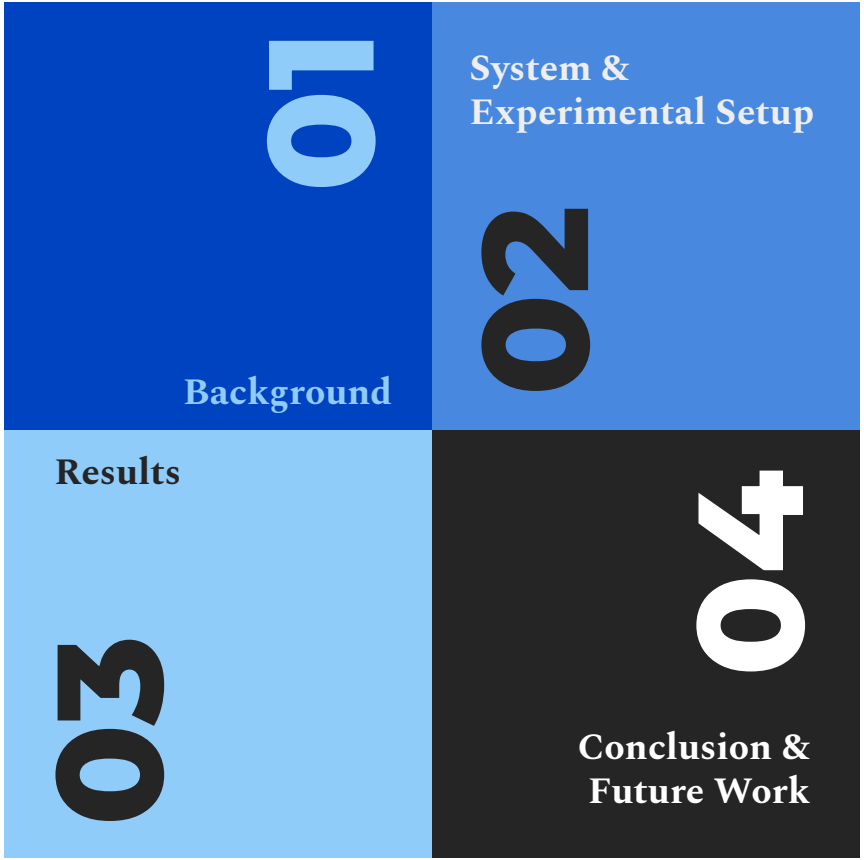
Analyzing GC and GC-free Approaches in ZNS SSDs

CS561 Spring 2026 · Final Presentation

HsiangEn (Shawna) Liu, Ruoxi (Rosalie) Cao, Zhixin(Tina) Li


01
Background
System &
Experimental Setup
02
Results
03
04
Conclusion &
Future Work

Agenda

Motivation

ZNS Storage Zone Management Problem

- ZNS SSDs expose storage as fixed-size sequential-write zones
- Hardware limits open zones to 8–32 at a time
- When limit reached, host issues FINISH → controller pads empty blocks with dummy data → DLWA
- Two competing strategies:
  - **GC-free**: finish early → no host GC, but high DLWA
  - **Host GC**: delay FINISH, mix data → lower DLWA, but GC hurts throughput

Research Question

When should the host finish a zone early (GC-free) vs delay finishing and use host-triggered GC? What are the tradeoffs in throughput, DLWA, and space amplification?

Problem & Approach

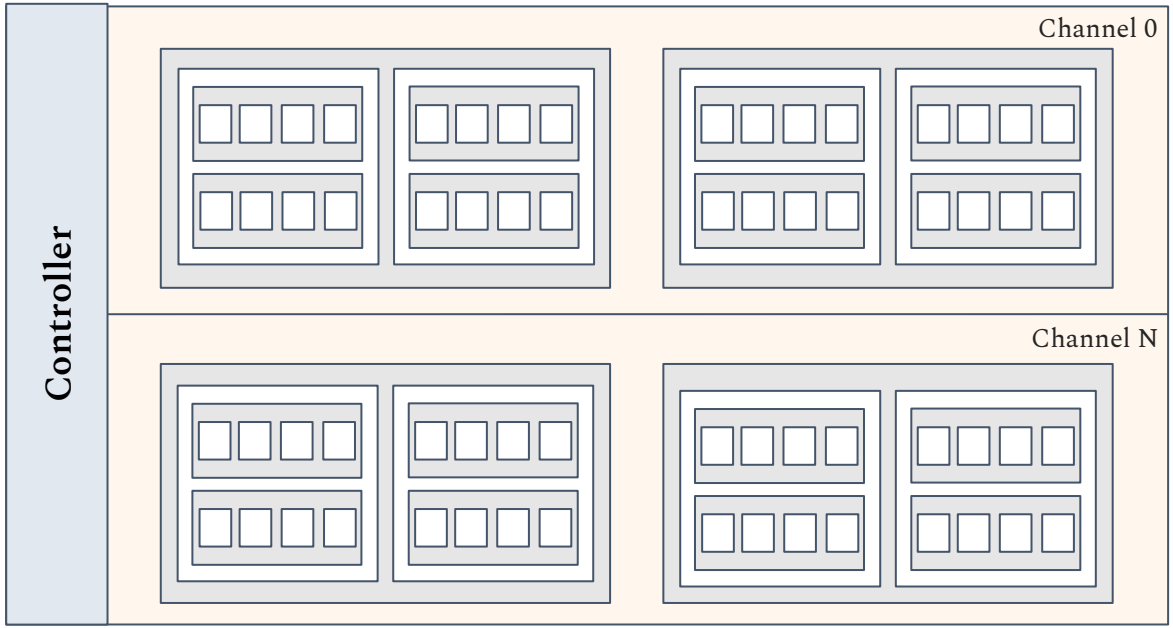
Why It Matters

- DLWA wastes SSD write endurance
- Space amplification wastes capacity
- GC overhead hurts throughput
- System designers need clear guidelines for which policy to use

Our Approach

- Vary the **occupancy threshold** (10-100%) in ZenFS/RocksDB on ConfZNS++ emulator
- Measure throughput, latency, DLWA, and space amplification across GC-free and Host GC modes

Background: SSD Architecture



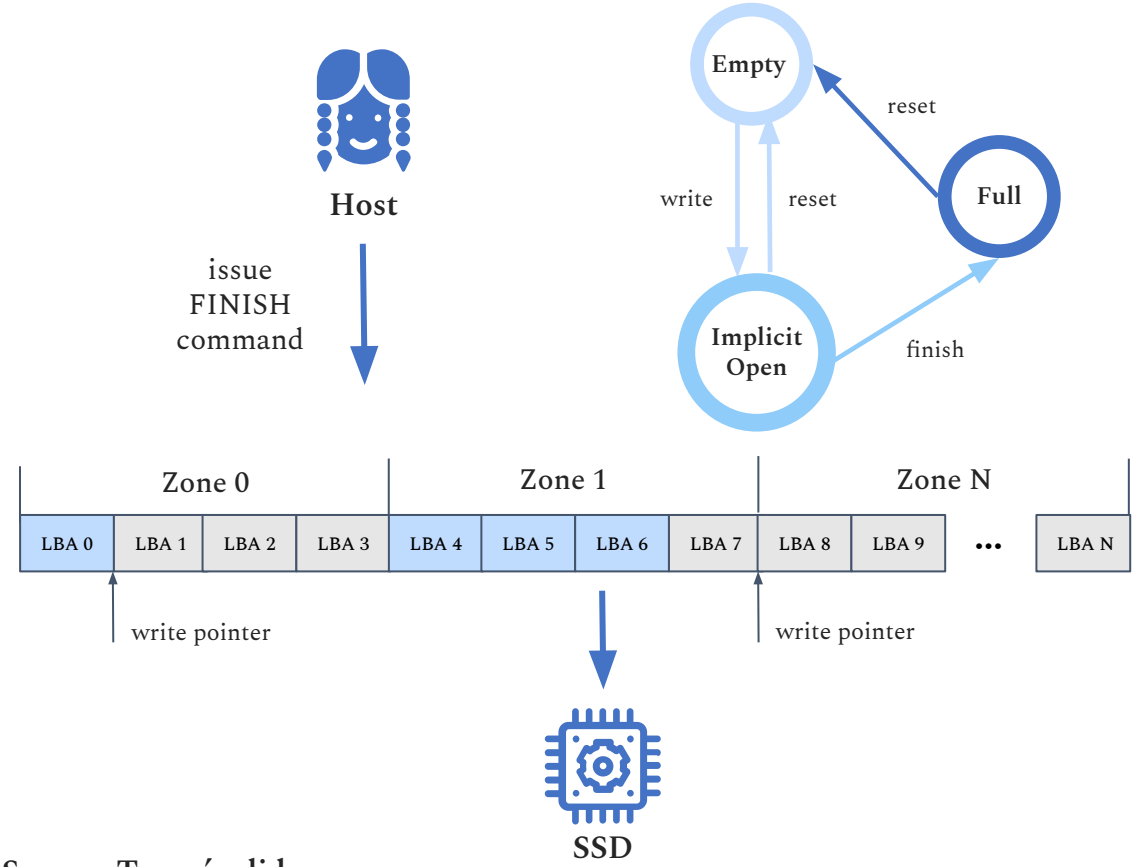
Flash Package → Chip → Die → Plane → Erase Block → Page → NAND Cell

Key Properties

- Page-level reads
  - fast random access
- Page-level writes
  - out-of-place updates only
- Block-level erase
  - must erase before rewrite
- Wear leveling
  - spread writes across blocks

Source: Teona's slides

Background: Zone Lifecycle & FINISH Command



Zone Lifecycle

- **Empty → Implicit Open**

Zone opens automatically when first write arrives

- **Implicit Open → Full**

Zone fills up naturally from sequential writes

- **FINISH command**

Host explicitly closes a zone early. Controller pads remaining empty blocks with DUMMY DATA → DLWA

- **Full/Finished → Empty**

RESET command reclaims zone for new writes

Source: Teona's slides

Background: What is ConfZNS++?

ConfZNS++ — A Configurable ZNS SSD Emulator

Built on top of FEMU (a QEMU-based storage emulator), ConfZNS++ extends previous ZNS emulators by modeling I/O management operations like FINISH and RESET as realistic device-level writes — capturing interference with host I/O that simpler emulators miss.

Realistic FINISH Model

Models FINISH command as a sequence of device-level write operations. Captures DLWA and interference with host I/O — not just a logical state change.

Configurable Geometry

Configurable channels, ways, dies, planes, pages/block, and latencies. We use: 8 channels, 2 ways, 2048 pages/block, 256MB zones.

Zone Mapping (vtable)

Flexible zone-to-physical mapping via vtable mode. zns_vtable_mode=1 enables realistic static mapping aligned to flash erase block boundaries.

Why emulator vs real HW

Real ZNS SSDs are expensive and hard to configure. ConfZNS++ lets us sweep parameters (zone size, latency, geometry) reproducibly on BU SCC cluster.

Background: What is ZenFS?

ZenFS — A ZNS-aware File System Plugin for RocksDB

Developed by Western Digital, ZenFS replaces RocksDB's default file system with one that is aware of ZNS zone constraints. It maps SST files to zones, respects sequential write requirements, and exposes key parameters for controlling zone lifecycle behavior.

How ZenFS Works

Zone allocation: Maps RocksDB SST files to ZNS zones based on data lifetime

Sequential writes: Ensures all writes follow zone sequential write constraint

Zone lifecycle: Issues OPEN, FINISH, RESET commands on behalf of RocksDB

GC integration: When enable_gc=true, triggers garbage collection to reclaim space from zones with mixed valid/invalid data

Key Parameters We Control

enable_gc

true = Host GC on
false = GC-free operation

aux_path

Path for ZenFS metadata and superblock

finish_threshold → occupancy threshold

Occupancy % below which zone is FINISHed early

Key Concept: Occupancy Threshold

Definition

ZenFS issues a FINISH command when a zone's remaining capacity falls below the occupancy threshold percentage.

From zbd_zenfs.cc: finish zone if (capacity < max_capacity * threshold / 100)

10% threshold

FINISH here

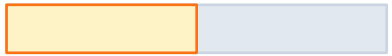
Finish when 10% full
Very high DLWA
Very low SA
GC-free operation

25% threshold

FINISH here

Finish when 25% full
High DLWA
Moderate SA
Less GC needed

50% threshold



FINISH here

Finish when 50% full
Moderate DLWA
High SA
Little GC

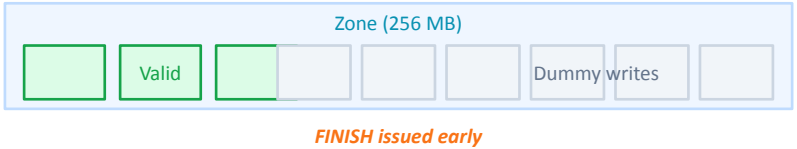
100% threshold

NEVER FINISH

Never Finish
No DLWA
Very high SA
Needs GC

GC-free vs Host GC: Tradeoff

GC-free (Low Occupancy Threshold)



FINISH issued early

+ No host-side GC overhead

+ Simple zone management

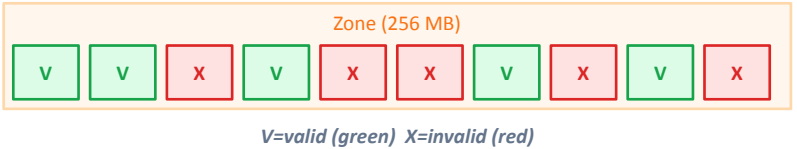
+ Better data lifetime separation

- High DLWA: controller fills empty space with dummy writes

- Wastes SSD write endurance

- More FINISH calls = more interference

Host GC (High Occupancy Threshold)



V=valid (green) X=invalid (red)

+ Lower DLWA: zones fill before close

+ Better space efficiency per zone

- Space amplification: valid+invalid data mixed in same zone

- GC must read valid data, rewrite it to new zones (overhead)

- GC competes with host I/O
  → throughput degradation

Experimental Setup

Parameters Swept

Occupancy Threshold	10% to 100%
GC Mode	GC-free vs Host GC
Zone Size	256 MB (64 zones total)
Max Active Zones	32
Max Open Zones	32
Device Size	16 GiB
Workloads	fillrandom

db_bench Configuration

Operations	3,000,000
Key size	16 bytes
Value size	1,024 bytes
Data/run	~3 GB
Compression	Disabled

SCC Cluster Setup

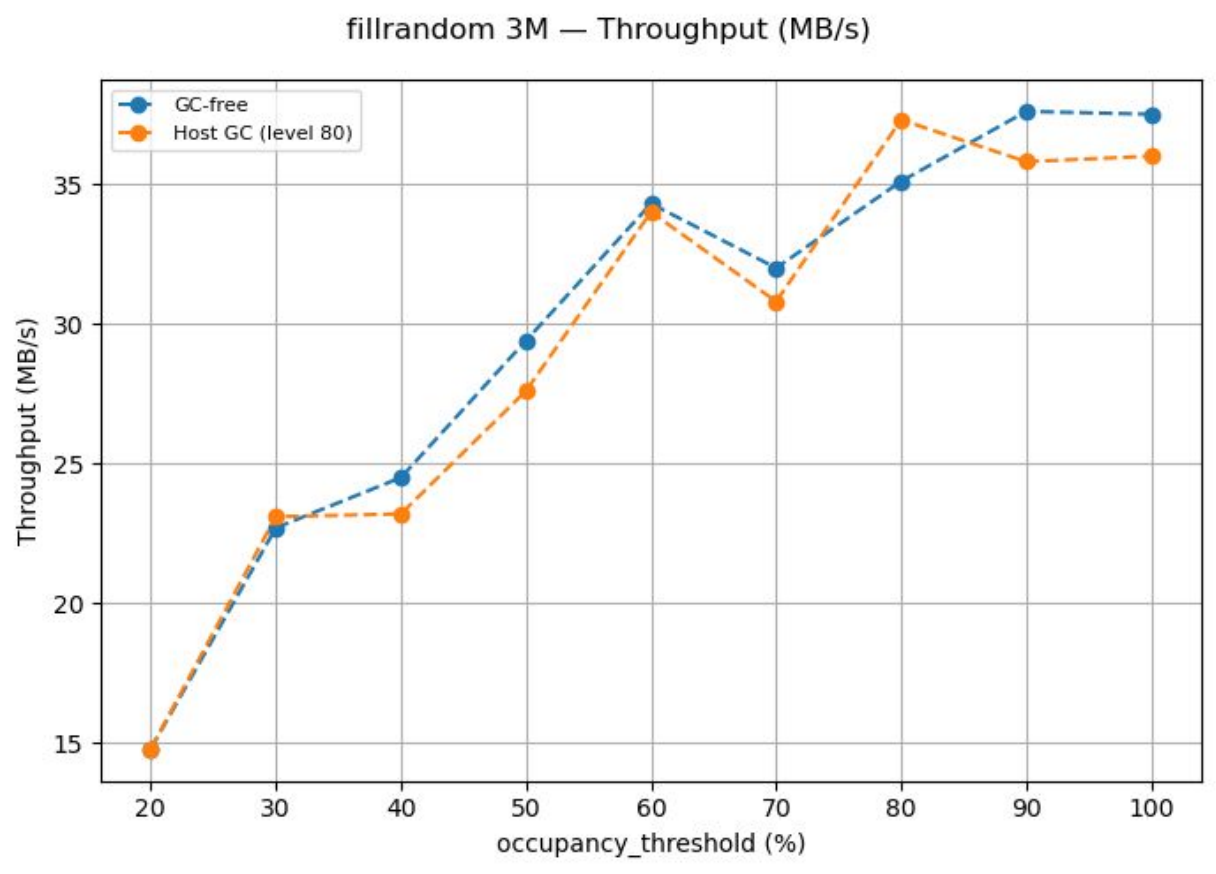
Platform: BU Shared Computing Cluster, 16 cores

VM: Ubuntu 20.04, kernel 5.11, smp=12

Emulator: ConfZNS++ via FEMU/QEMU

Batch job: qsub, 4 slots, 24h limit

Results: Throughput vs Occupancy Threshold



Key Findings

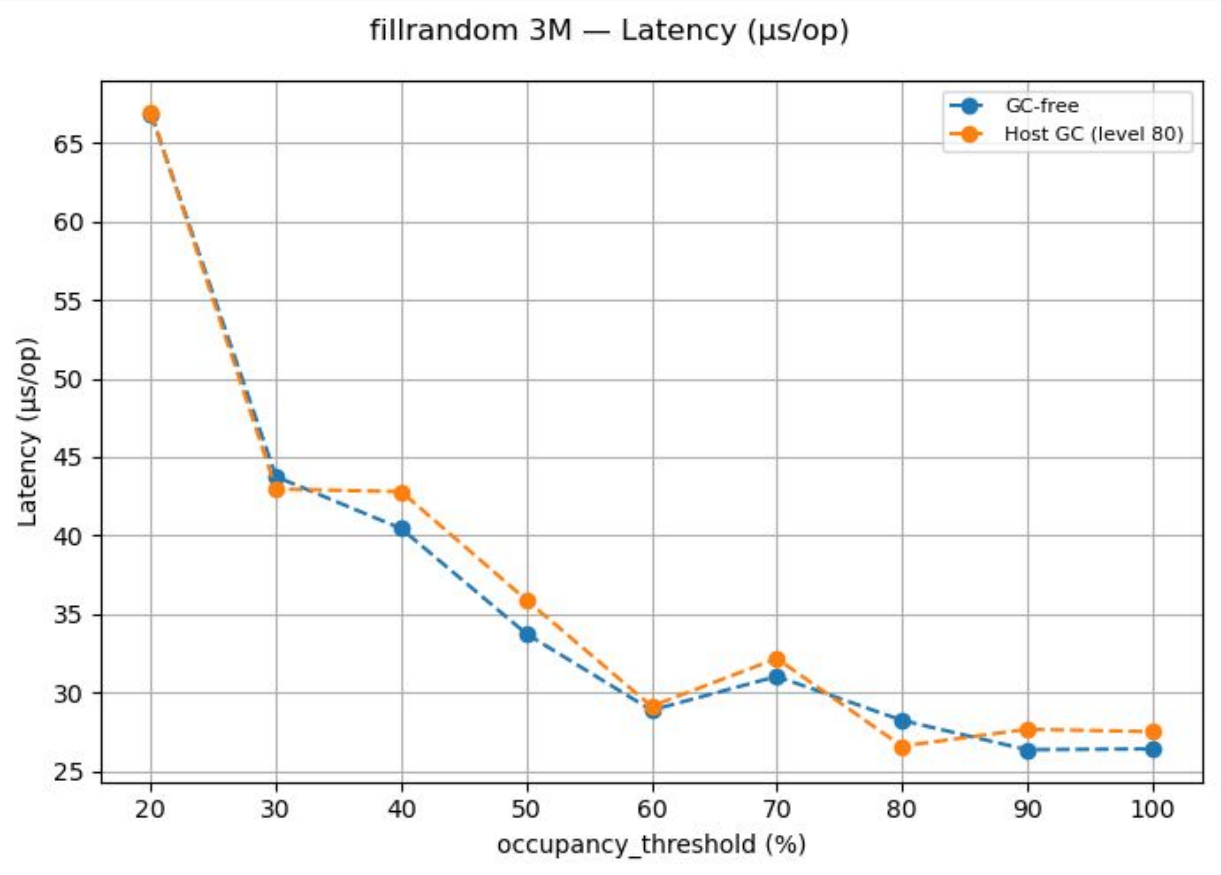
Trend — Clear upward
15 MB/s → 38 MB/s
as threshold rises

GC vs GC-free — Nearly identical
DLWA dominates
not GC overhead

Sharp increase — 70–80% threshold
critical breakpoint
for performance

Takeaway — Keep threshold
above 50% for
acceptable throughput

Results: Latency vs Occupancy Threshold



Key Findings

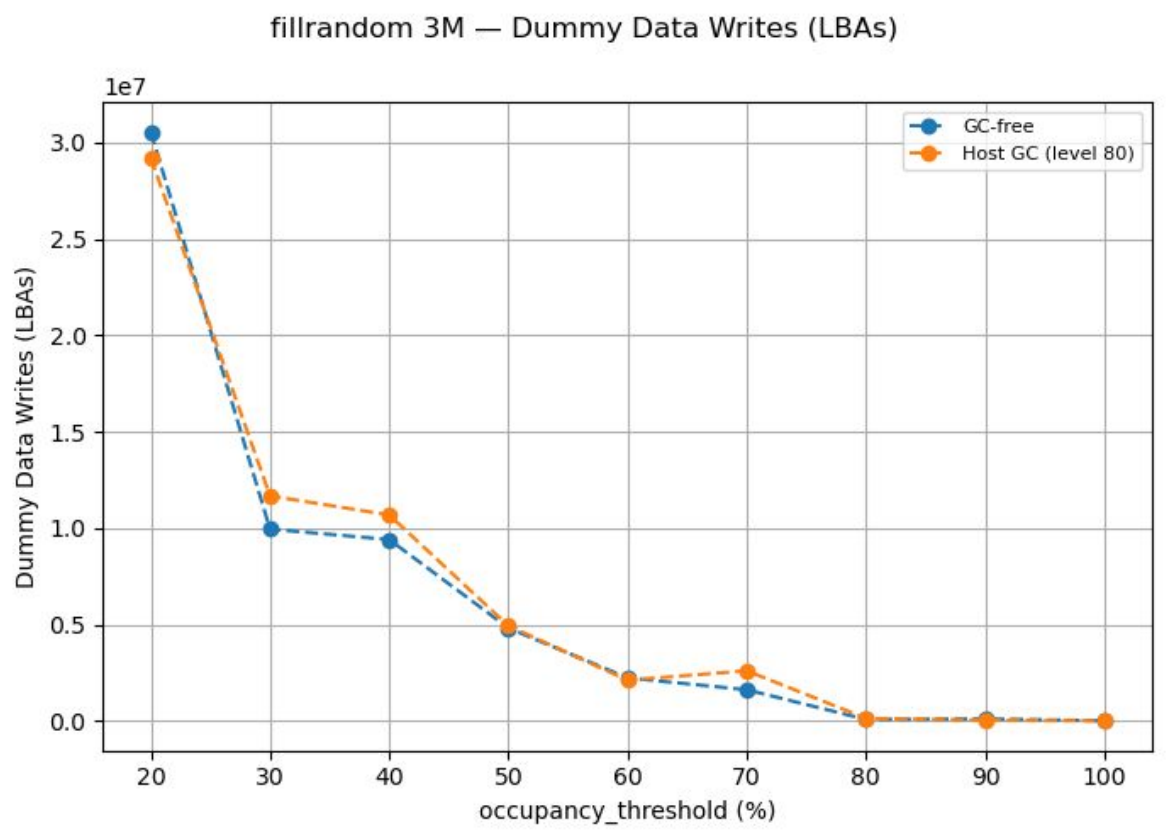
Trend — Clear downward
68 us ⟶ 28 us
mirrors throughput

Stable zone — Above 30% threshold
both strategies stay
around 40-45 us/op

GC-free — Slightly better
at high thresholds
(75%+)

Insight — Latency and throughput
both confirm threshold
is the key lever

Results: Device-Level Write Amplification (DLWA)



Key Findings

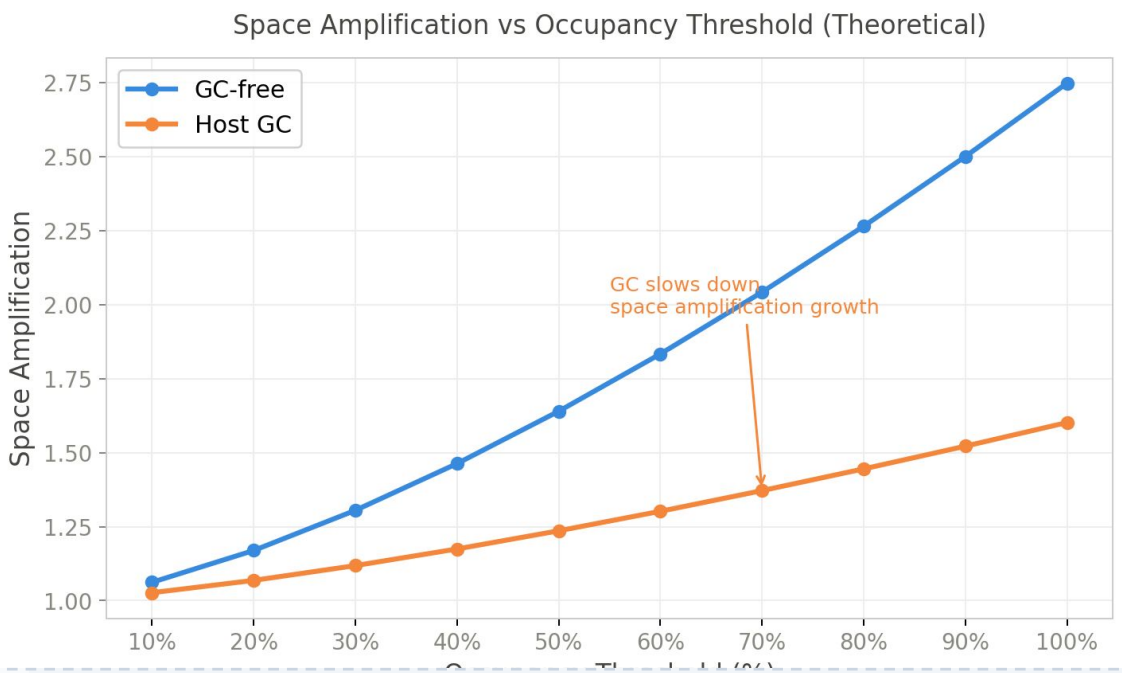
Trend — Clear downwards
3.0 LBAs → 0 LBAs at 80%

Sharp drop — Sharpest decrease at 20-30%
mirror throughput drop earlier

GC vs GC-free — nearly identical

Connection — Higher occupancy threshold
= less dummy data written

Theoretical Results: Space Amplification (SA)



What We Expect

Space Amp = disk space used / logical data size

Low occupancy threshold (GC-free):

Better lifetime separation -> lower space amplification

High occupancy threshold (GC):

Valid + invalid data mixed -> higher space amplification

Measurement: ZenFS zbd_zenfs.cc zone utilization stats

ZenFS's GC Strategy is Conservative

Trigger Condition

- GC only activates when free_percent drops below gc_start_level

Zone Selection: Full Zones Only

- GC worker iterates all zones but only visits zones where capacity == 0 (fully written)
- Open zones are never touched

Migration: Opportunistic, Not Guaranteed

- Migrated data is written into existing open zones — GC never opens a new zone
- If no suitable open zone is available, GC simply skips, even under space pressure

ZenFS's GC Strategy is Conservative

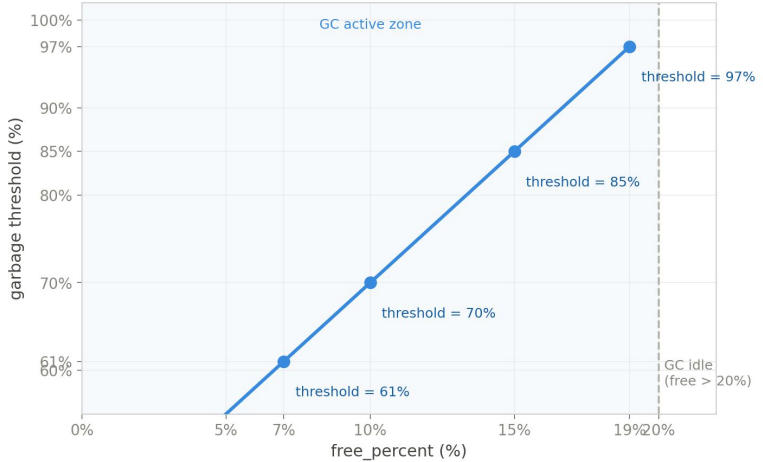
Dynamic Garbage Threshold

⚠️ At 8M ops: system hangs. At 9M ops: system crashes — GC deadlocks under space pressure.

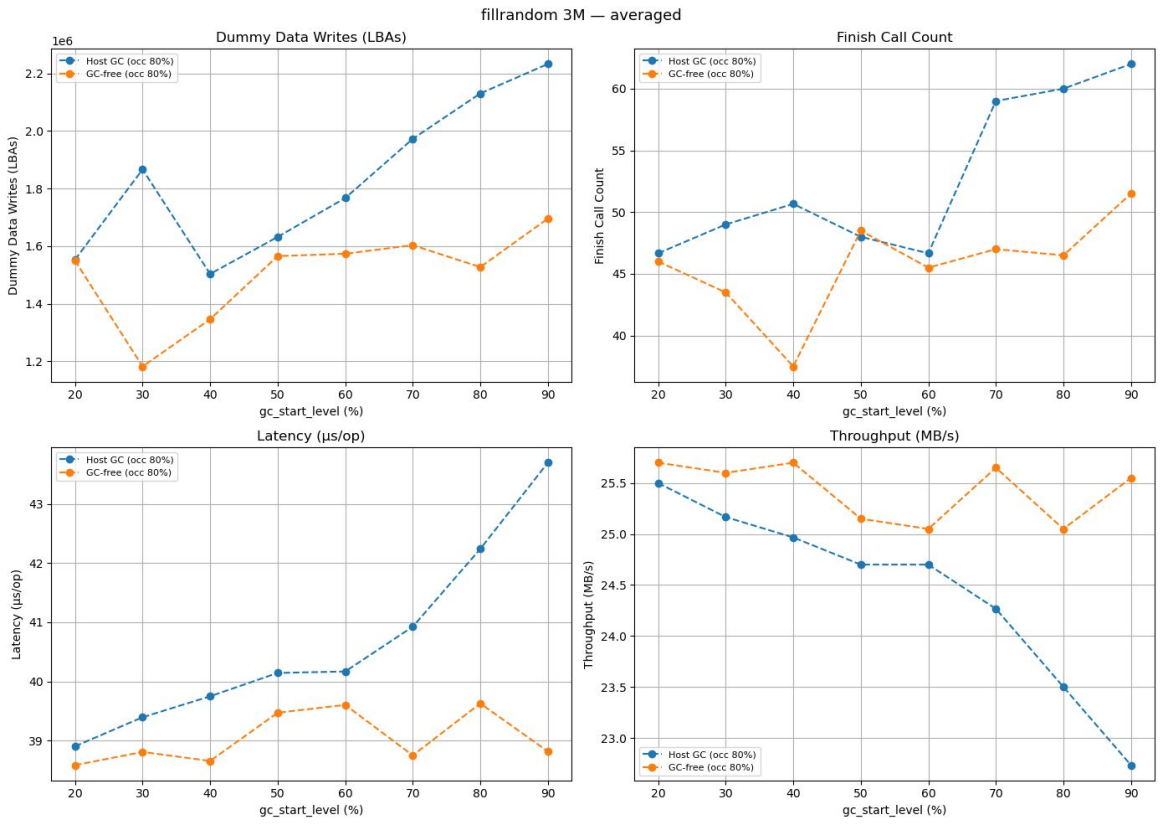
```
threshold = 100 - GC_SLOPE × (GC_START_LEVEL - free_percent)
```

As free space shrinks, threshold drops → GC becomes more aggressive, accepting zones with lower garbage ratio

free_percent	threshold
19%	97%
15%	85%
10%	70%
7%	61%



Results: GC Under Different GC Threshold



What We Expect

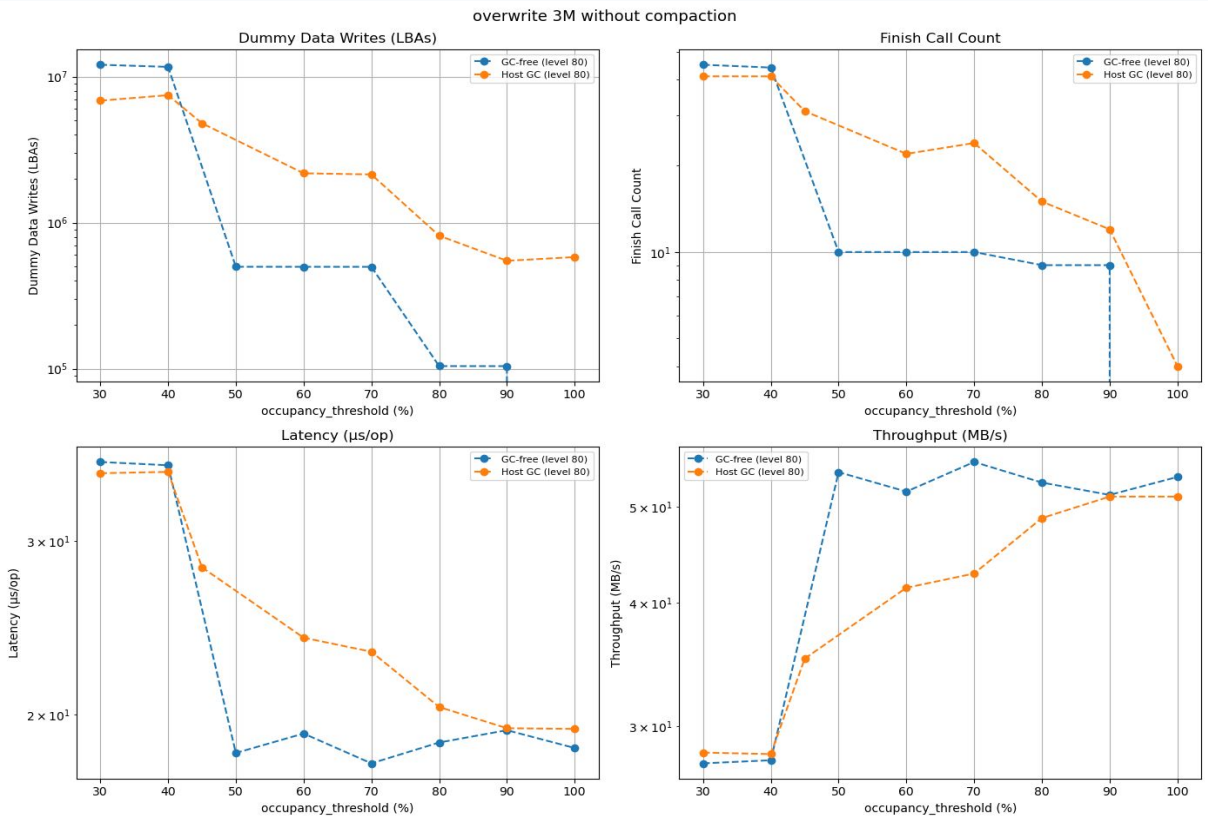
Higher gc_start_level → later, more aggressive GC → higher latency, lower throughput

Host GC → more writes into open zones → more FINISH calls → more dummy data

Host GC → lower space amplification than GC-free

GC-free baseline should remain stable across configurations

Results: GC dominates when no compaction



What We Find

- Host GC consistently higher latency — GC overhead is now visible
- Host GC shows more dummy data and more FINISH calls — active GC migration accelerates open zone fill rate
- Confirms: in fillrandom, garbage accumulation is low by nature — GC has little to reclaim, so its effect is minimal

How to Choose: Threshold & GC Strategy

Key Insight

- Threshold is the dominant lever — GC mode matters less in realistic workloads
- The right configuration depends on your workload

Decision Guide

	Write-heavy (fillrandom)	Update-heavy (overwrite)
GC mode	GC-free	Host GC
threshold	50~70%	50~70%
gc_start_level	N/A	20~30% (space-rich) / 40~50% (space-tight)

Conclusion & Future Work

Summary of Findings

- **Occupancy threshold is the dominant factor** — throughput rises from ~15 MB/s to ~35 MB/s as threshold increases from 20% to 90%
- Critical breakpoint at 50~70% threshold — below 50% DLWA dominates, above 70% space amplification grows
- In fillrandom, GC-free and Host GC perform nearly identically — garbage accumulation is low by nature, GC has little to reclaim
- In realistic deployments: **choose GC-free for write-heavy workloads, enable Host GC only when update/delete traffic is significant**

KVBench Workloads

Complete KVBench-I–V to see GC vs GC-free difference under update/delete workloads

Statistical Confidence

Run each configuration 3–5x and report mean +/- std for all metrics

Build Cost Model

Build cost model to measure the tradeoff between occupancy threshold and GC strategy

References

[1] Doekemeijer, K. et al. Exploring I/O Management Performance in ZNS with ConfZNS++. ACM SYSTOR 2024.

[2] Bjorling, M. et al. ZNS: Avoiding the block interface tax for flash-based SSDs. USENIX ATC 2021.

[3] Song, I. et al. ConfZNS: A novel emulator for exploring design space of ZNS SSDs. ACM SYSTOR 2023.

[4] RocksDB ZenFS Plugin - github.com/westerndigitalcorporation/zenfs

[5] K-V Workload Generator - github.com/BU-DiSC/K-V-Workload-Generator