计算机研究与发展
Journal of Computer Research and Development
https://crad.ict.ac.cn
63(7): 1763−1786, 2026

基于 ZNS 固态硬盘的存储优化技术综述

朱春晖[1,2] 陶秋铭[1] 周启明[1] 李明树[1]
[1](基础软件国家工程研究中心(中国科学院软件研究所) 北京 100190)
[2](中国科学院大学 北京 100049)
(chunhui2016@iscas.ac.cn)

Survey of Storage Optimization Techniques for ZNS SSDs

Zhu Chunhui[1,2], Tao Qiuming[1], Zhou Qiming[1], and Li Mingshu[1]
[1] (*National Engineering Research Center of Fundamental Software* (*Institute of Software*, *Chinese Academy of Sciences*), *Beijing* 100190)
[2] (*University of Chinese Academy of Sciences*, *Beijing* 100049)

Abstract Zoned namespaces solid-state drive (ZNS SSD) mitigates the performance overhead of the conventional "block interface tax," opening new opportunities for high-performance storage system design. At the same time, ZNS SSDs fundamentally reshape the storage model by exposing zones to the host, enforcing sequential writes within each zone, and offloading flash-management functions—such as garbage collection and address mapping—from the device to host software. These shifts introduce significant challenges for ZNS-based systems. This survey distills three key challenges: 1) write amplification induced by host-managed placement and reclamation, where co-locating mixed-lifetime data within zones increase GC copying, compaction-GC mismatch leads to redundant data migration, and inefficient zone-reset policies cause unnecessary rewrites; 2) limited parallelism under fixed zone/resource mapping and sequential-write constraints, further exacerbated by intra- and inter-zone contention; and 3) elevated consistency and reliability overheads arising from wear imbalance, crash-recovery metadata costs, and parity maintenance in multi-device settings. For each challenge, we organize state-of-the-art optimization techniques, summarize recent advances, and highlight representative studies. We also review widely used simulators/emulators and auxiliary tools that support ZNS prototyping, debugging, and performance tuning. Finally, we discuss promising research directions, including cross-layer hardware-software co-design, multi-level heterogeneous integration, systematic software-stack refinement, and holistic, system-wide optimization.

Key words ZNS SSD; novel storage hardware; write amplification; parallelism; data consistency and reliability

摘 要 ZNS 固态硬盘(zoned namespaces solid-state drive, ZNS SSD)减少了传统固态硬盘的"块接口税"所带来的性能开销,为构建高性能存储系统带来了新的机遇。然而,由于 ZNS 固态硬盘将垃圾回收、地址映射等操作上移至主机端,以及区域划分、顺序写入等存储范式的变化,导致基于 ZNS 固态硬盘的存储系统也面临诸多挑战。首先,系统地归纳了基于 ZNS 固态硬盘的存储系统所面临的 3 个核心挑战:写入放大、并行性和数据一致与可靠性。然后,系统地阐述了针对各类挑战的存储优化技术、研究进展和典型研究,并介绍了面向 ZNS 固态硬盘的模拟仿真器和性能优化辅助工具。最后,展望了基于 ZNS 固态硬盘的存储系统优化技术的未来研究方向,包括跨层协同设计、多层次异构集成、软件堆栈优化、系统全局优化。

关键词 ZNS 固态硬盘;新型存储硬件;写入放大;并行性;数据一致与可靠性

收稿日期:2025-09-10;**修回日期**:2026-02-04
基金项目:国家自然科学基金项目(62372438);北京市自然科学基金项目(4252024)
This work was supported by the National Natural Science Foundation of China (62372438) and the Beijing Natural Science Foundation (4252024).
通信作者:陶秋铭(qiuming@iscas.ac.cn)

中图法分类号 TP333.2

DOI: 10.7544/issn1000-1239.202550670 CSTR: 32373.14.issn1000-1239.202550670

ZNS 固态硬盘(zoned namespaces solid-state drive, ZNS SSD)将闪存地址空间划分为若干区域(zone),在每个区域进行顺序写入操作,并将传统固态硬盘中闪存转换层(flash translation layer, FTL)所管理的地址映射、垃圾回收等操作上移至主机端[1],有效减少了传统固态硬盘因地址映射、随机写入和过度空间预留(over-provisioning, OP)而产生的“块接口税”[2]。目前,ZNS 规范已被正式纳入非易失性存储器高速互连规范(non-volatile memory express, NVMe)体系[3-5],并已出现多款遵循 ZNS 规范的企业级固态硬盘产品[6-9]。在云计算、数据中心、高性能数据库等密集型工作负载场景中展现出良好的性能和应用潜力,受到学术界与工业界的广泛关注[10-14]。

然而,ZNS 固态硬盘存储结构与管理方式的革新也对基于 ZNS 固态硬盘的存储系统带来新的挑战。1)主机端的数据放置和垃圾回收加剧了写入放大:当不同生命周期的数据混写于同一区域时,在触发垃圾回收时将产生较大的写入放大[2,15-19];现有后台数据处理机制(如压缩(compaction))与垃圾回收之间缺乏协同,导致数据重复迁移[18,20-27];不恰当的区域重置策略(过频、过早或选区不当)亦会增加数据迁移与写入放大[28-34]。2)资源映射机制和顺序写约束限制了系统的并行性:固定区域大小与静态资源映射限制了在动态负载下的系统并行性[29-30,35-39];在顺序写入约束下,区域内的并行写入将产生锁争用与多线程串行[40-42];由于区域间操作的干扰与阻塞将导致系统性能波动[10,43-49]。3)现有的数据放置、同步与校验规则增大了维护数据一致和可靠性的开销:主机端的数据放置策略可能造成介质磨损不均,缩短了设备寿命[33,47,50];频繁的元数据同步将导致系统崩溃恢复的开销增大[51-52];在由多个 ZNS 固态硬盘组成的独立硬盘冗余阵列(redundant array of independent disks, RAID)中,顺序写操作与跨盘一致性使校验块频繁更新,产生“部分奇偶校验税”,导致一致性维护开销增大[39,41,53-55]。

针对上述写入放大、系统并行性以及数据一致和可靠性等问题,国内外研究者从不同角度对基于 ZNS 固态硬盘的存储系统的优化技术进行了大量的研究和探索,为构建基于 ZNS 固态硬盘的高性能存储系统提供了基础。

本文对近年来基于 ZNS 固态硬盘的存储系统优化技术前沿研究进行了系统调研和归纳分析,对该方向的研究趋势进行了预测,主要贡献有 3 个方面:

1)立足于 ZNS 固态硬盘的存储范式(paradigm)和硬件特性,分析和阐述了基于 ZNS 固态硬盘的存储系统所面临的 3 个核心挑战:写入放大、并行性和数据一致与可靠性。

2)系统地介绍了针对 3 个核心挑战的存储优化技术、研究进展和典型研究,并对现有的面向 ZNS 固态硬盘的模拟仿真器和性能优化辅助工具进行了总结归纳。

3)对基于 ZNS 固态硬盘的存储系统优化技术的未来研究方向进行了展望,指出跨层协同设计、多层次异构集成、软件堆栈优化和系统全局优化等是后续亟待深入研究的重要方向。

1 背景介绍

ZNS 固态硬盘在继承了 OC 固态硬盘(open-channel solid-state drive, OC SSD)的主机端介质管理机制和 SMR 硬盘(shingled magnetic recording hard disk drive, SMR HDD)的顺序写入和分区管理等存储范式的基础上,进一步优化了传统存储设备的介质管理复杂度过高、系统资源占用过大以及机械延迟高等问题,从而更好地提升了系统性能[1,13-14,56-65]。

在 ZNS 固态硬盘中,控制器将整个命名空间划分为若干区域(zone)。其中,每个区域由若干块(block)组成,块再细分为若干页(page),页则是闪存介质的最小写入/读取单元,常见大小为 4~16 KB,这种层次化的设计使区域边界与闪存擦除单位对齐,从而避免了跨块所产生的性能开销,为了更有效地提升并行性能,控制器通过条带化(striping)设计将同一区域分布到多个闪存芯片(chip)和通道(channel)上,ZNS 固态硬盘的基本结构如图 1 所示。此外,根据单个区域容量的规模,可以将 ZNS 固态硬盘划分为大区域(large-zone)固态硬盘和小区域(small-zone)固态硬盘两大类,前者更注重顺序吞吐量,后者则强调并发性和空间管理的灵活性。

ZNS 规范的核心语义包括顺序写入和随机读取。其中,顺序写入又分为区域写入(zone write)和区域追加(zone append)两种形式。在区域写入时,主机端需指定块地址以保证写入偏移量与设备端写入指针

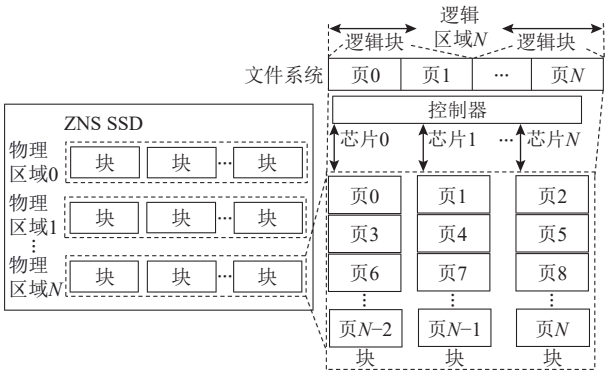


Fig. 1 ZNS SSD architecture[66]

图 1 ZNS 固态硬盘架构[66]

相匹配，主机端通常每次只向同一区域发出 1 个写请求，从而限制了区域写入的并行性。相对而言，区域追加则仅需主机端指定将要写入的区域，即可向该区域发出多个写追加，从而降低了在同一区域内的写指针竞争。同时，随机读取则无需关注区域状态（离线状态除外），若区域未触发重置，即可进行读取。ZNS 固态硬盘通过以上核心语义有效地支持了系统并行读写操作[58,67]。

尽管顺序写入方式有效提升了系统性能，但在元数据更新、日志记录等负载场景下，随机写入对系统延迟的影响更低，因此，ZNS 规范最新引入了 ZNS 随机写入区域（ZNS random write area, ZRWA）语义，允许主机端对 ZNS 固态硬盘中特定区域进行随机写入操作，从而进一步提升系统性能[4-5,68]。

ZNS 固态硬盘可将区域按照数据处理的阶段分为：空、打开、关闭、满和离线等状态，如图 2 所示。当区域重置（reset）时，其状态将转换为“空（empty）”，表示该区域内不包含任何数据。当区域打开时，其状态转换为“打开（open）”，并将其与设备资源相关联。当区域关闭时，其状态则从“打开”转换为“关闭（close）”，并释放该区域所占用的资源。当区域不再接受写入操作时，则通过“完成（finish）”操作将其状态由“打开”或“关闭”转换为“满（full）”。此外，当区域发生错误时，其状态将设置为“离线（offline）”。根据区域状态及管理操作，主机端能够有效识别区域所处阶段，从而执行资源调度和回收等策略，实现对资源的高效利用与动态管理[37,69-72]。

由于 ZNS 固态硬盘将设备中 FTL（flash translation layer）的地址映射、垃圾回收等操作移至主机端，按照存储堆栈路径可将基于 ZNS 固态硬盘的存储系统分为 3 种类型：主机端 FTL、支持 ZNS 接口的内核文件系统和区域感知的用户态文件系统，如图 3 所示。

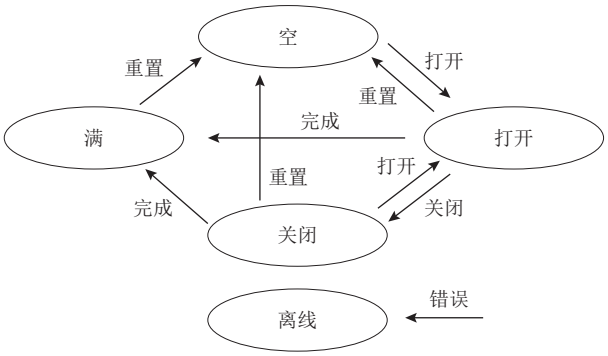


Fig. 2 ZNS SSD state[37]

图 2 ZNS 固态硬盘状态[37]

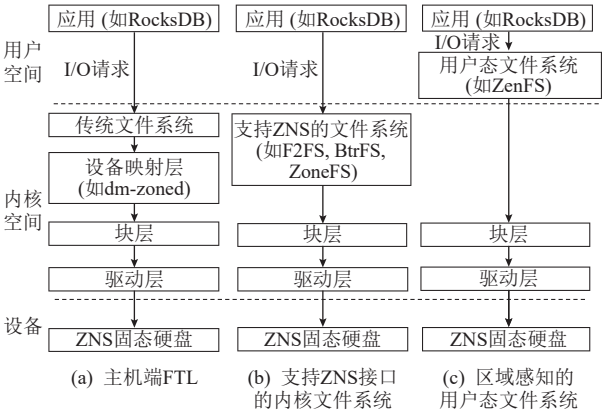


Fig. 3 Storage system with ZNS SSD[68]

图 3 基于 ZNS 固态硬盘的存储系统[68]

1）主机端 FTL。这类存储系统引入主机端 FTL 软件层（host-side flash translation layer, HFTL）来支持传统文件系统和 ZNS 固态硬盘之间的适配，在主机端实现传统固态硬盘中 FTL 的地址映射和垃圾回收等功能。在这种类型中，基于 ZNS 固态硬盘的存储堆栈主要由上层应用、传统文件系统、设备映射层（执行 FTL 操作）、块层、驱动层和设备层构成，如图 3（a）所示。内核中的设备映射层向上支持常规的块设备接口，对传统文件系统具有较好的兼容性，并承担地址映射、顺序写入等功能，避免了上层文件系统对 ZNS 语义的显式感知。例如，通过内核中设备映射层（如 dm-zoned）实现 ZNS 接口的顺序写入和区域管理等操作，使传统文件系统仍以随机写入方式访问设备，无需感知 ZNS 接口的区域限制。然而，由于在内核中引入了 FTL 层（设备映射层），增加了对系统资源的占用，并且延长了 I/O 执行路径，引入了额外的上下文切换与数据复制等操作，导致主机端的性能开销增大[59,73-76]。

2）支持 ZNS 接口的内核文件系统。这类存储系统通过改进内核文件系统的写入路径、地址空间管

理等机制达到对 ZNS 接口的支持[12,68,72,77-79]。存储堆栈主要由上层应用、支持 ZNS 接口的内核文件系统、块层、驱动层和设备层组成，如图 3(b)所示。该文件系统可感知并解析 ZNS 语义，获取区域大小、可用容量、活动区域上限、顺序写入等信息，并据此调整相应的数据布局和元数据管理。它以日志结构的文件系统为基础，如 F2FS(flash-friendly file system)[80-81]，通过改进其写入策略与区域管理机制使其匹配 ZNS 接口的顺序写入操作，实现对 ZNS 固态硬盘的支持。相较于主机端 FTL，该方式降低了对系统资源的占用，并且缩短了 I/O 执行路径，减少了主机端的性能开销。但由于现有内核文件系统一般以就地更新为基础，不支持顺序写入模型，因此，直接改造难度较大，同时，文件系统需执行元数据更新等操作，常需要随机写入，然而，由于 ZNS 接口中区域顺序写入方式无法支持直接覆盖更新元数据，因此，在高并发写入场景下，系统由于 I/O 排队等造成性能瓶颈，导致主机端的性能下降[78-79,82-84]。

3)区域感知的用户态文件系统。这类存储系统在用户态空间采用端到端的方式直接对 ZNS 固态硬盘进行管理[85-87]。其中，存储堆栈主要由上层应用、用户态文件系统、块层、驱动层和设备层构成，如图 3(c)所示。相比于主机端 FTL 和支持 ZNS 接口的内核文件系统，区域感知的用户态文件系统直接对设备进行区域管理，缩短了 I/O 执行路径，并减少了对系统资源的占用，因此，具有较好的系统性能。例如，研究人员将 ZenFS(zoned file system)[2,69,88] 作为 RocksDB 的用户态文件系统，采用部分内核旁路的方式，绕过了内核中文件系统等存储层，减少了上下文切换、数据复制所产生的性能开销，在进行数据密集型工作负载的场景下有效地提升了系统性能。但由于在主机端(上层应用和用户态文件系统)存在数据放置、垃圾回收等方面的局限性[2,15-19]，导致存储系统的性能难以进一步提升。

上述 3 种存储架构各具特点，为 ZNS 固态硬盘的发展提供了有力的支持，有效地提升了基于 ZNS 固态硬盘的存储系统的性能。

2 基于 ZNS 固态硬盘的存储系统所面临的挑战

ZNS 固态硬盘为存储系统的设计带来了巨大的机遇，但也伴随着诸多挑战。我们调研了从 2019 年至 2025 年 8 月的相关会议和期刊的论文，梳理了构建基于 ZNS 固态硬盘的高性能存储系统存在的诸多挑战。其中，最主要的挑战包括：1)写入放大(write amplification)；2)并行性(parallelism)；3)数据一致性(consistency)与可靠性(reliability)。下面依次阐述。

2.1 写入放大问题

在基于闪存的存储系统中，写入放大是设备端执行的物理写入量与主机端执行的逻辑写入量之比，理想情况下写入放大近似为 1，但在闪存设备“先擦除后写入”的物理特性下，系统常需要执行更多的有效数据写入，因此，将产生较大的写入放大，且占用大量的设备空间，导致系统性能下降。

ZNS 固态硬盘作为新型的基于闪存介质的存储设备，通过顺序写入和整区重置优化了对底层闪存介质的管理效率，同时，将原设备中地址映射和垃圾回收等操作上移至主机端，使设备端的性能开销大幅减少；但由于主机端的数据放置和垃圾回收(garbage collection, GC)等策略尚存在一定的局限性，导致在 GC 的过程中产生了较多的数据迁移[18,20,24,29-30]。写入放大已成为基于 ZNS 固态硬盘的存储系统中关键挑战之一。

具体而言，加剧基于 ZNS 固态硬盘的存储系统的写入放大问题可以划分为：不同生命周期的数据混写、压缩(compaction)与 GC 操作失调和不恰当的区域重置(zone reset)。

1)不同生命周期的数据混写。ZNS 固态硬盘通过区域数据放置和顺序写入等方式，有效降低了传统固态硬盘中随机写入所引起的性能开销，并且其写入方式与基于日志结构合并树(log-structured merge-tree, LSM-tree)的键值存储系统[87,89] 的追加写入机制非常契合，因此，基于 LSM-tree 的键值存储系统在 ZNS 固态硬盘上的性能得到了有效的提升[2,18,29,71]。然而，在主机端进行数据放置的过程中，由于将具有不同生命周期的排序字符串表(sorted string table, SST)数据混写在同一区域内，在触发 GC 时，需先将较长生命周期的 SST 拷贝到其他区域，因此增加了数据迁移，加剧了写入放大[2,15-19]。如图 4 所示，在 GC 的过程中，由于数据 1, 2, 3 具有较长的生命周期，在 GC 的过程中，需将这些数据迁移到其他区域，在此过程中加剧了写入放大。

2)compaction 与 GC 操作失调。在数据处理过程中，由于基于 LSM-tree 的键值存储系统执行的 compaction 操作与主机端的区域 GC 之间缺乏协同且相互影响，导致数据的重复迁移，加剧了系统的写入放

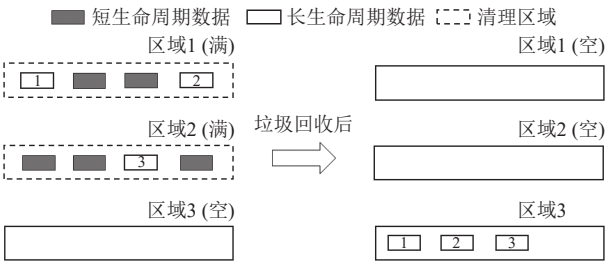


Fig. 4 Write amplification induced during the process of garbage collection[32]

图 4 垃圾回收过程中产生写入放大[32]

大。一方面，compaction 过程可能频繁进行大量有效数据的重写，占用大量区域空间，从而迫使主机端提前触发 GC；另一方面，当 compaction 与主机端 GC 各自独立执行时，将产生“叠加垃圾回收（GC-over-GC）”问题[18,20,22-24,26]。例如，在 compaction 将多个 SST 文件合并后写入新区域的过程中，系统可能已产生了部分有效数据的迁移，然而，如果此时目标区域的空闲空间已小于一定的阈值，则会再次触发 GC，导致数据的再次迁移，从而产生了多次数据迁移，加剧了写入放大[18,20-27]。

3）不恰当的区域重置。在 ZNS 固态硬盘的分区管理下，区域重置完成则标志着 GC 过程的结束。在某个区域需要进行重置操作之前，需先将区域中有效数据拷贝到其他区域，然后再对区域进行清理。在此过程中，如果频繁进行整区重置、提前触发区域重置或者选择不恰当的区域进行重置都将导致大量数据的迁移，从而产生较大的写入放大[28-34]。

2.2 并行性问题

并行性是系统同时调度并执行多个操作的能力，常以系统吞吐量和延迟作为重要衡量指标。

目前，ZNS 接口采用固定区域大小和静态资源映射的模式，即每个区域通常都映射了一定数量的芯片以及相应的通道、晶片（die）和平面（plane）等资源。主机端基于此模式，在逻辑层面进行数据放置和并行写入等操作。然而，在动态负载的场景下，静态区域映射资源的限制可能导致系统并行性无法充分发挥，从而降低系统并行性[29-30,35-36,39]。进一步地，在单个区域内，由于 ZNS 接口的强制顺序写入模式，导致多线程的写入操作由于地址空间的锁定而无法并行执行[40-42]，同时，由于区域间并行读写的相互干扰，或后台区域管理（如重置操作）对前台读写操作的阻塞[10,45,47-48]，导致系统性能产生波动甚至下降。

系统并行性的挑战可进一步细分为以下问题：区域资源分配受限、区域内并行写入路径锁争用和区域间 I/O 干扰与阻塞。

1）区域资源分配受限。大区域 ZNS 固态硬盘由于区域空间映射了较多数量的芯片等资源，因此具有较好的系统内部并行性[2,38]，但是，小区域 ZNS 固态硬盘（单个区域规模在几十至几百 MB 之间）由于区域空间映射的芯片资源较少，导致系统的并行性下降，如图 5 所示。由于现有静态区域资源映射的限制，ZNS 接口无法根据运行时的任务变化动态调整区域资源配置，从而限制了系统性能和灵活性，导致并行性能下降[29-30,35-39]。

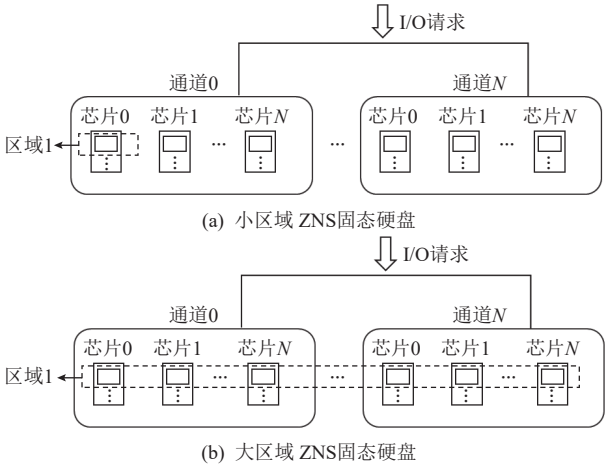


Fig. 5 Zone-level resource allocation in ZNS SSD[36]

图 5 ZNS 固态硬盘中区域的资源分配[36]

2）同一区域内写入路径锁争用。由于强制顺序写入的特性，ZNS 固态硬盘的同一区域在某一时刻仅允许 1 个写指针进行操作，若要在同一区域内发起多线程写入，则必须引入互斥或排队机制，将各线程的写请求串行化，以维护写指针的单一性并避免写冲突[40-42]。例如，在同一区域进行多线程预写日志（write-ahead log, WAL）追加时，必须通过锁机制或调度策略来确保其写入顺序，在此过程中，并发写入将被串行化执行，从而导致系统并行性下降，如图 6 所示。

3）区域间 I/O 干扰与阻塞。由于受到区域间底层共享硬件资源（芯片、通道等）的限制，当同时向存

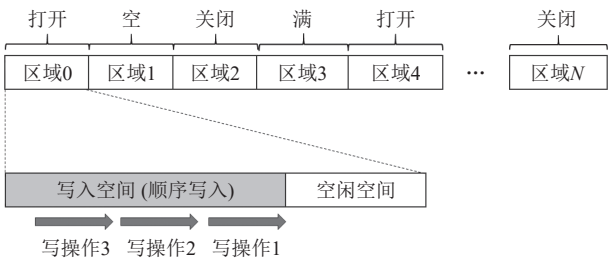


Fig. 6 Serialization of parallel writes within a zone[2]

图 6 同一区域内并行写入串行化[2]

在硬件资源共享的一组区域进行读写操作时，可能会产生区域间 I/O 相互干扰[43,47]，同时，后台操作（如重置）可能与前台读写操作抢占资源，阻碍前台操作的执行，导致前台操作的并行性下降[10,43-44,47-49]。此外，在多租户或多容器环境下，不同租户/容器的并发访问也可能产生互相干扰[45-46]。

2.3 数据一致性与可靠性

数据一致性是指在系统崩溃、电源故障或并发条件下，系统应保证原子性、持久性与写入有序性的能力；可靠性则衡量数据在长期运行及硬件失效（芯片损坏、整盘损坏等）条件下维持正确性与可恢复性的能力。

ZNS 固态硬盘的存储介质由主机端直接管理，它在执行 GC 的过程中，可能出现某些区域擦除过于频繁，从而加速存储介质老化[33,47,50]；并且由于区域强制顺序写入的特性，将产生更加频繁的元数据同步（fsync）以保证系统的崩溃恢复，从而增大了系统同步开销[51-52]。此外，与传统块固态硬盘组成的 RAID 相比，由多块 ZNS 固态硬盘构建的 RAID，在顺序写语义以及区域管理的约束下，部分条带写入所带来的奇偶校验税（partial parity tax）会被进一步放大，同时，跨盘条带写入的同步和调度，导致主机端的性能开销增大并形成新的性能瓶颈[41,53-55]。

对 ZNS 固态硬盘数据一致与可靠性带来的挑战可以分为：存储介质磨损不均衡、系统崩溃恢复开销大和数据冗余与校验性能瓶颈。

1）存储介质磨损不均衡。ZNS 固态硬盘消除了原有设备中 FTL 的地址映射和 GC 等操作，由主机端进行相应的操作。在此过程中，一方面，由于不同区域存储的数据热度存在差异，可能导致少数区域被频繁写入和重置，而另一些区域则由于始终存放冷数据而很少被擦除；另一方面，由于主机端对区域重置粒度的管理存在局限性，产生区域中空页擦除等情况，进一步加速了设备的老化[33,47,50]。例如，当触发区域重置时，由于文件系统（如 ZenFS[2,69,88]）缺乏对区域空间使用情况的监控，即使其中尚有大量空闲空间未被写入，仍将进行整区重置。这无疑加速了区域中介质的磨损，导致设备老化，进而影响设备寿命。

2）系统崩溃恢复开销大。在系统运行过程中，主机端需要保证数据一致性（如元数据更新顺序）并在系统崩溃时进行数据恢复。在此过程中，由于 ZNS 顺序写约束限制，通常会产生频繁的元数据同步更新操作，导致系统恢复较慢且产生额外的性能开销。例如，在顺序写入条件下，即使上层应用仅修改了少量元数据，仍需将整个块（写入单元）进行重写，从而产生了大量的性能开销[51-52]。

3）数据冗余与校验性能瓶颈。由多块固态硬盘组成的 RAID 能够有效提升系统的可靠性[90]。然而，在顺序写入和区域管理约束下，小规模、频繁且非整条带的数据写入将导致奇偶校验块被反复更新，即产生“部分奇偶校验税”问题；同时，主机端不仅需管理逻辑块地址（logical block address, LBA），还需在跨设备的条带布局中遵循顺序写语义并维护写入原子性，进一步增大了在 RAID 中维护冗余数据一致性的性能开销[39,41,53-55]。

本文接下来将依次阐述基于 ZNS 固态硬盘的存储系统的优化技术的研究进展。

3 写入放大优化的研究进展

我们根据数据路径的主要阶段（写入前组织、写入后维护和空间回收）将写入放大优化技术的研究分为基于生命周期感知的数据放置、compaction 与 GC 协同优化、自适应的区域重置。

3.1 基于生命周期感知的数据放置

目前，针对不同生命周期的数据混写的问题，现有研究主要通过对数据的生命周期进行更细粒度的评估或预测，并将相近生命周期的数据放置在同一区域，从而减少 GC 中有效数据的迁移。研究可以分为 2 类：细粒度生命周期的预测和相近生命周期数据的异构区域放置。

细粒度生命周期的预测研究[2,15-18] 主要聚焦在数据的细粒度生命周期的预测，通过引入生命周期评估方法或预测模型，将相似生命周期的数据分配到同一区域，从而减少由于不同生命周期的数据混写所导致的写入放大。

Bjørling 等人[2] 提出了一种基于 LSM-tree 层级进行区域分配的方法 LIZA，根据 SST 数据所在层级来预测其生命周期。LIZA 认为同一层级的 SST 间具有相近的生命周期，因此将相同层级的数据放置在同一区域，以减少 GC 时有效数据的迁移，进而优化写入放大。在此基础上，LIZA 提出了一种基于 LSM-tree 键值存储（RocksDB）的用户态文件系统 ZenFS[2,69,88]，其具有日志管理、数据管理和区域管理等功能，能够将相近生命周期的 SST 分配并存储在同一区域中，如图 7 所示。相比于主机端集成 FTL 层和支持 ZNS 接口的内核态文件系统，ZenFS 采用端到端的数据访问，避免了内核态之间大量的上下文交互、数据复制

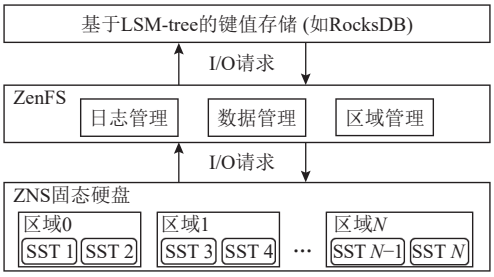


Fig. 7 ZenFS document system architecture[2]

图 7 ZenFS 文件系统架构[2]

等性能开销，后续很多工作[16-19]均以此为基础展开研究。

有学者进一步研究发现，不同的数据，即使所属为相同层级，它们的生命周期也存在着较大差异。因此，仅凭层级作为生命周期的评估标准进行数据分配，在 GC 的过程中仍会产生较大的写入放大[15-18]。为此，国内外研究者陆续提出了更加细粒度的基于生命周期预测的数据分配方法。

Xu 等人[15]提出了一种新型的基于 LSM-tree 的键值存储系统 LeanKV，并引入了一种根据 compaction 历史操作来预测 SST 生命周期的方法，通过统计 SST 的压缩次数作为预测生命周期的量化标准；Lu 等人[16]提出了一种新型的基于 LSM-tree 的键值存储系统 ZoneKV，对于 LSM-tree 中 L_0，L_1 层级的 SST，根据其生命周期相近的特点，将其分配到相同区域，同时，对 L_2 层或以上的 SST，将其切分成若干切片，并且将切片分配到不同区域，有效减少了写入放大；Shen 等人[17]提出了一种根据数据访问频率和所属层级联合预测 SST 生命周期的方法 OAZA，通过综合分析数据所属的 LSM-tree 层级以及该层级中 SST 的相对热度（访问频率）来预测其生命周期。以上研究相比于 LIZA[2]，均通过增加辅助指标对数据的生命周期进行更加准确的评估和预测，并将相近生命周期数据放置在相同区域，在 GC 的过程中减少了有效数据迁移，降低了写入放大。

为了实现更加细粒度的 SST 生命周期的预测，Liu 等人[18]提出了一种新型的基于 LSM-tree 键值存储的框架 Prophet，如图 8 所示。Prophet 引入了一种根据后台操作（如刷新（flush）和 compaction）的执行总和预测 SST 生命周期的方法 FC-Tick，与传统依靠物理时间度量 SST 生命周期的方法不同，Prophet 通过统计后台的 flush 与 compaction 操作次数，将“时间”量化为 2 类操作之和，并基于不同的 compaction 触发场景，构建了生命周期预测模型，对各场景下的生命周期进行比较，以其最小值作为最终预测的生命周

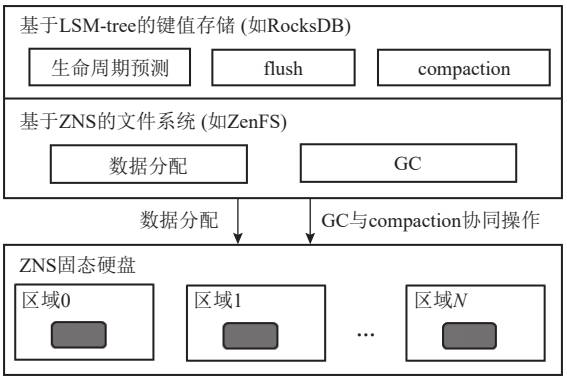


Fig. 8 Prophet architecture[18]

图 8 Prophet 架构[18]

期。同时，Prophet 为每个区域设置了一个生命周期区间，将预测数据的生命周期与区域设置的生命周期区间相匹配放置，从而减少了有效数据的迁移，优化了写入放大。

相近生命周期数据的异构区域放置通过分析 ZNS 固态硬盘中存储介质的特性，将长/短生命周期的数据分别分配到由不同介质构建的存储区域，这种优化方法既能充分发挥各种介质的性能优势，又可以减少有效数据的迁移，进而达到降低写入放大的目的[19]。

Wang 等人[19]针对异构介质 SLC/QLC（single-level cell/quad-level cell）构成的 ZNS 固态硬盘提出了一种主机端管理异构区域数据分配的方法 ZnH_2，如图 9 所示。ZnH_2 采用主机端和设备端协同的设计，在主机端引入按层级预测数据生命周期的方法 LIZA，并根据 SLC/QLC 异构介质的物理特性，将短生命周期数据分配到高性能介质（SLC）构成的区域，长生命周期数据则分配到大容量介质（QLC）构成的区域，同时，ZnH_2 采用 ZNS 模拟仿真器 FEMU[91]模拟设备固件并实现了原型设计，使其支持异构介质 SLC/QLC 及相应的区域管理，并将数据复制操作迁移至设备

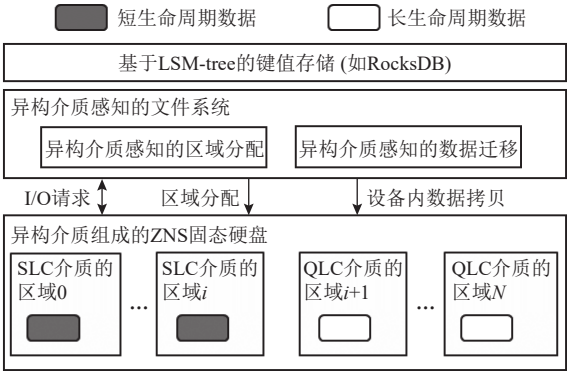


Fig. 9 ZnH_2 architecture[19]

图 9 ZnH_2 架构[19]

端，优化了数据从设备到主机端的传输路径，降低了数据传输开销，减少了由于写入放大所造成的性能开销。

上述典型研究[2,15-19]根据数据的生命周期存在差异的特征，将短生命周期数据和长生命周期数据进行分区存储，提升了同一区域内数据的生命周期的“相似度”，在数据处理的初始阶段减少了有效数据迁移，与混合数据写入相比，有效降低了写入放大。此外，根据异构介质特性进行数据放置的研究在感知数据生命周期的基础上，根据异构存储介质的不同特性进行数据放置，发挥了各类型介质的性能优势，改善了写入放大，并提出了一个新的优化维度以优化系统性能。表 1 从预测生命周期粒度、预测准确性和优化层次等方面对比了基于生命周期感知的数据放置。

Table 1 Comparison of Data Lifetime Prediction Methods

表 1 数据生命周期预测方法对比

相关研究	预测生命周期粒度	预测准确性	优化层次
LIZA[2]	粗粒度	较低	上层应用+文件系统
LeanKV[15]	较细粒度	较高	上层应用
ZoneKV[16]	较细粒度	较高	上层应用+文件系统
OAZA[17]	较细粒度	较高	文件系统
Prophet[18]	细粒度	高	上层应用+文件系统
ZnH_2[19]	粗粒度	较低	上层应用+文件系统+设备端

3.2 compaction 与 GC 协同优化

针对 compaction 与 GC 操作失调的问题，已有研究主要通过优化 compaction 机制或协同 compaction 与 GC 的处理过程予以改进，以减少 GC 过程中有效数据的迁移开销。主要研究可分为 2 类：1)软件层的 compaction 与 GC 协同优化；2)基于设备辅助的 compaction 与 GC 协同优化。

软件层的 compaction 与 GC 协同优化研究[18,20-26]，一方面通过优化 compaction 机制减少后续 GC 阶段中有效数据迁移，另一方面通过协调 compaction 与 GC 操作使二者在执行时机和处理策略上趋于一致，从而降低 GC 操作的频率，进一步减少有效数据的迁移。

在 LSM-tree 的 compaction 机制优化方面，Jung 等人[20]提出了一种对数据的生命周期进行均衡压缩的方法 LL-compaction，在压缩过程中，将短生命周期 SST 分配到专门设置的区域(T-Zone)进行单独的 GC，而长生命周期 SST 则分配到普通的区域，从而在后续 GC 过程中减少了有效数据迁移；Liu 等人[21]提出了一种新型的基于 LSM-tree 的键值存储系统 Life-timeKV，通过优化 compaction 机制消除新生成的 SST 间的跨层键值重叠，同时，根据对 SST 间重叠比率和生命周期的综合评估来选择 SST 文件进行 compaction，以缩小同层 SST 间生命周期的差异；Lee 等人[22]提出了一种压缩感知的区域分配方法 CAZA，通过计算 SST 间的键值重叠率将可能参与同一次 compaction 的 SST 分配到同一区域，从而在 GC 中减少有效数据的拷贝。

在 compaction 与 GC 协同优化方面，Byeon 等人[23]在 CAZA[22]的基础上，提出了一种评估最佳压缩区域的方法 ZACA，根据 SST 间的重叠率以及所属层级，选择最佳的压缩区域，将 compaction 和 GC 相结合，以减少 GC 的执行次数，进一步优化了写入放大。Wu 等人[24]提出了一种基于 LSM-tree 的键值存储系统 ZNSKV，并引入了一种 compaction 和 GC 相互协同的方法，在 compaction 过程中同步执行部分无效数据的 GC，同时，在空间利用率和系统性能之间进行权衡，根据空闲空间大小调整 compaction 触发频率和 SST 选择策略，避免在空间不足时压缩过多的数据，在降低写入放大的同时也保证了系统性能。

在细粒度预测 SST 生命周期和区域分配的基础上，Liu 等人[18]评估了 compaction 和 GC 对写入放大的影响，在基于 LSM-tree 键值存储的框架 Prophet 中引入了基于 GC 的压缩补偿机制，将 compaction 与 GC 相结合。首先，选择有效数据最少的区域，以减少进行 GC 的数据迁移量；然后，通过对区域内各 SST 执行压缩的评估，选择执行主动压缩策略或 GC 策略，减少了有效数据拷贝的频率，进而降低了写入放大。

Liang 等人[25]提出了一种基于碎片化 LSM-tree(fragmented LSM-tree, FLSM-tree)的键值存储系统 ZonesDB。首先，引入了一种区域感知的数据管理机制。该机制在 FLSM-tree 的每一个层级都新增一个区域进行顺序写入，并将新生成的 SST 写入该区域；同时，通过维护内存位图与计数器来监测未使用区域，提升对压缩优化和空间自适应方法的支持；然后，引入了一种新的压缩机制，选择包含最多失效 SST 的区域进行压缩，以减少 GC 过程中的有效数据迁移。ZonesDB 引入伴随式压缩(piggyback compaction)的方法将同一层级相关区域的 SST 一并进行处理，以减少 GC 的频次；最后，提出了一种自适应的区域空间优化方法，根据系统空间使用情况进行自适应空间管理，以达到优化空间使用的目的。

不同于上述将 compaction 与 GC 的过程相统一

的策略[18,23-25]，Hwangbo 等人[26]在 BlobDB(一种基于 LSM-tree 键值分离的存储系统)上提出了一种优化 GC 的方法 BAZA，如图 10 所示。该方法通过设定相同 GC 目标(same-victim GC)，将 compaction 和 GC 作用在同一组 SST 上，以减少有效数据的迁移。首先，将 flush/compaction 操作中产生的“键”按照 LIZA 算法[2]写入 SST 中，同时，将“值”写入 Blob 文件，并按照 Blob 文件顺序映射到区域中，并且采取“先创建先回收”的策略，确保 GC 为同一组文件；然后，为了将每次删除的 Blob 文件规模与删除区域规模相对齐，引入了裁剪算法 Z_Cutoff，目的是细粒度控制 BlobGC 一次处理的数据量，限制了 BlobDB 中每次 GC 的规模，从而减少多次 GC 中所产生有效数据的重复迁移。

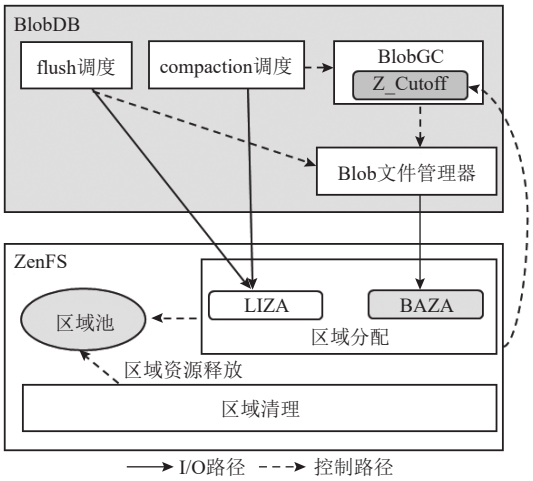


Fig. 10 BAZA architecture[26]

图 10 BAZA 架构[26]

上述方法聚焦于软件层面进行优化，而基于设备辅助的 compaction 与 GC 协同优化通过在 ZNS 接口中引入新原语，实现主机端与设备间的高效协作，从而降低 compaction 与 GC 操作所引起的性能开销。

Han 等人[27]提出了一种设备端支持 compaction 的接口 ZNS+，通过引入 ZNS 新原语(zone_compaction)，并将有效数据复制操作迁移到设备端，降低了主机端的段压缩(segment compaction)的性能开销；同时，在设备端识别同一区域内的碎片数据，并在设备内完成数据重排、合并，优化了有效数据从“设备”到“主机”再到“设备”的数据迁移路径，减少了数据传输开销，减小了写入放大。

目前，研究人员在多层次提出了 compaction 与 GC 协同优化。软件层面的优化可细分为：compaction 机制优化[20-22]、compaction 与 GC 过程协同[18,23-26]，这 2 种典型方法旨在减少有效数据的迁移频次，从而减少写入放大的性能开销。表 2 从优化类别、是否存在独立的 GC 和优化层次等方面对比了软件层中 compaction 与 GC 协同优化的相关方法。在软硬件协同层面，通过引入 ZNS 新原语(zone_compaction)，进行了设备接口增强，实现了将主机端部分功能迁移到设备内，优化了 compaction 与 GC 操作所引起的性能开销[27]，这类研究突破了传统纯软件层面优化的限制，有效减少了主机端的资源占用和性能开销。

Table 2 Comparison of compaction and GC Collaborative Methods in Software-Level

表 2 软件层的 compaction 与 GC 协同方法对比

相关研究	优化类别	是否存在独立的 GC	优化层次
LL-compaction[20]	compaction 优化	是	上层应用
LifetimeKV[21]	compaction 优化	是	上层应用
CAZA[22]	compaction 优化	是	上层应用+文件系统
ZACA[23]	compaction-GC 优化	否	上层应用+文件系统
ZNSKV[24]	compaction-GC 优化	否	上层应用+文件系统
ZonesDB[25]	compaction-GC 优化	否	上层应用
Prophet[18]	compaction-GC 优化	否	上层应用+文件系统
BAZA[26]	compaction-GC 优化	是	上层应用+文件系统

3.3 自适应的区域重置

为减轻由于不恰当的区域重置所产生的系统写入放大，现有研究主要通过优化区域重置粒度或优化区域重置过程来调整区域重置的频次，从而减少有效数据的迁移。已有研究可分为 2 类，分别是细粒度的区域重置、区域重置过程优化。

细粒度的区域重置研究[28-30]主要聚焦在细粒度的区域重置技术。由于整区的规模较大且具有较多的数据，当区域重置时，将产生大量的数据迁移，导致较大的写入放大。因此，现有研究将整区划分为较小的分区单元进行部分区域重置，减少有效数据的迁移，进而降低系统的写入放大。

为了减少整区重置的频次，Choi 等人[28]提出了一种基于段(segment)GC 的方法 LSM_ZGC，将一个区域划分为若干段(如每一段为 2 MB)，并以段为单元进行细粒度的 GC；Huang 等人[29]提出了一种对区域进行重新划分的新架构 SplitZNS，根据闪存的管芯资源，将传统规模较大的区域划分为较小的子区域(subzone)，并采用 FEMU[91]模拟设备固件实现了原型设计，优化了区域管理，以支持子区域粒度的划分和区域重置，减少了有效数据的迁移，进而降低了写入放大。

然而, 即使以小区域规模进行重置, 也无法避免有效数据的迁移。因此, Liu 等人[30] 提出了一种动态调整区域规模的方法 Hi-ZNS, 如图 11 所示。Hi-ZNS 不再以 ZNS 固态硬盘中原有的逻辑区域作为区域重置的目标, 而是根据具体的 SST 或 WAL 文件的大小, 通过动态映射不同数量的闪存块的方式来调整区域的规模, 将区域和 SST 的大小相对齐, 使每个区域恰好容纳 1 个 SST 或 WAL 文件, 从而在区域重置时无需再进行有效数据迁移。Hi-ZNS 通过 FEMU[91] 模拟设备固件并实现了设备原型, 将条带单元(strip)作为区域映射的基本单元, 通过设备容量维护器管理 ZNS 固态硬盘的实际可用容量, 同时在设备端中引入容量分配器, 用于记录所有 strip 的状态(有效和无效), 并引入区域映射表记录逻辑区域与 strip 之间的映射关系, 从而根据文件大小动态映射相应的区域, 改善了写入放大。

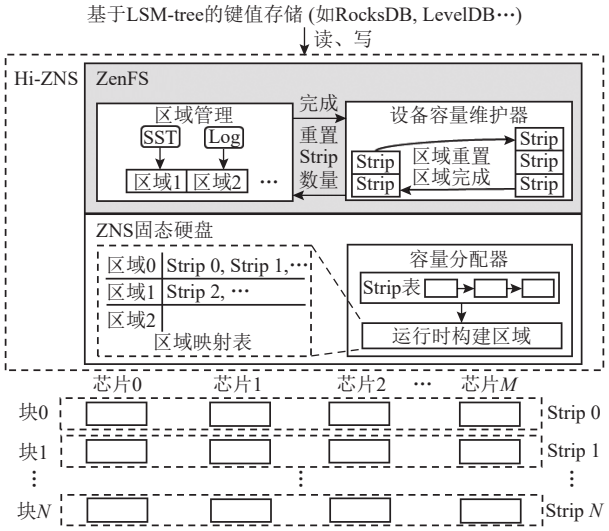


Fig. 11 Hi-ZNS architecture[30]

图 11 Hi-ZNS 架构[30]

区域重置过程优化针对区域重置过程中的主要阶段: 重置前数据布局(根据工作负载调整数据放置)、重置时机选择(根据区域空间的使用情况决定何时进行区域重置)以及重置执行(在区域重置过程中优化有效数据迁移), 减少区域重置发生的频次和有效数据的迁移, 进而改善写入放大。

为了更好地进行数据布局, 减少区域重置的频次, 刘扬等人[31] 提出了一种基于 ZNS 固态硬盘数据感知的新型索引 ZB$^+$-tree, 基于该索引结构将随机写操作分配到设备内的一小部分(支持随机写入)的区域, 降低了由于数据频繁更新引起的性能开销; 同时, 系统将顺序写操作分配到设备内大部分区域, 并且根据节点更新的冷热频率, 将相近生命周期的数据分配到同一区域, 从而减少由于随机写入所产生的频繁的区域重置, 改善了写入放大。

为了更加细粒度地优化区域重置的触发时机, Byeon 等人[32] 提出了一种基于空闲空间自适应的区域重置方法 FAR, 根据区域中空闲空间的情况来选择区域重置的触发时间。当区域中空闲空间充足时, 提高写入指针的阈值等优化, 减少主动进行区域重置的频次; 当空闲空间紧张时, 则增加区域重置频率, 释放更多的可用空间, 进而改善写入放大。

Liu 等人[33] 提出了一种自定义区域重置的方法 ZNS-Cleaner, 将区域重置与 GC 解耦。当主机端发出重置命令后, 并不立即进行整区重置, 而是采用一种延迟重置的方法: 首先, 将整区拆分成跨多芯片的页级条带, 以继续使用区域中的空页资源; 然后, 对区域中空页资源进行监控, 当监测到区域中的空页逐渐减少时, 再由设备端决定是否进行区域重置, 相比于 FAR[32] 进一步降低了写入放大。

为了更好地优化重置执行时有效数据的迁移, Tan 等人[34] 提出了一种在区域重置过程中优化有效数据迁移的新架构 Brick-ZNS, 如图 12 所示。当主机端接收到区域重置命令后, 先在设备端引入“存储内数据迁移”和“地址重新映射”机制, 将区域内的有效数据从待清理区域重新映射到目标迁移区域, 然后再执行区域重置操作, 从而优化了 GC 过程中有效数据的迁移路径, 减少了数据传输, 进而改善了写入放大。

目前, 自适应的区域重置的研究旨在解决数据处理的后期阶段(空间回收)中数据迁移所引发的写入放大问题。其技术路线针对影响区域重置的主要

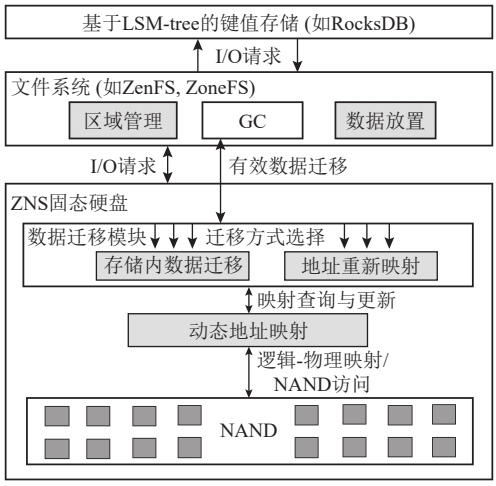


Fig. 12 Brick-ZNS architecture[34]

图 12 Brick-ZNS 架构[34]

因素(区域构成、数据分布以及回收时机等)进行优化，主要可分为 2 种：区域细粒度重置和区域重置过程优化。区域细粒度重置的研究将整区划分为较小的分区单元，进行部分区域重置，减少了整区重置的频次[28-30]。表 3 从区域重置粒度、区域大小是否可配置、是否具有数据迁移和优化层次等方面对比了细粒度的区域重置方法。区域重置过程优化研究[31-34]，则针对区域重置的主要阶段，根据工作负载调整数据放置策略、区域空闲空间等实际情况来优化触发时机以及重置执行时的数据迁移优化，分别降低了在区域重置过程中的写入放大。虽然 ZNS-Cleaner 的主要目的是解决设备介质磨损不均衡问题，但其延迟重置等策略也达到了减少整区重置的频次和改善写入放大的目的。

Table 3 Comparison of Fine-Grained Zone Reset Methods

表 3 细粒度的区域重置方法对比

相关研究	区域重置粒度	是否区域可配置	是否数据迁移	优化层次
LSM_ZGC[28]	段	否	是	上层应用+文件系统
SplitZNS[29]	子区域	是	是	上层应用+文件系统+设备层
Hi-ZNS[30]	条带单元	是	否	文件系统+设备层

4 并行性优化的研究进展

基于 ZNS 固态硬盘的系统并行性主要受到资源限制、局部(区域内)并行和全局(区域间)并行等 3 方面的影响。鉴于此，我们将并行性优化技术的研究分为区域资源分配优化、区域内并行优化和区域间并行优化 3 类。

4.1 区域资源分配优化

针对区域资源分配受限的问题，目前已有研究主要通过优化并行调度或动态调整区域资源映射等策略提升系统并行性。研究可分为 2 类：并行调度优化和区域资源映射优化。

并行调度优化研究[35-36]主要聚焦在小区域资源限制下的系统并行调度优化，在不改变区域资源映射的前提下，通过在主机端优化并行调度策略提升系统吞吐量、降低延迟。

Im 等人[35]提出了一种面向小区域 ZNS 固态硬盘的并行调度方法。首先，引入了一种并行写入策略，对于 LSM-tree 中 L_0 层级的 SST，由于其大小固定，采用基于块的并行写入机制，而对于其他层级的 SST，则采用文件拆分策略，将 SST 拆分成若干部分，并行写入到各区域中。同时，引入了一种并行读取策略，从多个区域并行读取块文件，提升读取并行性，并且采用了一种预读双缓冲的机制，通过在每个 SST 上维护 2 个交替使用的内存缓冲区减少读取等待时间。

此外，Choi 等人[36]提出了一种自适应的区域组管理方法 AutoZG。首先，AutoZG 根据工作负载写入频率等特征，将具有相近生命周期的数据划分到同一个数据流中；然后，根据数据流的规模将多个区域组合成区域组；最后，将相近生命周期的数据流写入到同一组的区域内。AutoZG 通过将多个区域组合成区域组，有效地利用了设备带宽，提升了系统性能。

区域资源映射优化研究[29-30,37-39]采用动态资源映射或区域规模调整的方法，优化硬件资源的利用率，提升系统并行性。

为了进一步优化区域资源利用率，Tan 等人[37]提出了一种区域资源动态映射的方法 Para-ZNS(如图 13 所示)，旨在优化单个区域映射(die 级)资源并行性的限制。Para-ZNS 采用 FEMU[91]模拟设备固件并进行了原型设计，在设备端引入了并行块分组(parallel block grouping)机制，在单个 die 内将不同 plane 上具有相同写入偏移的块组成多平面块组，并将此类块组作为区域映射的最小单元。在区域写入时，Para-ZNS 首先识别空闲资源，然后，将其动态映射到所需区域，并将待写入数据分布到多个芯片上并行写入，提升了系统的并行性。

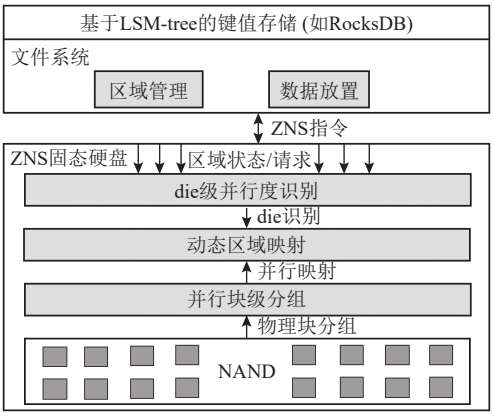


Fig. 13 Para-ZNS architecture[37]

图 13 Para-ZNS 架构[37]

Min 等人[38]提出了一种针对多租户并行优化的弹性分区命名空间的接口 eZNS，引入了区域资源扩展机制(zone ballooning)，基于对区域所需资源的预测，动态地调整区域条带配置和资源映射，从而灵活地分配每个区域的资源，以提高 die 级的系统并行性。此外，通过识别不同租户间的 I/O 操作进行调度优化，

有效避免了区域间的干扰,提升了系统并行性。如图 14 所示,eZNS 主要分为 2 个部分:一为区域仲裁器,位于硬件抽象层。其中,设备影子视图为区域分配和 I/O 调度提供近似的数据位置,序列化区域分配和区域资源扩展模块则用于动态映射区域资源和优化数据放置。二是 I/O 调度器,位于主机端,识别不同租户间的区域 I/O 冲突,并通过优化调度减少系统的延迟。

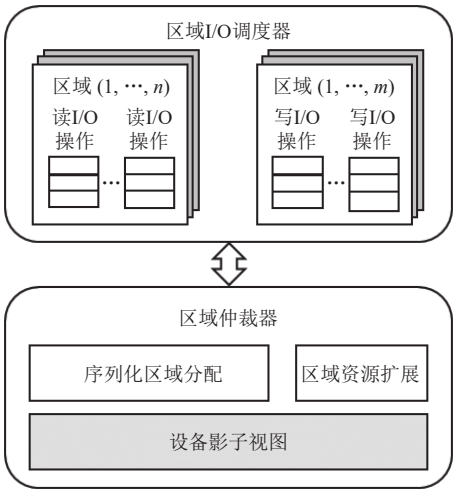


Fig. 14 eZNS architecture[38]

图 14 eZNS 架构[38]

此外,SplitZNS[29] 则在优化写入放大的基础上引入子区域加速机制(subzone ring),在每个芯片上维护了一个 FIFO 顺序缓冲,加速数据写入,避免了由于大区域拆分所造成的并行性下降,从而提升了系统并行性。Hi-ZNS[30] 在优化写入放大的基础上动态调整区的资源映射,有效提升了系统并行性能。FlexZNS[39] 则在优化系统可靠性的基础上,基于 FEMU[91] 模拟设备固件并实现了系统原型,引入了区域规模调整机制,通过灵活调整区域大小,在相同资源条件下支持同时打开更多的区域,从而提升了系统并行性。

上述研究分别从并行调度优化和区域资源映射优化 2 方面进行了区域资源分配优化。并行调度优化针对小区域内部并行性相对较低的情况,通过优化并行写入策略[35-36] 有效提升了系统并行性。区域资源映射优化方面,动态资源映射[30,37-38] 和调整区域规模[29-30,37-39] 等研究优化了区域与物理资源的动态映射方式,避免了由于静态映射资源所造成的并行性下降以及资源浪费等问题。表 4 从区域规模是否可配置、资源动态映射和优化层次等方面比较了区域资源分配优化方法。

Table 4 Comparison of Zone Resource Allocation Optimization Methods

表 4 区域资源分配优化方法对比

相关研究	区域规模是否可配置	资源动态映射	优化层次
文献 [35]	否	否	上层应用+文件系统
AutoZG[36]	否	否	文件系统
SplitZNS[29]	是	否	上层应用+文件系统+设备层
Hi-ZNS[30]	是	是	文件系统+设备层
Para-ZNS[37]	是	是	设备层
eZNS[38]	是	是	主机端接口层
FlexZNS[39]	是	否	文件系统+设备层

4.2 区域内并行优化

针对同一区域内写入路径锁争用的问题,目前已有研究主要引入 ZNS 新原语或在设备内进行写入顺序调整优化,提升了系统并行性。研究可分为 2 类:1)基于 ZNS 新原语的并行优化,2)设备端支持的区域并行优化。

基于 ZNS 新原语的并行优化研究[40-41] 通过引入 ZNS 新原语 zone append[67],实现并行写入追加,避免了同一区域内多线程写入所产生的锁竞争,进而提升了系统的并行性。

Lee 等人[40] 提出了一种支持同一区域内并行写入的方法 WALTZ,通过引入 ZNS 新原语 zone append,避免了 WAL 并发写入所产生的锁争用。当某个 WAL 区域写满时,WALTZ 将自动触发 WAL 区域替换,后续日志即切换到新区域继续进行并行追加。同时,WALTZ 引入了惰性元数据管理(lazy metadata management)机制,采用延后处理的方式,避免了每次写操作都进行元数据更新,进一步减少了性能开销。

在基于 ZNS 固态硬盘组成的 RAID 阵列中,Wang 等人[41] 提出了一种区域并行写入优化的方法。ZapRAID 将 ZNS 新原语 zone append 融入 RAID 写流程中,将多个逻辑条带构成一组,组内的数据通过顺序追加的方式并行写入到各盘的区域中,同时,将组与组之间进行粗粒度序列化,避免了主机端需锁定小条带的操作限制,从而提升了系统并行性。

设备端支持的区域并行优化[42] 在兼容主机端顺序写语义的前提下,通过在设备端维护写入顺序一致性,实现区域内多队列并发写入,从而提升了系统并行性。

Wang 等人[42] 提出了一种设备内并行写入的架构 oZNS,如图 15 所示。oZNS 在 ZNS 固态硬盘内增加了一个间接寻址层,用以记录主机端发送写命令

时附带逻辑位置信息，并将其写入各数据缓存块中。同时，oZNS 不会立即更新写入顺序，而是在设备端空闲时将写操作按照正确顺序再组织到同一区域中，进而优化了在同一区域内多个线程的并行写入，避免了线程间的锁竞争，从而提升了系统并行性。

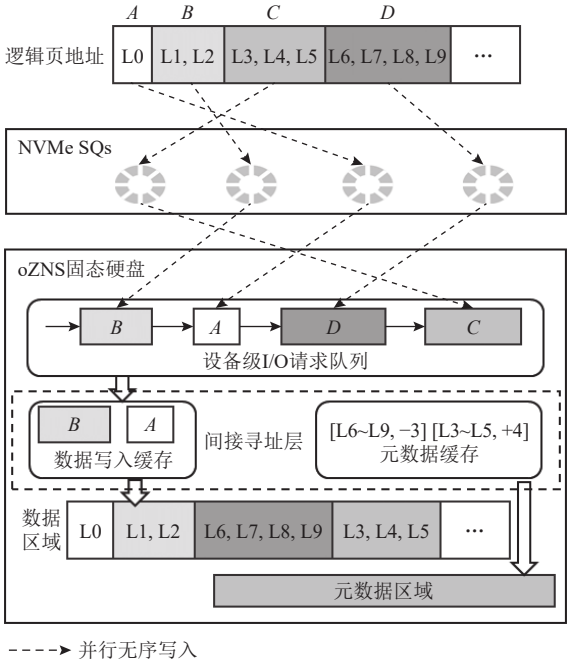


Fig. 15 oZNS architecture[42]

图 15 oZNS 架构[42]

为了缓解同一区域内并行写入所产生的性能瓶颈，已有典型研究主要以 ZNS 新原语 zone append 为基础进行优化[40-41]，并与现有 ZNS 接口保持兼容，因而具有较强的可用性。相比之下，硬件层面[42]则无需引入 zone append 等新原语，而在设备固件中实现无锁并行写机制，将顺序约束写操作的性能开销由主机端下发至设备内部，同时，无需修改现有的 ZNS 接口，为 ZNS 固态硬盘优化提供了重要的支持。

4.3 区域间并行优化

针对区域间 I/O 操作引起的干扰与阻塞问题，目前已有研究主要通过识别区域间存在的 I/O 冲突进行调度优化，以提升系统的并行性。相关研究可分为 2 类：在软件层面进行 I/O 并行调度优化和基于硬件辅助的 I/O 并行调度优化。

在软件层面进行 I/O 并行调度优化研究[10,43-46]通过识别并量化区域间并行冲突进行调度优化，从而提升系统的并行性。

Bae 等人[10]提出了一种针对区域间并行读写冲突的优化方法，当识别到区域内出现性能下降的情况时，该方法首先识别由于硬件资源共享所造成的并行冲突，然后基于冲突情况将区域划分为多个冲突组，并将相互干扰的区域划分在同一组内，最后引入区域冲突感知的调度机制，有效分配区域间的 I/O 请求，避免将并发请求调度至相互冲突的区域，从而有效提升了系统并行性。Long 等人[43]提出了一种基于内核态文件系统 F2FS 的区域间并行写入优化的方法 FPZone，其核心是将数据写入流并行映射到不同芯片组的多个区域，以提高区域间的并行性。首先，提出了一种并行区域分组方法，将映射到不同芯片组的区域划分为多个“并行区域组”；然后，引入了一种基于并行区域组的数据分配方法，将不同热度的数据写入流分配到不同的并行区域组；最后，提出了一种并行区域组写入机制，以区域的条带为单元并行写入区域组中的多个区域，以提升系统的并行性。

Zhang 等人[44]针对 POSIX 文件系统无法有效利用 ZNS 设备中多个区域带宽的问题，提出了一种面向大区域 ZNS 固态硬盘的内核态文件系统 SchInFS，通过在文件系统中集成块层调度机制，减少了传统块层中调度产生的性能开销。SchInFS 首先采用多头日志（multi-head logging）结构，对多个区域进行并行写入，提升了区域的吞吐量；其次，为每种日志提供了一个独立的合并队列，优化了跨线程写入的效率；最后，引入了块 I/O 提交机制，代替传统内核中 mq-deadline 调度机制，保证了在多个日志并行时每个区域写提交的有序进行，如图 16 所示。SchInFS 首先识别每个页中数据类型和冷热程度，为其选择相应类型的日志，并将其写入相应的数据块或节点块；然后，将各日志的写请求在其专属日志队列中完成合并；最后，监控每个区域的状态来判断是否有正在写入的请求，并通过块 I/O 提交控制器进行区域数据分配。

近年来有研究者在基于 ZNS 固态硬盘支持的高并发的云存储环境中进行了并行调度优化方面的研究[45-46]。Lv 等人[45]提出了一种基于分布式云存储的缓存系统 ZonedStore，通过引入多层缓冲区管理者机制（multi-layer buffer manager）和内存并发索引（in-memory concurrent index）结构提高了 I/O 调度效率，有效提升了系统吞吐量，并降低了延迟。Han 等人[46]提出了一种针对容器化环境中读写干扰的优化方法 ZNS-CRIU，为每个容器分配独立的区域进行检查点（checkpoint）数据的写入；由于将每个容器分配在不同区域，避免了自身检查点写入对其他容器的影响；同时，引入了多区域条带机制，将一个容器的

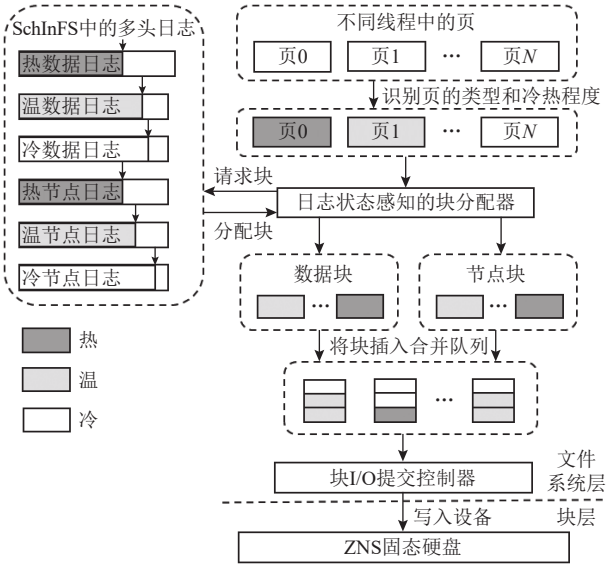


Fig. 16 SchInFS architecture[44]

图 16 SchInFS 架构[44]

检查点数据并行地写入到多个区域，有效地减少了读写干扰，进而提升了系统的并行性。

基于硬件辅助的 I/O 并行调度优化研究[47-49]主要聚焦在基于硬件辅助的 I/O 并行优化。相比于主机端调度优化，设备端可以更有效地识别 I/O 请求间冲突和优化资源占用，并从设备内部减少 I/O 之间的干扰，因此，设备端可以实现更加有效的 I/O 调度策略，使其更好地利用 ZNS 接口的并行特性。

Liu 等人[47]提出了一种基于硬件辅助的 I/O 调度架构 Fair-ZNS，目的是避免多个应用间并行所产生的 I/O 干扰与阻塞。首先，Fair-ZNS 引入了不公平 I/O 请求（引起较大延迟的 I/O 请求）的识别机制；然后，通过 ZNS 感知调度方法，将已识别的不公平 I/O 请求调整到队列首部以减少系统延迟；最后，引入了自平衡协调机制，微调 I/O 请求的顺序以减少由于 ZNS 感知调度对系统延迟的影响。如图 17 所示。Fair-ZNS 是一个设备级调度器，首先，其动态监控各区域的队列的延迟和吞吐量，若发现某队列变慢程度明显高于其他队列，则将不公平标识符标注在影响队列延迟较大的 I/O 请求上；然后，ZNS 感知调度器扫描 I/O 队列，发现其中的不公平 I/O 请求后对队列进行重排，将不公平 I/O 请求调度到队列首部；最后，通过自平衡协调器对 I/O 队列进行再次微调，降低在 ZNS 感知调度的过程中调整队列对系统并行性造成的影响。

为了减少读写操作之间的相互干扰，Zhu 等人[48]提出了一种在设备端缓解读写干扰的方法 TWIW。该方法观察到，当系统将区域状态标记为“满”时，区域中仍存在一些空闲空间，于是将此空闲空间用于备份热度较高的读数据，从而减少了读写之间的相互干扰，同时，TWIW 引入了请求拆分调度机制，设备端将读写请求划分为小块进行并发处理，避免了长时间占据芯片资源，从而提升了系统并行性。

此外，Doekemeijer 等人[49]提出了一种基于 ZNS 固态硬盘的模拟仿真器 ConfZNS++，用于评估并处理后台 I/O（如 finish 和 reset）对前台 I/O 的干扰和阻塞。通过引入 ZINC 调度器（优先调度主机端的前台 I/O 操作）优化 I/O 顺序，避免过多的后台 I/O 同时执行；同时，引入 SoftFinish 操作（将设备 finish 操作部分移至主机端），通过主机端发出 finish 指令，使目标区域进入“即将完成”状态，此操作将缓解对前台读写操作执行的阻塞，待前台读写操作完成后，再更新区域的状态，从而进一步降低了后台 I/O 对前台 I/O 操作的不利影响。

目前，区域间并行优化的技术路线主要采用全局优化策略，可分为软件层的 I/O 并行调度和基于硬件辅助的 I/O 并行优化两大类。在软件层面[10,43-46]识别区域间并行冲突关系，优化调度算法将可能冲突的请求错开或隔离，达到最大化系统并行性的目的。其中，面向 ZNS 文件系统研究[43-44]通过改进文件系

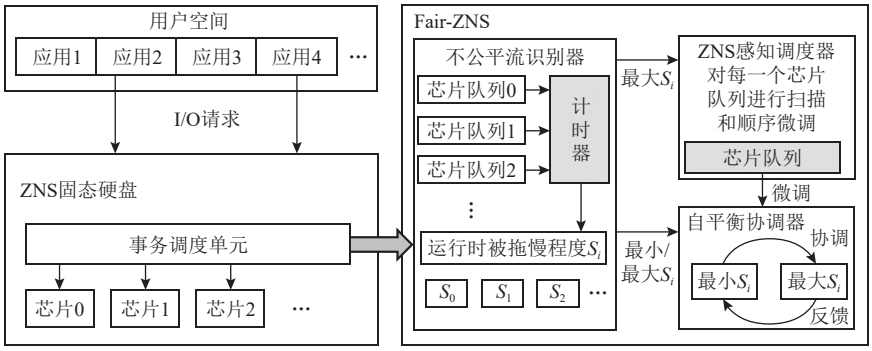


Fig. 17 Fair-ZNS architecture[47]

图 17 Fair-ZNS 架构[47]

统结构降低了上层应用直接管理区域的难度，使之与 ZNS 固态硬盘的顺序写入更加契合，有效利用了 ZNS 的并行特性，降低了内部干扰和性能瓶颈，从而提升了系统并行性。基于硬件辅助的优化研究[47-49]则充分利用了设备端的优势，通过识别并行冲突，在设备层直接进行调度，实现了更有效的性能提升。

5 数据一致性与可靠性优化的研究进展

基于 ZNS 固态硬盘的存储系统在数据一致与可靠性上主要受到介质磨损与寿命约束、持久化顺序保障、数据正确性与容错 3 方面的影响，据此我们将数据一致性与可靠性的优化技术研究分为存储介质磨损均衡优化、系统崩溃恢复与一致性优化、数据冗余与校验优化 3 方面。

5.1 存储介质磨损均匀优化

针对存储介质磨损不均的问题，目前已有研究主要采用优化冷热数据的区域放置策略、优化区域内空闲介质的无效擦除，以及公平 I/O 调度等方法，减少区域内介质的擦除次数、优化整体磨损均衡，以达到提升设备使用寿命的目的。现有研究可以归纳为 3 类：基于冷热数据的区域磨损感知优化、基于空间利用率的擦除优化和公平调度驱动的均匀磨损优化。

基于冷热数据的区域磨损感知优化研究[50]不同于 3.1 节（基于生命周期感知的数据放置）的研究，这类研究将区域间介质磨损状态纳入数据放置决策评估，以提升设备使用寿命。具体而言，首先，系统基于数据的访问频率评估其“冷热”程度；然后，将更新频率较快的数据分配到擦除次数较少的区域，将更新频率较慢的数据分配到擦除次数较多的区域；同时，平衡各区域间的擦除次数，进而延缓整体介质的老化进程。

Long 等人[50]提出了一种磨损区域感知的方法 WA-Zone，如图 18 所示。首先，WA-Zone 引入了磨损感知的区域分配机制，将具有不同冷热程度的数据动态分配到不同的区域，并基于对区域擦除次数等信息的记录，使各个区域之间的擦除情况呈均匀分布；然后，提出了一种部分擦除的区域重置方法，以减少区域中空闲介质的擦除；最后，引入了一种基于块的擦除方法，采用基于区域内循环偏移的块分配机制，即每次写入数据时都从上一次起始位置向后偏移，使区域内所有块均匀参与擦写，避免少数块被反复擦除而其他块长期闲置。

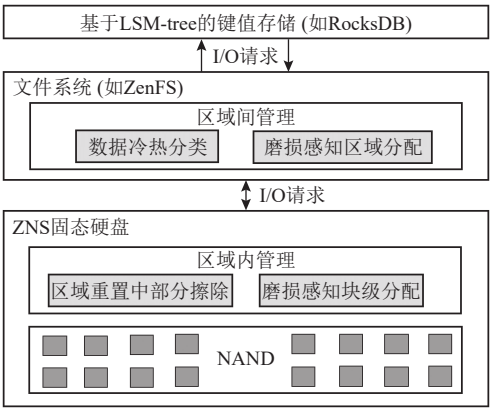


Fig. 18 WA-Zone architecture[50]

图 18 WA-Zone 架构[50]

基于空间利用率的擦除优化研究[33]不同于 3.3 节（自适应的区域重置）的研究，这类研究感知区域内空闲空间的利用率，将区域重置与介质擦除相解耦，具体而言，通过部分重置的方式仅擦除区域中已使用的部分，同时，根据区域中空闲空间的情况选择重置的区域，或通过延迟触发区域重置的方法减少区域中未使用介质的反复擦除，从而提升设备的使用寿命。

Liu 等人[33]提出一种将区域重置和介质擦除解耦的方法 ZNS-Cleaner，通过引入细粒度的空间管理机制继续使用区域中存在的空页，从而减少了区域重置阶段的空页擦除操作。ZNS-Cleaner 从 2 个方面（减少闪存块的擦除次数和保证所有闪存块的磨损均衡）优化了 ZNS 固态硬盘的使用寿命。如图 19 所示，ZNS-Cleaner 主要包括 3 个部分：细粒度的区域管理、磨损优先的区域重构和延迟擦除。首先，细粒度的区域管理对存储空间进行细粒度的划分，将存储空间划分为页级条带，以继续使用闪存块中的空闲页（p-strip）；然后，采用磨损优先的策略，通过空闲页级条带来重建新的区域，从而优化了 ZNS 固态硬盘中所有闪存块的磨损均衡；最后，基于延迟擦除的方式自主管理闪存块的擦除时机，使其不再受区域重置操作的限制。

公平调度驱动的均匀磨损优化研究[47]通过优化公平调度算法，使 I/O 请求均匀分布在不同区域，避免某些区域因连续高负载而产生磨损过大的情况，从而优化均衡磨损。

Liu 等人[47]提出了一种公平调度算法 Fair-ZNS，该方法根据不同应用的延迟情况进行调度优化，当监测到某应用延迟明显高于平均水平时，Fair-ZNS 动态识别并优先处理其中的高延迟 I/O 请求，缓解应用间的运行时间不均衡，达到平衡应用间 I/O 带宽占用

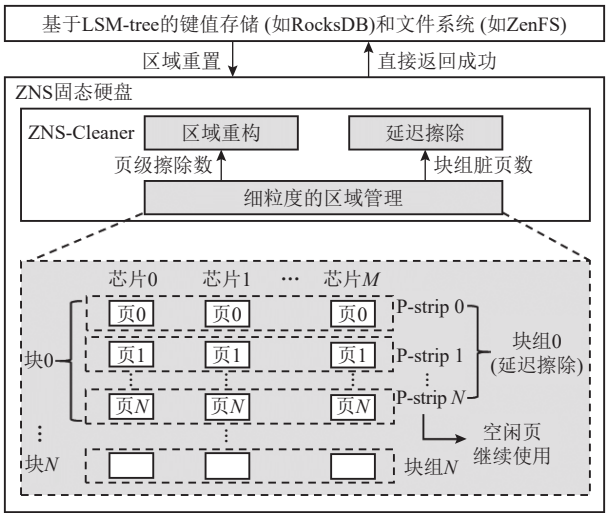


Fig. 19 ZNS-Cleaner architecture[33]

图 19 ZNS-Cleaner 架构[33]

和减少单应用持续高负载, 进而优化磨损平衡。

为了更好地均衡各区域的磨损程度和延长闪存介质的使用寿命, 存储介质磨损均匀优化的研究主要聚焦在基于冷热数据的区域磨损感知优化[50]、基于空闲空间利用率的擦除优化[33]和公平调度驱动的均匀磨损优化[47] 3 个方面。其中, 基于冷热数据的区域磨损感知优化实现了跨区域间的负载平衡, 缩小了各区域擦除次数差异; 基于空闲空间利用率的擦除优化通过感知区域内空闲空间的利用率, 将区域重置与介质擦除解耦, 减少了空间介质的反复擦除; 同时, 公平调度驱动的均匀磨损优化通过公平调度算法, 避免了某一区域处于连续高负载的状态。上述技术路线旨在避免“短板效应”, 防止任何一个区域过早达到使用寿命, 最大限度地延长 ZNS 固态硬盘整体的使用寿命, 降低由于介质失效导致的系统风险。

5.2 系统崩溃恢复与一致性优化

针对系统崩溃恢复性能开销大的问题, 已有研究主要通过优化元数据管理避免整块重写, 减少元数据同步写入量, 从而降低在顺序写入下的同步开销, 实现更加高效的一致性恢复。研究分为软件层元数据更新优化和硬件辅助的系统崩溃快速恢复 2 类。

软件层元数据更新优化研究[51]主要聚焦在面向 ZNS 接口的文件系统的元数据更新的优化, 通过设计轻量级的元数据日志更新策略, 避免重写整个区域, 以减少元数据同步写入量, 进而优化系统崩溃时的恢复效率。

Qi 等人[51]提出了一种优化系统崩溃一致性的文件系统 BlzFS, 该系统引入了字节级日志区域(byte-loggable zone), 用于支持细粒度的元数据更新。在元数据块发生部分修改(如文件大小变化需更新索引节点(inode)的部分字节)时, BlzFS 通过记录字节范围(byte range)的方式标识受影响区域, 并以小粒度日志项的形式, 将这些增量更新顺序写入到 WAL 区域, 从而避免整体重写元数据块, 提高了系统崩溃后的恢复效率。

硬件辅助的系统崩溃快速恢复研究[52]主要聚焦在硬件辅助的系统崩溃快速恢复机制, 利用 ZNS 固态硬盘所维护的写指针的持久状态, 确定崩溃时已写入的数据范围, 从而无需依赖额外的日志信息确定写入数据的顺序性, 优化了文件同步(fsync)的性能, 并有效减少了系统恢复阶段的扫描操作, 实现了更高效的系统一致性恢复。

Lee 等人[52]提出了一种写指针顺序恢复的方法 OPRW, 优化文件同步性能的同时也保障了文件系统恢复过程中的数据一致性。OPRW 利用每个区域中写指针的状态来确定数据的持久性, 从而避免了传统方法中额外的 I/O 操作, 优化了文件系统恢复过程中的数据一致性, OPRW 过程如图 20 所示。在此过程中, 系统崩溃导致部分节点块已到达设备, 而对应数据块仍在传输或位于 DRAM 中, 此时系统存在潜在的数据损坏。在系统恢复过程中, OPRW 首先获取每个区域的写指针位置, 并创建写指针表(write pointer table, WPT); 然后, 依次处理待恢复的节点块, 并将对应的数据块地址与写指针表进行比较, 若存在任一地址不小于写指针地址, 则判定该节点数据不一致, 因而删除该节点, 并跳过同一 inode 的后续节点块。此过程减少了传统方法中判定数据有效性而引入的额外 I/O 操作, 有效提升了系统恢复效率。

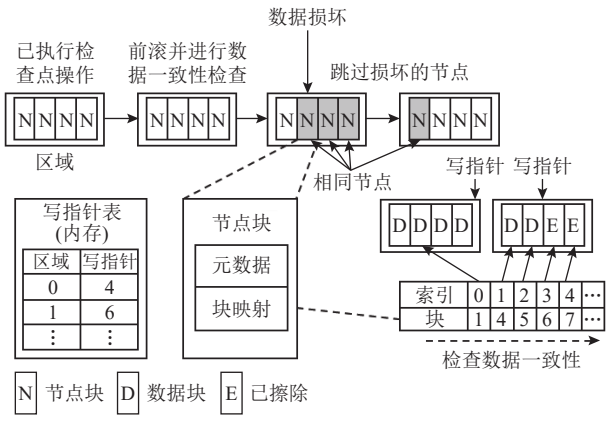


Fig. 20 OPRW process[52]

图 20 OPRW 过程[52]

在上述系统崩溃恢复与一致性优化的研究领域，已有研究多关注元数据更新机制的优化，主要分为 2 种策略。1)写时一致性[51]，在文件系统层级通过位图记录元数据的细粒度变更，从而避免了整个原数据块的重写所引发的性能开销。2)恢复时一致性[52]，充分利用 ZNS 顺序写语义的特性来确定崩溃时哪些数据已落盘，减少了传统文件系统中的日志扫描等操作，在保证数据一致性前提下，有效提升了恢复效率和系统性能。

5.3 数据冗余与校验优化

针对数据冗余与校验性能瓶颈的问题，现有工作主要优化条带布局和奇偶校验。一方面，在多设备环境下通过优化条带管理和奇偶校验等方式提升整体系统性能；另一方面，在单设备场景下，通过本地条带优化与轻量级奇偶校验实现高效的数据管理。已有研究可分为：基于 RAID 的冗余校验优化和单设备的冗余校验优化两大类。

基于 RAID 的冗余校验优化研究[41,53-55] 主要聚焦在基于 ZNS 固态硬盘构建的 RAID 阵列的冗余校验优化和条带布局优化，并引入 ZNS 新原语，以减少由于奇偶校验税所带来的性能开销。

Kim 等人[53] 提出了由 ZNS 固态硬盘构建的 RAID 阵列 RAIZN，其核心设计是地址空间的管理，当主机端向 RAID 发送 I/O 请求时，RAIZN 将逻辑地址转化为相应的物理地址，分别发送给每个 ZNS 固态硬盘，并在 RAID 中分配专用区域分别存储奇偶数据和元数据，相较于传统 SSD 组成的 RAID 阵列，具有更稳定的吞吐量和更低的尾延迟。如图 21 所示，RAIZN 至少由 3 个设备组成，每个逻辑区由若干条带组成，每个条带由 2 个数据单元和 1 个奇偶校验单元组成，其中，每个条带单元对应物理设备的一个区域。条带化设计不仅提升了系统并行性，还有效增强了数据的可靠性。

Wang 等人[41] 针对 RAID 阵列中区域并行性受限的问题，提出了一种提升区域并行性的 RAID 列阵 ZapRAID，通过引入 ZNS 新原语 zone append 以实现高并行顺序写入和轻量级条带管理。为降低条带管理开销，ZapRAID 引入了分组(group-based)数据布局机制，将多个条带构建成条带组，并在完成收集条带数据后一次性生成并写入奇偶存储块中，从而提升了写入效率，同时也增强了系统的数据可靠性。

尽管 ZNS 接口能够支持更细粒度的存储管理，但由于其与上层软件(一般为块接口)的兼容性问题，导致系统性能下降，因此，Yi 等人[54] 提出了一种优化 RAID 接口的阵列 BIZA，向上通过块接口对接上层应用，向下通过 ZNS 接口控制存储设备。BIZA 引入了 ZNS 新原语 ZRWA，将部分奇偶校验和其他高频更新数据随机写入到划定的小区域内，并直接覆盖已失效的部分奇偶校验数据，从而减少了部分奇偶校验税引起的性能开销；同时，引入影子缓存算法(ghost-cache-based algorithm)，将奇偶校验块集中管理，避免了在各区域中都写入数据，并在后台自动完成奇偶更新和校验，进一步提升了系统的可靠性。

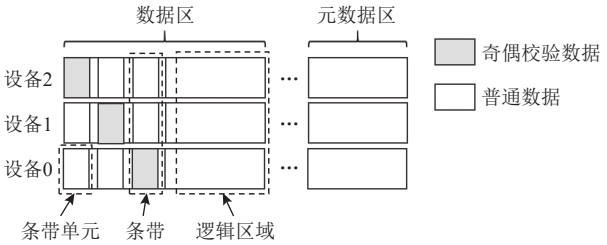


Fig. 21 RAIZN address management mechanism[53]

图 21 RAIZN 地址管理机制[53]

Kim 等人[55] 提出了一种优化部分奇偶校验税的阵列 ZRAID，以进一步减少系统性能开销。ZRAID 在每个数据区内划分了一小部分作为 ZRWA 区域，将部分奇偶校验数据写入到此区域，不再将奇偶校验写入到专有区域，后续写入直接覆盖原数据块，从而，无需再汇总或迁移奇偶校验数据，避免了由于设置专用的奇偶校验区域引起的系统写入瓶颈，优化了系统的性能。

单设备的冗余校验优化研究[39] 针对 ZNS 固态硬盘内部的奇偶校验问题，Wang 等人[39] 在 FlexZNS 中引入了一种优化奇偶校验可靠性的机制，将奇偶校验区域与数据区域解耦，并将奇偶校验数据存放到专门设置的小区域内，当系统执行写入操作时，区域数据和奇偶校验数据各自顺序追加，分别写入到不同的区域中，互不干扰，从而降低了区域间的数据竞争，同时减轻了由于高频写入所引起的“热奇偶区”的性能开销。

基于 RAID 的冗余校验优化将多块 ZNS 固态硬盘构建为 RAID 阵列，通过引入 zone append，ZRWA 等 ZNS 新原语优化了跨盘无锁并行写[41,53-55]，并减少了奇偶校验数据的性能开销，同时提升了接口的兼容性。表 5 从接口类型、是否引入 ZNS 新原语和优化层次等方面对比了基于 ZNS 固态硬盘的 RAID 阵列方法。单设备的冗余校验优化研究在区域大小可变的 ZNS 固态硬盘内部进行奇偶校验、集中奇偶区域的研究[39] 将可靠性逻辑迁移至设备内部，降低了校验开销，提升了系统可靠性。

Table 5 Comparison of RAID Array Methods Based on ZNS SSDs

表 5 基于 ZNS 固态硬盘的 RAID 阵列方法对比

相关研究	接口类型	是否引入 ZNS 新原语	优化层次
RAIZN[53]	ZNS 接口	是	主机端
ZapRAID[41]	块接口	是	主机端
BIZA[54]	混合接口	是	主机端
ZRAID[55]	ZNS 接口	是	主机端

6 面向 ZNS 固态硬盘的模拟仿真器与性能优化辅助工具

ZNS 固态硬盘因其出色的性能日益受到研究者关注。为便于在不同配置条件下验证系统优化策略的正确性与有效性，并降低对昂贵硬件的依赖，研究人员通常需要基于模拟仿真器进行实验验证[49,91-93]。ZNS 固态硬盘的模拟仿真器不仅能够精确再现区域的顺序写入、重置等物理特性，支持对 ZNS 命令集的扩展与定制，还允许研究人员灵活配置各种设备参数(如区域大小、开放区域个数、介质特性)，以支持更全面的系统行为评估。此外，为进行系统级性能优化实验和分析，研究中还需使用具有运行时监控与性能优化的辅助工具[66,94-95]，以实现对存储系统行为的细粒度观察与调优。

针对上述需求，相关研究[49,66,91-95]分为 2 类：1) ZNS 固态硬盘模拟仿真器；2)性能优化辅助工具。

6.1 模拟仿真器

传统固态硬盘的模拟仿真器[96-100]大多支持模拟传统块接口设备，缺乏对于顺序写入、主机介质管理等新型存储范式的支持，难以模拟 ZNS 固态硬盘的相关硬件特性及设备配置(如区域大小、开放区域个数、介质特性)，因此，为支撑在不同系统配置与设计策略下对 ZNS 优化机制的全面评估，亟需构建具备 ZNS 协议语义、行为建模与参数调配能力的专用模拟仿真平台。

Li 等人[91]提出了一种支持 ZNS 固态硬盘的模拟仿真器 FEMU，以 QEMU 虚拟化技术为基础，将原有中断机制改为轮询实现，并以 RAM 作为后端存储介质，从而有效减少了客户端操作系统(guest os)与虚拟设备交互过程中退出以及 I/O 延迟等开销，提升了高并发场景下的仿真性能。FEMU 具有成本低、准确性高、可扩展性强等优势，为 ZNS 相关研究提供了灵活高效的实验平台。

Song 等人[92]在 FEMU[91]的基础上提出了一种面向 ZNS 固态硬盘的模拟仿真器 ConfZNS。该平台作为可配置的模拟器，主要包含 4 个模块，如图 22 所示，分别是 ZNS 配置、ZNS 架构管理、定时估计和资源监控。ZNS 配置模块用于管理用户配置 ZNS 固态硬盘的硬件参数，例如，区域大小、芯片数量等；ZNS 架构管理模块支持区域资源映射等相关操作，以实现各种 ZNS 固态硬盘架构的构建；定时估计模块通过统计排队延迟等信息预估 I/O 请求的整体延迟；资源监控模块则负责收集 ZNS 固态硬盘内部的统计数据。

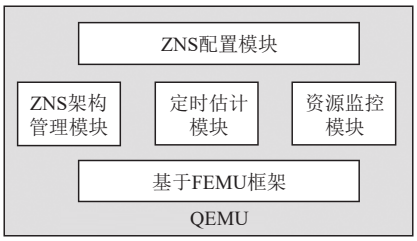


Fig. 22 ConfZNS architecture[92]

图 22 ConfZNS 架构[92]

Doekemeijer 等人[49]在 ConfZNS[92]的基础上，进一步提出了一种功能增强的 ZNS 固态硬盘的模拟仿真器 ConfZNS++，旨在减少后台 I/O 对前台 I/O 操作的影响。ConfZNS++可以定量分析 finish 和 reset 等后台管理操作对前台操作所产生的性能影响，通过引入 ZINC 调度机制(优先调度主机前台 I/O 操作)和 SoftFinish 机制(将设备 finish 操作部分移至主机端)来减少了后台管理 I/O 对前台操作的影响，为系统性能优化提供了支持。

Kim 等人[93]提出了一种基于 Linux 内核的设备仿真器 NVMeVirt，该仿真器以内核模块形式运行，在 PCIe 层对 NVMe 控制器进行虚拟化模拟，从而提供了一个可定制的虚拟 NVMe 驱动器环境。NVMeVirt 支持多种类型的固态硬盘的模拟仿真，包括传统固态硬盘、Optane 固态硬盘、ZNS 固态硬盘和 KV(key-value)固态硬盘等类型，为多种新型存储接口的功能验证与研究提供了平台支持。

6.2 性能优化辅助工具

为了支持基于 ZNS 固态硬盘的存储系统的性能优化，相关研究[66,94-95]对系统碎片优化、GC 优化等辅助分析工具进行了设计实现，为提升系统性能提供了有力的支持。

Qi 等人[66]提出了一种软硬件协同优化碎片整理的工具 InDeF，该工具根据系统的碎片化程度和数据访问频率，评估 I/O 数据的碎片整理优先级，仅对碎片化严重且访问较为频繁的数据执行整理，以减少

不必要的数据迁移。同时，为降低从设备内读取并写回数据所产生的性能开销，InDeF 将碎片整理过程中的有效数据迁移操作下发至设备内执行，从而改善了碎片整理过程中主机端和设备端的数据迁移和带宽占用等问题。

Chen 等人[94]提出了一种在主机端的碎片分析的工具 ZoneTrace，支持优化面向 ZNS 接口的文件系统中的空间利用与碎片分布。该工具利用扩展 Berkeley 包过滤器（extended Berkeley packet filter，eBPF）来实时监控文件系统的段位图变化，并结合 FIBMAP 文件映射信息实现对每个区域空间使用状态及碎片分布的分析，达到支持空间管理优化的目的。

Bergman 等人[95]针对 ZNS 固态硬盘通过交换分区（swap）进行 GC 所引起的性能开销问题，提出了一种 GC 分析工具 ZNSwap，实现在主机端进行 GC，根据内核中空闲 swap 页等信息对 GC 中的区域选择和清理时机选择提供了支持，使 swap 数据布局更加有序，将原本不可控的设备内部的 GC 转化为主机端可控的操作，以达到支持系统性能优化的目的。

目前，基于 ZNS 固态硬盘的大量系统研究是在模拟仿真器上进行实验验证，并且已发表在多个高水平会议和期刊[17,19,23,29,33-34,50,66,101-102]。表 6 总结了基于 ZNS 固态硬盘的模拟仿真器与性能优化辅助工具的研究分类和方法。在模拟仿真研究中，目前最主流的模拟仿真器是 FEMU[91]和 NVMeVirt[93]，很多研究都是基于这 2 种模拟仿真器进行的实验评估。而 ConfZNS 和 ConfZNS++作为在特定环境下的模拟仿真器，能够有效进行更细粒度的数据分析，对 ZNS 固态硬盘的性能优化分析提供了一定的支持。InDeF，ZoneTrace，ZNSwap 作为性能优化辅助工具的研究，支持分析主机端与 ZNS 设备交互过程中不同层次的工作负载特征和设备利用情况，从而为进一步优化基于 ZNS 固态硬盘的系统提供了支持。

Table 6 Simulators and Performance-Tuning for ZNS SSDs

表 6 模拟仿真器与性能优化 ZNS 固态硬盘

研究分类	相关研究
模拟仿真器	FEMU[91]，ConfZNS[92]，ConfZNS++[49]，NVMeVirt[93]
性能优化辅助工具	InDeF[66]，ZoneTrace[94]，ZNSwap[95]

7 总结与展望

本文首先从 ZNS 固态硬盘的特性和存储范式出发，系统分析了基于 ZNS 固态硬盘的存储系统所面临的挑战。然后，对各类挑战相关的优化技术、研究进展和典型研究进行了归纳、分析和评述。同时，对现有面向 ZNS 固态硬盘的模拟仿真器与性能优化辅助工具进行了归纳和总结。

为了更加充分释放固态硬盘的并行性与介质潜力，并持续降低写入放大与 GC 引起的性能开销，未来固态硬盘将逐步摆脱传统“黑盒 FTL”抽象，以“可编程”的方式向主机端开放数据放置等相关原语，使主机端能够在不同控制约束下参与物理布局优化。在此过程中，存储系统可根据工作负载特点与改造成本，选择不同粒度的设备控制权，例如，通过 FDP（flexible data placement）[103-106]向设备提供灵活的数据放置提示，或由 ZNS 同时管理数据放置与 GC 过程，以进一步提升系统性能。然而，由于控制权开放得越充分往往意味着越高的系统改造代价，而更强的生态兼容性又通常限制主机端可施加的放置调度空间，因此“控制权开放程度”与“生态兼容性”之间的权衡将长期存在，并持续影响未来固态硬盘的优化路径与发展方向。

ZNS 固态硬盘尽管需要在主机端实现较多的存储管理，存在较多的软件改造成本，但其通过顺序写入和区域管理，很大程度降低了传统固态硬盘的“块接口税”，展现出良好的性能优化潜力。结合现有研究进展与发展态势来看，跨越存储设备、系统软件与应用的多层次协同优化有望成为未来的研究热点，具体涉及如下 4 个方面：

1）跨层协同设计。ZNS 固态硬盘将地址映射、GC 等存储管理操作从设备端上移到主机端，为系统优化提供了更好的支持。然而，仅在某一层面进行优化仍难以充分发挥 ZNS 固态硬盘的性能优势，在应用层、操作系统层以及硬件层等相关层联合进行优化将避免传统系统中分层隔离、“各自为战”的弊端。例如，ZNS+[27]将文件系统的段压缩过程迁移到设备端，避免了主机端产生的性能开销；Balloon-ZNS[101]通过主机端和设备端的协同，优化了 ZNS 固态硬盘的压缩（compress）机制，避免了在压缩过程中产生的资源浪费，有效提升了系统性能；CCZNS[102]在主机端引入了特定的读/写命令，并将 compress 操作下发到设备内以实现跨层管理，进而提升了系统的吞吐量；Z-CacheLib[107]则通过跨层的映射和缓存机制优化，有效提高了存储命中率并减少了写入放大。上述研究显示出跨应用层、系统层以及设备层联合进行优化的潜力。如何更好地进行跨层协同设

计,包括如何更好地实现信息跨层共享、负载跨层分配、策略跨层实施,这些问题都值得进一步深入研究。

2)多层次异构集成。随着非易失性存储介质、硬件架构的持续发展,存储硬件呈现多元发展的特点,单一种类的存储设备难以兼顾性能、容量和成本等多方面需求,因此,基于不同类型、架构的存储介质/设备所构建的存储系统将成为研究热点。对于ZNS固态硬盘而言,将其优势融入异构存储体系以兼顾实现高性能与大容量显得尤为必要。例如,ZnH_2[19]将ZNS固态硬盘的异构区域(SLC和QLC介质)与主机管理相结合,将热数据优先写入高性能的SLC介质区域,而将冷数据存储在大容量的QLC介质区域,有效减少了写入放大;Yang等人[108]根据ZNS固态硬盘的硬件特性,将其作为二级缓存,并与其他存储设备相结合,有效降低了系统尾延迟并提升了整体吞吐量。上述研究表明,在异构存储环境中充分发挥ZNS固态硬盘的优势能够有效提升系统性能并降低写入放大等开销。如何在异构环境中更好地发挥ZNS固态硬盘的性能特性,如何实现ZNS固态硬盘与持久内存、不同闪存介质固态硬盘以及传统硬盘等设备的高效协同,这些将成为推动系统架构演进的重要方向。

3)软件堆栈优化。传统操作系统以块设备接口作为默认抽象模型,缺乏针对ZNS固态硬盘的特性(顺序写入约束和区域管理)的支持,无法充分释放其硬件潜力,并且可能产生额外的性能开销和兼容性等问题。为了充分发挥ZNS固态硬盘的硬件能力,改造传统的存储软件堆栈,将ZNS新原语(如zone append, ZRWA等)引入到上层存储堆栈将是重要的研究方向。例如,WALTZ[40], ZapRAID[41]通过引入ZNS新原语对LSM-tree的键值存储系统进行并行写入优化,有效提升了系统并行性。此外,新型高性能存储框架(如io_uring[109], SPDK[110])通过优化传统存储堆栈I/O接口或内核旁路机制,与面向ZNS接口的软件栈相结合,有效降低了性能开销。这些研究表明,面向ZNS接口的存储软件栈优化能够显著减少系统开销,提升系统性能。如何针对ZNS固态硬盘的特性重新设计和优化存储软件堆栈,如何将zone append等ZNS原语以及io_uring, SPDK等高性能I/O框架更紧密地融入上层存储系统,这些都有待进一步探索。

4)系统全局优化。在基于ZNS固态硬盘构建的存储系统中,主机端需要同时进行GC、磨损均衡、顺序写入等多方面的操作,然而,由于上述存储管理操作间可能存在相互干扰,对系统性能将产生一定的影响。需要从全局层面探索冷热数据分离、自适应数据布局等优化技术。例如,Prophet[18]通过评估刷新与compaction过程来预测SST数据文件的生命周期,在此基础上提出了新的数据布局与分配策略,并在GC过程中融合compaction操作,更好地平衡了系统性能与写入放大之间的关系。上述研究表明,采用全局视角统筹刷新、compaction和GC等操作能够有效提升系统性能并降低写入放大。如何在全局层面实现更高效的冷热数据分离和自适应数据布局,如何利用强化学习等人工智能技术设计智能的全局优化算法,从而更好地保障ZNS固态硬盘系统的高效运行,这些问题都值得进一步深入研究和探索。

作者贡献声明:朱春晖负责文献调研、论文结构设计、论文撰写以及修改;陶秋铭参与论文的结构设计、指导论文修改;周启明和李明树参与论文审阅和修改指导。

参 考 文 献

[1] SNIA. Zoned storage models v1.0[EB/OL]. [2025-08-05]. https://www.snia.org/sites/default/files/technical-work/zoned/release/Zoned-Storage-Models-v1.0.pdf

[2] Bjørling M, Aghayev A, Holmberg H, et al. ZNS: Avoiding the block interface tax for flash-based SSDs[C]//Proc of the 2021 USENIX Annual Technical Conf (USENIX ATC 2021). Berkeley, CA: USENIX Association, 2021: 689−703

[3] NVM Express. NVMe zoned namespaces (ZNS) command set specification[EB/OL]. [2025-08-10]. https://nvmexpress.org/specification/nvme-zoned-namespaces-zns-command-set-specification/

[4] NVM Express. NVM Express Q3 webcast Q&A: Answering your questions about NVMe zoned namespace SSDs and the Linux zoned storage ecosystem[EB/OL]. [2025-06-06]. https://nvmexpress.org/nvm-express-q3-webcast-qa-answering-your-questions-about-nvme-zoned-namespace-ssds-and-the-linux-zoned-storage-ecosystem/

[5] SNIA. Zoned namespaces (ZNS) SSDs: Disrupting the storage industry[EB/OL]. [2025-06-06]. https://www.snia.org/educational-library/zoned-namespaces-zns-ssds-disrupting-storage-industry-2020

[6] Western Digital. Ultrastar DC ZN540: Zoned storage for the zettabyte era[EB/OL]. [2025-08-06]. https://documents.westerndigital.com/content/dam/doc-library/en_us/assets/public/westerndigital/collateral/data-sheet/data-sheet-ultrastar-dc-zn540.pdf

[7] Samsung. Samsung PM1743a[EB/OL]. [2025-08-06]. https://semiconductor.samsung.com/news-events/news/samsung-introduces-its-first-zns-ssd-with-maximized-user-capacity-and-enhanced-lifespan/

[8] DapuStor. DapuStor H3200[EB/OL]. [2025-08-06]. https://en.dapu-

stor.com/product/6.html

[9] HPCwire. Inspur launches a new generation of enterprise NVMe SSD[EB/OL]. [2025-08-06]. https://www.hpcwire.com/off-the-wire/inspur-launches-a-new-generation-of-enterprise-nvme-ssd/

[10] Bae H, Kim J, Kwon M, et al. What you can't forget: Exploiting parallelism for zoned namespaces[C]//Proc of the 14th ACM Workshop on Hot Topics in Storage and File Systems (HotStorage 2022). New York: ACM, 2022: 79−85

[11] Maheshwari U. From blocks to rocks: A natural extension of zoned namespaces[C]//Proc of the 13th ACM Workshop on Hot Topics in Storage and File Systems (HotStorage 2021). New York: ACM, 2021: 21−27

[12] Chung S. Benefits of ZNS in datacenter storage systems[EB/OL]. [2025-07-06]. https://www.flashmemorysummit.com/Proceedings-2019/08-06-Tuesday/20190806_ARCH-102-1_Chung.pdf

[13] Tehrany N, Trivedi A. Understanding NVMe zoned namespaces (ZNS) flash SSD storage devices[J]. arXiv Preprint, arXiv: 2206.01547, 2022

[14] Zhang Xiangqun, Bhimani J, Pei Shuyi, et al. Storage abstractions for SSDs: The past, present, and future[J]. ACM Transactions on Storage, 2025, 21(1): 1−44

[15] Xu Yishun, Jin Peiquan, Lu Mingchen, et al. LeanKV: Efficient garbage collection for LSM-based key-value stores on ZNS SSDs through lifetime-based SSTable clustering[C]//Proc of the 29th IEEE Int Conf on Parallel and Distributed Systems (ICPADS 2023). Piscataway, NJ: IEEE, 2023: 1895−1902

[16] Lu Mingchen, Jin Peiquan, Wang Xiaoliang, et al. ZoneKV: A space-efficient key-value store for ZNS SSDs[C]//Proc of the 60th Annual ACM/IEEE Design Automation Conf (DAC 2023). Piscataway, NJ: IEEE, 2023: 1−6

[17] Shen Jingcheng, Yang Lang, Long Linbo, et al. Overlapping-aware zone allocation for LSM-tree-based store on ZNS SSDs[C]//Proc of the 29th Asia and South Pacific Design Automation Conf (ASP-DAC 2024). Piscataway, NJ: IEEE, 2024: 448−453

[18] Liu Gaoji, Yang Chongzhuo, Yu Qiaolin, et al. Prophet: Optimizing LSM-based key-value store on ZNS SSDs with file lifetime prediction and compaction compensation[C/OL]//Proc of the 38th Int Conf on Massive Storage Systems and Technology (MSST 2024). 2024[2025-08-02]. https://www.msstconference.org/MSST-history/2024/Papers/msst24-3.1.pdf

[19] Wang Yingjia, Chow L Y, Nie Xirui, et al. ZnH_2: Augmenting ZNS-based storage system with host-managed heterogeneous zones[C]//Proc of the 43rd IEEE/ACM Int Conf on Computer-Aided Design (ICCAD 2024). New York: ACM, 2024: 1−8

[20] Jung J, Shin D. Lifetime-leveling LSM-tree compaction for ZNS SSD[C]//Proc of the 14th ACM Workshop on Hot Topics in Storage and File Systems (HotStorage 2022). New York: ACM, 2022: 100−105

[21] Liu Biyong, Xia Yuan, Wei Xueliang, et al. LifetimeKV: Narrowing the lifetime gap of SSTs in LSMT-based KV stores for ZNS SSDs[C]//Proc of the 41st IEEE Int Conf on Computer Design (ICCD 2023). Piscataway, NJ: IEEE, 2023: 300−307

[22] Lee H, Lee C, Lee S, et al. Compaction-aware zone allocation for LSM-based key-value store on ZNS SSDs[C]//Proc of the 14th ACM Workshop on Hot Topics in Storage and File Systems (HotStorage 2022). New York: ACM, 2022: 93−99

[23] Byeon S, Ro J, Han J Y, et al. Ensuring compaction and zone cleaning efficiency through same-zone compaction in ZNS key-value store[C/OL]//Proc of the 38th Int Conf on Massive Storage Systems and Technology (MSST 2024). 2024[2025-07-25]. https://www.mss-tconference.org/ MSST-history/2024/Papers/msst24-9.2.pdf

[24] Wu Denghui, Liu Biyong, Zhao Wei, et al. ZNSKV: Reducing data migration in LSMT-based KV stores on ZNS SSDs[C]//Proc of the 40th IEEE Int Conf on Computer Design (ICCD 2022). Piscataway, NJ: IEEE, 2022: 411−414

[25] Liang Yuhong, Wang Yingjia, Yang T Y, et al. ZonesDB: Building write-optimized and space-adaptive key-value store on zoned storage with fragmented LSM tree[J]. ACM Transactions on Storage, 2025, 21(2): 1−24

[26] Hwangbo H, Ro J, Byeon S, et al. Towards a unified garbage collection strategy in ZNS key-value store file systems using same-victim GC[C]//Proc of the 32nd IEEE Int Symp on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS 2024). Piscataway, NJ: IEEE, 2024: 1−8

[27] Han K, Gwak H, Shin D, et al. ZNS+: Advanced zoned namespace interface for supporting in-storage zone compaction[C]//Proc of the 15th USENIX Symp on Operating Systems Design and Implementation (OSDI 2021). Berkeley, CA: USENIX Association, 2021: 147−162

[28] Choi G, Lee K, Oh M, et al. A new LSM-style garbage collection scheme for ZNS SSDs[C]//Proc of the 12th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 2020). New York: ACM, 2020: 1−6

[29] Huang Dong, Feng Dan, Liu Qiankun, et al. SplitZNS: Towards an efficient LSM-tree on zoned namespace SSDs[J]. ACM Transactions on Architecture and Code Optimization, 2023, 20(3): 1−26

[30] Liu Renping, Chen Junhua, Chen Peng, et al. Hi-ZNS: High space efficiency and zero-copy LSM-tree-based stores on ZNS SSDs[C]//Proc of the 53rd Int Conf on Parallel Processing (ICPP 2024). New York: ACM, 2024: 1217−1226

[31] Liu Yang, Jin Peiquan. ZB^+-tree: A novel ZNS SSD-aware index structure[J]. Journal of Computer Research and Development, 2023, 60(3): 509−524 (in Chinese)
(刘扬, 金培权. ZB+-tree: 一种 ZNS SSD 感知的新型索引结构[J]. 计算机研究与发展, 2023, 60(3): 509−524)

[32] Byeon S, Ro J, Jamil S, et al. A free-space adaptive runtime zone-reset algorithm for enhanced ZNS efficiency[C]//Proc of the 15th ACM Workshop on Hot Topics in Storage and File Systems (HotStorage 2023). New York: ACM, 2023: 109−115

[33] Liu Renping, Xiao Xiao, Zou Yang, et al. ZNS-Cleaner: Enhancing lifespan by reducing empty erase in ZNS SSDs[J]. Journal of Systems Architecture, 2024, 157: 103303

[34] Tan Zhenhua, Long Linbo, Shen Jingcheng, et al. Optimizing garbage collection for ZNS SSDs via in-storage data migration and address remapping[J]. ACM Transactions on Architecture and Code Optimization, 2024, 21(4): 1−25

[35] Im M, Kang K, Yeom H. Accelerating RocksDB for small-zone ZNS SSDs by parallel I/O mechanism[C]//Proc of the 23rd Int Middleware Conf Industry Track (Middleware 2022). New York: ACM, 2022: 15−21

[36] Choi J, Jun Y, Kim J, et al. An adaptive zone-grouping scheme enabling general-purpose file systems on ZNS SSDs[C]//Proc of the 17th ACM Int Systems and Storage Conf (SYSTOR 2024). New York: ACM, 2024: 132−145

[37] Tan Zhenhua, Long Linbo, Shen Jingcheng, et al. Para-ZNS: Improving small-zone ZNS SSD parallelism through dynamic zone mapping[C]//Proc of the 2024 Design, Automation & Test in Europe Conf (DATE 2024). Piscataway, NJ: IEEE, 2024: 1−6

[38] Min J, Zhao Chenxingyu, Liu Ming, et al. eZNS: An elastic zoned namespace for commodity ZNS SSDs[C]//Proc of the 17th USENIX Symp on Operating Systems Design and Implementation (OSDI 2023). Berkeley, CA: USENIX Association, 2023: 461−477

[39] Wang Yu, Zhou You, Lu Zhonghai, et al. FlexZNS: Building high-performance ZNS SSDs with size-flexible and parity-protected zones[C]//Proc of the 41st IEEE Int Conf on Computer Design (ICCD 2023). Piscataway, NJ: IEEE, 2023: 291−299

[40] Lee J, Kim D U, Lee J W. WALTZ: Leveraging zone append to tighten the tail latency of LSM tree on ZNS SSD[J]. Proceedings of the VLDB Endowment, 2023, 16(11): 2884−2896

[41] Wang Qiuping, Lee P P C. ZapRAID: Toward high-performance RAID for ZNS SSDs via zone append[C]//Proc of the 14th ACM SIGOPS Asia-Pacific Workshop on Systems (APSys 2023). New York: ACM, 2023: 24−29

[42] Wang Yu, Zhou You, Wu Fan, et al. Land of Oz: Resolving orderless writes in zoned namespace SSDs[J]. IEEE Transactions on Computers, 2024, 73(11): 2520−2533

[43] Long Linbo, Dong Xiaoyu, Wu Tao, et al. Optimizing F2FS performance with the inter-zone parallelism in small-zone ZNS SSDs[J]. Future Generation Computer Systems, 2026, 175: 108044

[44] Zhang Jintong, Hu Haichuan, Chen Jianxi, et al. SchInFS: A file system integrating functions of the block I/O scheduler for ZNS SSDs[C]//Proc of the 42nd IEEE Int Conf on Computer Design (ICCD 2024). Piscataway, NJ: IEEE, 2024: 324−331

[45] Lv Yanqi, Jin Peiquan, Wang Xiaoliang, et al. ZonedStore: A concurrent ZNS-aware cache system for cloud data storage[C]//Proc of the 42nd IEEE Int Conf on Distributed Computing Systems (ICDCS 2022). Piscataway, NJ: IEEE, 2022: 1322−1325

[46] Han Y, Oh M, Lee J, et al. Achieving performance isolation in Docker environments with ZNS SSDs[C]//Proc of the 12th IEEE Non-Volatile Memory Systems and Applications Symp (NVMSA 2023). Piscataway, NJ: IEEE, 2023: 25−31

[47] Liu Renping, Tan Zhenhua, Shen Yan, et al. Fair-ZNS: Enhancing fairness in ZNS SSDs through self-balancing I/O scheduling[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2024, 43(7): 2012−2022

[48] Zhu Weilin, Tong Wei. Turn waste into wealth: Alleviating read/write interference in ZNS SSDs[C]//Proc of the 41st IEEE Int Conf on Computer Design (ICCD 2023). Piscataway, NJ: IEEE, 2023: 316−319

[49] Doekemeijer K, Maisenbacher D, Ren Z, et al. Exploring I/O management performance in ZNS with ConfZNS++[C]//Proc of the 17th ACM Int Systems and Storage Conf (SYSTOR 2024). New York: ACM, 2024: 162−177

[50] Long Linbo, He Shuiyong, Shen Jingcheng, et al. WA-Zone: Wear-aware zone management optimization for LSM-tree on ZNS SSDs[J]. ACM Transactions on Architecture and Code Optimization, 2024, 21(1): 1−23

[51] Qi Wenjie, Tan Zhipeng, Zhang Ziyue, et al. BlzFS: Crash consistent log-structured file system based on byte-loggable zone for ZNS SSD[C]//Proc of the 41st IEEE Int Conf on Computer Design (ICCD 2023). Piscataway, NJ: IEEE, 2023: 206−213

[52] Lee E, Son I, Kim J. An efficient order-preserving recovery for F2FS with ZNS SSD[C]//Proc of the 15th ACM Workshop on Hot Topics in Storage and File Systems (HotStorage 2023). New York: ACM, 2023: 116−122

[53] Kim T, Jeon J, Arora N, et al. RAIZN: Redundant array of independent zoned namespaces[C]//Proc of the 28th ACM Int Conf on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2023). New York: ACM, 2023: 660−673

[54] Yi Shushu, Sun Shaocong, Peng Li, et al. BIZA: Design of self-governing block-interface ZNS AFA for endurance and performance[C]//Proc of the 30th ACM SIGOPS Symp on Operating Systems Principles (SOSP 2024). New York: ACM, 2024: 313−329

[55] Kim M, Jeong S, Kim J. ZRAID: Leveraging zone random write area (ZRWA) for alleviating partial parity tax in ZNS RAID[C]//Proc of the 30th ACM Int Conf on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2025). New York: ACM, 2025: 1151−1165

[56] Doekemeijer K, Tehrany N, Chandrasekaran B, et al. Performance characterization of NVMe flash devices with zoned namespaces (ZNS)[C]//Proc of the 25th IEEE Int Conf on Cluster Computing (CLUSTER 2023). Piscataway, NJ: IEEE, 2023: 118−131

[57] Lerner A, Bonnet P. Not your grandpa's SSD: The era of co-designed storage devices[C]//Proc of the 2021 Int Conf on Management of Data (SIGMOD 2021). New York: ACM, 2021: 2852−2858

[58] Bjørling M. From open-channel SSDs to zoned namespaces[C/OL]//Proc of the 2019 Linux Storage and Filesystems Conf (Vault '2019). Berkeley, CA: USENIX Association, 2019[2025-06-21]. https://www.usenix.org/conference/vault19/presentation/bjorling

[59] Bjørling M, Gonzalez J, Bonnet P. LightNVM: The Linux open-channel SSD subsystem[C]//Proc of the 15th USENIX Conf on File and Storage Technologies (FAST 2017). Berkeley, CA: USENIX Association, 2017: 359−374

[60] Lee S, Liu M, Jun S, et al. Application-managed flash[C]//Proc of the 14th USENIX Conf on File and Storage Technologies (FAST 2016). Berkeley, CA: USENIX Association, 2016: 339−353

[61] Ouyang Jian, Lin Shiding, Jiang Song, et al. SDF: Software-defined flash for web-scale Internet storage systems[C]//Proc of the 19th ACM Int Conf on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2014). New York: ACM, 2014: 471−484

[62] Wang Peng, Sun Guangyu, Jiang Song, et al. An efficient design and

implementation of LSM-tree-based key-value store on open-channel SSD[C]//Proc of the 9th European Conf on Computer Systems (EuroSys 2014). New York: ACM, 2014: 1−14

[63] INCITS T10 Technical Committee. Draft Standard T10/BSR INCITS 536 Information Technology—Zoned Block Commands (ZBC)[S]. Washington: American National Standards Institute, 2014

[64] INCITS T13 Technical Committee. Draft Standard T13/BSR INCITS 537 Information Technology—Zoned Device ATA Command Set (ZAC)[S]. Washington: American National Standards Institute, 2015

[65] Stavrinos T, Berger D S, Katz-Bassett E, et al. Don't be a blockhead: Zoned namespaces make work on conventional SSDs obsolete[C]// Proc of the 18th Workshop on Hot Topics in Operating Systems (HotOS 2021). New York: ACM, 2021: 144−151

[66] Qi Wenjie, Tan Zhipeng, Shao Jicheng, et al. InDeF: An advanced defragmenter supporting migration offloading on ZNS SSD[C]//Proc of the 40th IEEE Int Conf on Computer Design (ICCD 2022). Piscataway, NJ: IEEE, 2022: 307−314

[67] Bjørling M. zone append: A new way of writing to zoned storage [C/OL]//Proc of the 2020 Linux Storage and File Systems Conf (Vault 2020). Berkeley, CA: USENIX Association, 2020[2025-07-21]. https://www.usenix.org/conference/vault20/presentation/bjorling

[68] Holmberg H. Accelerating RocksDB with NVMe zoned SSDs[EB/OL]. [2025-09-25]. https://www.snia.org/sites/default/files/SDC/2019/presentations/NVMe/Holmberg_Hans_Accelerating_RocksDB_with_Zoned_Namespaces.pdf

[69] Holmberg H. ZenFS, Zones and RocksDB - Who likes to take out the GarbageAnyway?[EB/OL]. [2025-07-23]. https://www.snia.org/educational-library/zenfs-zones-and-rocksdb-who-likes-take-out-garbage-anyway-2020

[70] Bjørling M. Zoned namespaces in practice[EB/OL]. [2025-12-25]. https://files.futurememorystorage.com/proceedings/2019/08-06-Tuesday/20190806_ARCH-102-1_Bjorling.pdf

[71] Gonzalez J. Zoned namespaces: Use cases, standard and Linux ecosystem[EB/OL].[2025-09-25].https://www.snia.org/sites/default/files/SDCEMEA/2020/3%20-%20Javier%20Gonzalez%20Zoned%−20namespacese.pdf

[72] Le M D, Yao Ting. zonefs: Mapping the posix file system interface to raw zoned block device accesses[C/OL]//Proc of the 2020 Linux Storage and File Systems Conf (Vault 2020). Berkeley, CA: USENIX Association, 2020[2025-08-21]. https://www.usenix.org/conference/vault20/presentation/lemoal

[73] Gonzalez J. Zoned namespaces: Standardization and Linux ecosystem[EB/OL].[2025-08-05].https://www.youtube.com/watch?v=5hdUHSILkzA

[74] Linux Kernel. mdraid layer[EB/OL]. [2025-09-01]. https://github.com/torvalds/linux/tree/master/drivers/md

[75] Le M D. dm-zoned: Zoned block device device mapper[EB/OL]. [2025-09-03]. https://lwn.net/Articles/714387/

[76] Holmberg H. dm-zap: Host-based FTL for ZNS SSDs[EB/OL]. [2025-09-03]. https://github.com/westerndigitalcorporation/dm-zap

[77] Cho Y, Park S, Han H. Performance analysis of ZNS-aware file systems on distributed applications[C]//Proc of the 43rd IEEE Int Conf on Distributed Computing Systems (ICDCS 2023). Piscataway, NJ: IEEE, 2023: 1047−1048

[78] Josephson W K, Bongo L A, Li Kai, et al. DFS: A file system for virtualized flash storage[J]. ACM Transactions on Storage, 2010, 6(3): 1−25

[79] Zoned Storage. zonefs file system[EB/OL]. [2025-07-23]. https://zonedstorage.io/docs/filesystems/zonefs

[80] Lee C, Sim D, Hwang J, et al. F2FS: A new file system for flash storage[C]//Proc of the 13th USENIX Conf on File and Storage Technologies (FAST 2015). Berkeley, CA: USENIX Association, 2015: 273−286

[81] Le M D. f2fs: Zoned block device support[EB/OL]. [2025-08-12]. https://www.spinics.net/lists/linux-fsdevel/msg103443.html

[82] Yang Jingpei, Plasson N, Gillis G, et al. Don't stack your log on my log[C]//Proc of the 2nd Workshop on Interactions of NVM/Flash with Operating Systems and Workloads (INFLOW 2014). Berkeley, CA: USENIX Association, 2014: 1−10

[83] Rodeh O, Bacik J, Mason C. BTRFS: The Linux B-tree filesystem[J]. ACM Transactions on Storage, 2013, 9(3): 1−32

[84] Aota N. btrfs: Zoned block device support[EB/OL]. [2025-08-12]. https://lwn.net/Articles/836726/

[85] Berg B, Berger D S, McAllister S, et al. The CacheLib caching engine: Design and experiences at scale[C]//Proc of the 14th USENIX Symp on Operating Systems Design and Implementation (OSDI 2020). Berkeley, CA: USENIX Association, 2020: 753−768

[86] Seastar Project. Ceph SeaStore optimizes for NVMe SSDs[EB/OL]. [2025-08-21]. https://docs.ceph.com/en/latest/dev/seastore/

[87] Facebook. RocksDB[EB/OL]. [2025-08-11]. https://rocksdb.org/

[88] Western Digital Corporation. ZenFS[EB/OL]. [2025-08-06]. https://github.com/westerndigitalcorporation/zenfs

[89] Google. LevelDB[EB/OL]. [2025-08-06]. https://github.com/google/leveldb

[90] Jin Hai, Zhou Xinrong, Feng Dan, et al. Improving partial stripe write performance in RAID level 5[C]//Proc of the 2nd IEEE Int Caracas Conf on Devices, Circuits and Systems (ICCDCS 1998). Piscataway, NJ: IEEE, 1998: 396−400

[91] Li Huaicheng, Hao Mingzhe, Tong M H, et al. The case of FEMU: Cheap, accurate, scalable and extensible flash emulator[C]//Proc of the 16th USENIX Conf on File and Storage Technologies (FAST 2018). Berkeley, CA: USENIX Association, 2018: 83−90

[92] Song I, Oh M, Kim B S J, et al. ConfZNS: A novel emulator for exploring design space of ZNS SSDs[C]//Proc of the 16th ACM Int Systems and Storage Conf (SYSTOR 2023). New York: ACM, 2023: 71−82

[93] Kim S, Shim J, Lee E, et al. NVMeVirt: A versatile software-defined virtual NVMe device[C]//Proc of the 21st USENIX Conf on File and Storage Technologies (FAST 2023). Berkeley, CA: USENIX Association, 2023: 379−394

[94] Chen P X, Seo D, Sung C, et al. ZoneTrace: Zone monitoring tool for F2FS on ZNS SSDs[J]. ACM Transactions on Design Automation of Electronic Systems, 2024, 29(5): 1−15

[95] Bergman S, Cassel N, Bjørling M, et al. ZNSwap: Un-block your swap[C]//Proc of the 2022 USENIX Annual Technical Conf (USENIX ATC 2022). Berkeley, CA: USENIX Association, 2022:

1−18

[96] Gouk D, Kwon M, Zhang Jie, et al. Amber: Enabling precise full-system simulation with detailed modeling of all SSD resources[C]// Proc of the 51st IEEE/ACM Int Symp on Microarchitecture (MICRO 2018). Piscataway, NJ: IEEE, 2018: 469−481

[97] Yoo J, Won Y, Hwang J, et al. VSSIM: Virtual machine based SSD simulator[C]//Proc of the 29th IEEE Symp on Mass Storage Systems and Technologies (MSST 2013). Piscataway, NJ: IEEE, 2013: 1−14

[98] Agrawal N, Prabhakaran V, Wobber T, et al. Design tradeoffs for SSD performance[C]//Proc of the 2008 USENIX Annual Technical Conf (USENIX ATC 2008). Berkeley, CA: USENIX Association, 2008: 57−70

[99] Zhang Yiying, Arulraj L P, Arpaci-Dusseau A C, et al. De-indirection for flash-based SSDs with nameless writes[C]//Proc of the 10th USENIX Conf on File and Storage Technologies (FAST 2012). Berkeley, CA: USENIX Association, 2012: 1−16

[100] Bucy J S, Ganger G R. The DiskSim simulation environment version 3.0 reference manual[R]. Pittsburgh, PA: School of Computer Science, Carnegie Mellon University, 2003

[101] Wang Yu, Sun Zibin, Zhou You, et al. Balloon-ZNS: Constructing high-capacity and low-cost ZNS SSDs with built-in compression[C]// Proc of the 61st ACM/IEEE Design Automation Conf (DAC 2024). New York: ACM, 2024: 125: 1−6

[102] Wang Yingjia, Lu Tao, Liang Yuhong, et al. Reviving in-storage hardware compression on ZNS SSDs through host-SSD collaboration[C]//Proc of the 2025 IEEE Int Symp on High Performance Computer Architecture (HPCA 2025). Piscataway, NJ: IEEE, 2025: 608−623

[103] NVM Express, Inc. NVM Express Revision 2.1 Changes[S]. Beaverton, OR: NVM Express, 2024

[104] Allison M, George A, González J, et al. Towards efficient flash caches with emerging NVMe flexible data placement SSDs[C]//Proc of the 20th European Conf on Computer Systems (EuroSys 2025). New York: ACM, 2025: 1142−1160

[105] Chen P X, Seo D, Dutt N. FDPFS: Leveraging file system abstraction for FDP SSD data placement[J]. IEEE Embedded Systems Letters, 2024, 16(4): 349−352

[106] Park J, Kim H, Ha J, et al. FDPVirt: Flexible data placement SSD emulator[C]//Proc of the 2024 IEEE Int Conf on Cluster Computing Workshops (CLUSTER Workshops 2024). Piscataway, NJ: IEEE, 2024: 198−199

[107] Yang Chongzhuo, Guo Chang, Zhao Ming, et al. A zoned storage optimized flash cache on ZNS SSDs[J]. arXiv preprint, arXiv: 2410.11260, 2024

[108] Yang Chongzhuo, Cao Zhang, Guo Chang, et al. Can ZNS SSDs be better storage devices for persistent cache?[C]//Proc of the 16th ACM Workshop on Hot Topics in Storage and File Systems (HotStorage 2024). New York: ACM, 2024: 55−62

[109] Axboe J. liburing[EB/OL]. [2025-08-06]. https://github.com/axboe/liburing

[110] Malikowski W. Storage performance development kit (SPDK) flash translation layer library for zoned namespace SSDs[EB/OL]. [2025-08-26]. https://www.flashmemorysummit.com/Proceedings2019/08-06-Tuesday/20190806_ARCH-102-1_Malikowski.pdf



Zhu Chunhui, born in 1986. PhD candidate. Student member of CCF. His main research interests include operating system, storage system, and software engineering.

朱春晖, 1986 年生。博士研究生。CCF 学生会员。主要研究方向为操作系统、存储系统、软件工程。



Tao Qiuming, born in 1979. PhD, associate professor. Member of CCF. His main research interests include operating system, compilation technology, and software engineering.

陶秋铭, 1979 年生。博士，副研究员。CCF 会员。主要研究方向为操作系统、编译技术、软件工程。



Zhou Qiming, born in 1973. Senior engineer. Her main research interests include operating systems and software engineering.

周启明, 1973 年生。高级工程师。主要研究方向为操作系统、软件工程。



Li Mingshu, born in 1966. PhD, professor, PhD supervisor. Fellow of CCF. His main research interests include operating systems, central processing unit, and fundamental software and hardware technologies.

李明树, 1966 年生。博士，研究员，博士生导师。CCF 会士。主要研究方向为操作系统、中央处理器以及软硬件深度融合和协同设计。