基于 ZNS 并行性的探索

柴若愚 1)

1) (华中科技大学计算机科学与技术学院，武汉 430074)

摘 要 分区命名空间 (ZNS) 是一种新兴的存储接口，它可以匹配数据写入设备端后端闪存的方式，同时隐藏闪存特定管理的细节。具体来说，ZNS 提供了一个区域概念，每个区域都映射到一个或多个闪存物理块，并将它们的操作约束暴露给主机。与此同时，现有研究发现，随着区域容量的下降，ZNS 接口的产生在一定程度上会剥夺 SSD 的内部并行性，从而降低效率。同时，一些研究也将关注点放在了内部并行性上，从而对 ZNS-SSD 做出了一些改进。本文整合现有的有关 ZNS 并行性的研究以及在相应领域的改进并加以介绍。

关键词 分区命名空间；内部并行性；区域间干扰

Exploration of parallelism based on ZNS

Ruoyu Chai[1)]

[1)](Department of Computer Science and Technology, Huazhong University of Science and Technology, WuHan 430074, China)

Abstract Zoned namespace (ZNS) is a new storage interface, which can match the way data is written to the back-end flash memory on the device side, while hiding the details of specific flash management. Specifically, ZNS provides a region concept. Each region maps to one or more flash physical blocks and exposes their operational constraints to the host. At the same time, existing studies have found that, with the decline of regional capacity, the generation of ZNS interfaces will deprive SSDs of internal parallelism to some extent, thereby reducing efficiency. At the same time, some researches also focus on internal parallelism, thus making some improvements to ZNS-SSD. This paper integrates the existing research on ZNS parallelism and the improvements in the corresponding fields and introduces them.

Key words Zoned namespace; Internal parallelism; Interregional interference

1 引言

分区命名空间 (ZNS) 是 NVMe 规范[1][2] 的一部分，它通过将逻辑区域与固态驱动器 (SSD) 中的物理介质，例如 NAND 闪存对齐来向主机系统公开存储块接口[3]。传统的 SSD 使用 FTL（闪存转换层）将数据有效地从块接口放置到物理介质，同时隐藏不在位的更新约束[3][4][5]。与传统的 SSD 不同，ZNS 允许主机系统通过向主机系统公开的逻辑区域放置数据。尽管区域中仅附加写入约束的限制，ZNS 通过减少内部 DRAM 使用和 SSD 内部的过度配置区域[6]来提高其在存储市场的竞争力，从而最大限度地提高可用存储容量[7]。

最近，几项研究报告称，由于 ZNS 的透明设计[8][9][10]，因此与传统存储接口相比，ZNS 有望提供更好的性能。例如，有研究[10][11]声称，由于主机比底层闪存固件更能利用应用程序知识，因此它可以更好地进行应用程序感知数据放置并执行接近最佳的垃圾收集（GC）。这可以使 GC 脱离关键路径，这样 ZNS 有望提高性能可预测性并减少读尾延迟；从另一方面即成本效益的角度来看：SSD 是一个复杂的系统，而不是像 DRAM 存储那样的被动设备[12]。SSD 需要页面到页面的地址转换，并

在其设备中维护所有映射信息。这会导致更高的货币成本和可扩展性限制，从而阻碍数据中心和企业服务器更积极地用 SSD 替换 HDD。需要注意的是，闪存固件和内部 DRAM 的成本是 SSD 的一半或更多[13]。

ZNS 作为一个相较于 OCSSD[14] 的更实用的接口，可以使存储更具成本效益。ZNS 定义了一组区域，每个区域都直接映射到一个或多个物理闪存块。与 OCSSD 相比，ZNS 允许底层 SSD 具有闪存固件，但它可以更轻量，因为固件只管理跨不同区域的映射和设备级错误，而不是页面。从主机端来看，ZNS SSD 的地址空间由一组区域公开，每个区域包含自己的逻辑地址，可以像传统 SSD 一样访问。然而，由于区域是闪存块的抽象，每个区域的地址空间应按顺序写入，并且在存储数据之前需要重置（即擦除）。因此，主机接管了现有闪存固件的所有繁重任务，如 GC、缓存和页到页地址转换。主机和 SSD 之间的边界可以降低为大尺寸的内部 DRAM 和固件支付的货币成本。

尽管 ZNS 的优势非常明显，但不合适的 Zone 大小仍然会通过影响 SSD 的内部并行性从而较大程度的影响 SSD 的性能。胡等人[15]对 SSD 内部并行性已经加以研究。SSD 内部有四个级别的并行性：通道并行性，chip 并行性，die 并行性和 plane 并行性。而如何去利用这些不同级别的并行性来提高 SSD 的性能又是主要由三个内在因素决定的：advanced commands，allocation schemes 和 4 个并行性的优先级。首先，提供不同 advanced command 的 flash 可以在 SSD 内部开发出不同级别的并行性，他们可以降低也可以提高 SSD 的性能，这取决于如何使用；其次，不同的物理页映射方式会使用不同的 advanced command，从而具有不同级别的内在并行性，会对 SSD 性能产生不同的影响；最后，在三个内在因素中，4 个级别的并行性的优先级顺序对 SSD 性能产生的影响最大。4 个级别的并行性最好的优先顺序是：通道并行性，die 级别并行性，plane 级别并行性，chip 级别并行性。个 IO 线程是无法充分利用所有这些并行特性的，只使用单个线程进行小 IO 访问，会导致整体访问延时更长。使用多个线程并发访问，则可以利用 SSD 内部的这些并发特性。SSD 上对本机命令队列（Native Command Queuing）的支持可以有效地在多个 channel 之间分配读写操作，从而提高内部 IO 并发性。因此，上层应用或存储系统尽可能并发访问小 IO，是非常有益于读写性能提升的。如果针对单个应用很难进行多线程并发，则可以考虑多个应用对数据进行并发访问，从而充分使用 SSD 的并发特性。然而在 ZNS 的背景下又会增加新的并行性约束，从而进一步影响整个存储的效率。因此本文重点介绍 ZNS 带来的并行性问题以及现有研究对其改进、优化和新的应用。

2 问题与挑战

2.1 Small-Zone内部并行性挑战

现有几项研究敦促 ZNS 支持小区域[3][9]，但其实际实施可能会失去大区域带来的平衡性，从而严重降低性能。Bae 等人[16]比较了测试的两个生产 ZNS SSD 的顺序和随机读取带宽。这两个 SSD，称为 ZNS-small 和 ZNS-large，使用相同的闪存固件控制器和相同的后端存储技术（小于 32TB，基于 TLC 的高密度闪存）。后端存储的每个闪存芯片都包含四个平面，完全并行运行。唯一的区别是 ZNS 配置。ZNS-large 为每个区域配置 2.18GB，支持 12 个开放区域（允许 12 个当前写入），而 ZNS-small 使用 96MB 和 4096 个开放区域。所有 ZNS 固态硬盘都通过 PCIe 3.0×4（最大 3.94GB/s）连接到具有 2.3GHz 英特尔至强 20 核 CPU 的主机。访问模式由 FIO [17] 和 ZoneFS [18] 生成。

由上述配置得出的结果，Bae 等人发现随着读取的新任务大小的增加，ZNS-large 的带宽会变得更好。当其将块大小增加 16MB 时，它会达到 PCIe 的最大带宽并饱和。它在更长的请求长度下表现出更好的性能的原因是 SSD 的内部并行性[19][20][21][22][23][24]。由于大型请求可以拆分为多个子请求，因此可以将其条带化到 ZNS-large 的不同内部资源（与一个或多个区域相关联）并并行服务（例如，闪存通道和芯片）。其他传统类型的 SSD 也观察到这种现象，因为在请求大小足以跨越所有内部资源的情况下，底层固件控制所有内部并行性。

2.2 ZNS区域间干扰挑战

在多应用程序进程环境下，Bae 等人研究了 ZNS-Large 和 ZNS-Small 的读取性能比较。据其结果来看，ZNS-small 的性能根据区域间干扰水平而显着变化，且依据不同的带宽将干扰级别（IL）进行分类为 0 至 7 级。据结果来看，具有 IL 7 的

ZNS-Small 平均表现出 809 MB / s，而 ZNS-Small 具有 32 个进程的带宽达到其 PCIe 接口在没有干扰的情况下提供的最大带宽（3.94 GB / s）（IL 0）。随着干扰水平的降低，ZNS-small 有望提高 SSD 的内部并行度，从而显着增加带宽。

作者因此还总结了基于不同干扰水平的 ZNS-Small 的性能改进。ZNS-Small 的 IL0 比 IL7 的 ZNS-Small 性能上优秀 2.84 倍。即使在一定程度上（不是完全）降低干扰水平的情况下，性能提升也是显着的。例如，如果主机可以通过将干扰级别从 7 控制到 4 来明智地将区域分配给不同的进程，则 ZNS-Small 的带宽将平均速度提高 1.33GB/s。但是，不幸的是，主机无法控制干扰级别，因为 ZNS 中没有接口抽象，从而将内部硬件配置暴露给主机。

3 研究现状

3.1 ZNS小区域改进

为充分利用内部并行性对小区域 ZNS 做了改进，即在 ZNS 中添加简单的功能来计算干扰级别或硬件布局，从而使主机可以在不损失并行性的情况下控制 ZNS-small，同时显着减少区域回收延迟。作者展示了一种简单有效的方法，通过利用内部并行性来提高 ZNS SSD 的带宽。此方法由区域干扰探测器和区域感知调度器组成。区域干扰探测器用来检测每个区域的性能下降，并将区域分类为多个冲突组（CG），每个冲突组都包含相互干扰的区域。然后，区域感知调度程序会以均匀分布的方式发出多个 CG 的 I/O 请求并进行最终调度。

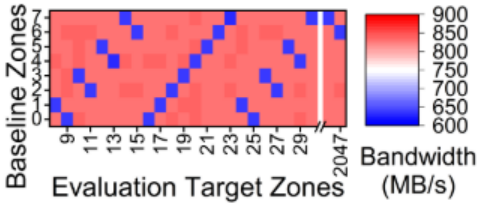

图 3-1 区域间干扰示例

图 3-1 展示了如何检测和分析标准 ZNS 区域之间的干扰。在此示例中，y 轴用以显示一组要比较的基线区域（0~7），而 x 轴则表示将要与相应基线区域并行评估的不同区域索引。为简洁起见，该图通过选择彼此完全没有干扰的区域来配置基线区域。从该图中可以看出，每个比较项目表现出相同的性能值：600MB/秒（蓝色）与 900MB/秒（红色）。蓝色项目表示基线和目标区域之间存在干扰。

出于动态考虑，需要适当地比较所有区域。图 3-2 即算法 1 解释了如何将区域分类为不同的 CG。一开始，该算法使用区域 0 初始化 CG0 的基线区域。它还需要设置性能阈值来确定访问是否区域是否干扰基线区域。阈值可以通过计算两个区域的平均平均值来实现，每个区域都表现出高性能和低性能。然后，它访问所有区域并评估每个区域是否与 CG 的基线区域相关联。在评估过程中，主机可以并行发出一些访问区域（即 $Zone_k$）和 CG 的基线区域（即 CG[l][0]）的请求。如果区域的带宽低于阈值，则会将此区域添加到目标 CG（CG[l]）中。否则，它将创建一个新的 CG 并使用访问区域设置基线区域。一旦访问了所有区域，该算法就会生成一个区域到 CG （Z2C） 映射表，该表可用于运行时 I/O 调度程序。

Algorithm 1: Interference detection.

Input: N := Number of zones
Output: CG := an array of conflic groups

```
CG[0][0] = zone_0 // Set a init baseline zone
for k ← 1 to N − 1 do // For all zones
    bw.append(bandwidth(zone_k, CG[0][0]))
threshold = average(bw.max, bw.min)
for k ← 1 to N − 1 do // For all zones
    newCG = 1
    for l ← 0 to len(CG)-1 do // For all CGs
        if bandwidth(zone_k, CG[l][0]) < threshold
            newCG = 0
            CG[l].append(zone_k)
            break
    if newCG
        CG.append([]) // Add new CG
        CG[len(CG)][0]=zone_k // Set new baseline zone
```

图 3-2 区域间干扰探测算法

干扰感知调度器的主要目标是调度来自不同 CG 的 I/O 请求，而不是尽可能多地调度同一个 CG。它使用 Z2C 映射表增加了 SSD 的内部并行度。由于块 I/O 请求（bios）没有关于区域的信息，块层（例如 blk_mq）检查传入 bio 的逻辑块地址（LBA）以确定哪个区域拥有它。基于检索到的区域索引，块层可以通过参考 Z2C 映射表找到对应的 CG，并将 CG 索引标记到目标 bio。在此 CG 标记之后，调度程序检查已为每个 CG 发出了多少请求，并为要调度的未完成请求数较少的 CG 提供最高优先级。这有助于通过在利用内部并行性的同时考虑负载平衡来跨不同 CG 公平地调度 I/O 请求。显然，块层需要保持每个 CG 的未完成请求数，这将在其干扰感知调度中参考。

3.2 GC改进

Choi 等人[25]做了一组实验，比较了单独读取区域中所有块（每个请求 4KB I/O 大小）和以组方式（每个请求 8KB ~ 128KB I/O 大小）在访问 ZNS 区域过程的总耗用时间。结果表明，以组方式访问比个人访问快得多。这是因为它不仅可以减少请求数量，还可以利用 ZNS SSD 的内部并行性。一般来说，ZNS SSD 中的一个区域分布到多个通道中，这使得有机会并行处理具有连续块的请求，如 OCSSD。这一观察促使其设计 LSM 风格的垃圾收集方案。

新颖的垃圾回收方案采用冷热数据收集的方法，在垃圾回收期间，它不仅读取有效块，还读取 128KB I/O 大小的无效块，而 Basic ZGC 只读取有效块。为硬盘设计的原始 LFS [26]会像新颖垃圾回收的方案一样读取所有数据。然而，大多数为 SSD 设计的文件系统和 FTL 仅读取有效数据，因为闪存中没有寻道开销[27]。这里作者仔细地论证了读取所有块是 ZNS SSD 中的一个可行选项，以完全获得上文中所观察到的内部并行性。实际上，作者在设计 LSM ZGC 时考虑了候选段的利用。具体来说，当一个段中的有效块数小于 16 时，LSM ZGC 只读取有效块。否则，它会读取所有块，因为 16 个 128KB 大小的请求可以覆盖整个 2MB 段数据。

3.3 加速RocksDB

分区命名空间可为主机系统提供与物理介质（例如 NAND 闪存）对齐的逻辑区域。由于物理介质的异地更新方案，基于日志结构合并树 (LSM-Tree) 的键值存储（例如 RocksDB）非常适合 ZNS SSD 的仅附加约束。尽管作为 RocksDB 文件系统插件的 ZenFS 在大区域 ZNS SSD 上显示出不错的吞吐量，但小区域 ZNS SSD 的性能令人失望，因为小区域没有空间利用内部并行性。因此，Kang 等人[28]提出了两种并行 I/O 机制，通过将 I/O 数据并行分布到多个区域来利用外部并行性。所提出的 I/O 机制根据 LSM 树级别的特点应用于 ZenFS。此外，还提出了一种动态区域管理策略，通过将具有相似生命周期的数据块放置在同一区域中来最大化空间利用率，从而使放置在一个区域中的文件大约在同一时间被删除。这使得区域可以在没有垃圾收集的情况下被重用。所提出的方法在 RocksDB 的 db_bench 基准测试工具中使用各种工作负载进行评估。实验结果表明，与传统的 ZenFS 相比，所提出的方法取得了显着的改进。它显示平均 I/O 性能为 7.5 倍，同时保持几乎相同程度的空间效率。

3.4 减少碎片整理开销

分区命名空间具有垃圾收集开销低、资源调配成本低等优点。然而，由于它的异地更新和应用程序的多线程编写行为，它很容易出现碎片。碎片化会分割 I/O 请求并导致设备内的资源冲突，从而降低 I/O 性能。大多数碎片整理工具需要将文件的全部内容从 SSD 读取到主机内存，然后将数据重写到 SSD 中的连续空间。主机级迁移操作会延长碎片整理的时间，数据迁移的过多写入操作甚至会缩短设备寿命。Qi 等人[29]通过实验发现，对碎片程度较低或数据较冷的数据进行碎片整理几乎不会提高性能。通过观察，提出了一个名为 InDeF 的新碎片整理工具，以减少碎片整理开销。InDeF 结合了逻辑和物理碎片的程度以及访问热度，以过滤掉对 I/O 性能影响不大的碎片，从而减少 SSD 的写入流量。为了利用内部闪存芯片并行性，InDeF 在 SSD 上卸载数据迁移，从而减少碎片整理所需的时间。评估结果表明，与传统的碎片整理工具相比，InDeF 减少了 91.6%～94.2%的碎片整理时间和 38.1%～66.7%的数据迁移量。

4 总结与展望

最近许多研究都倾向于解开 SSD 内部的结构以增强 I/O 的性能。ZNS SSDs 是其中的努力之一，其方法是通过 zone 的概念暴露自己的地址。可通过将不同的负载分发到不同的 zones 上，以降低写放大，提高性能和寿命。另外一个 ZNS SSDs 的特征是，闪存管理 (如 maping 和 GC) 在主机端执行。优点是减少了 SSDs 中 DRAM 的需求量和减少了 overprovisioning area。

然而，ZNS SSDs 引发了几个问题。一是怎么样管理 zone (如在 host-level 进行 reset 和 GC)，二是 ZNS SSDs 的 zone 内顺序写的约束。

为了创建算法合适的管理 ZNS SSDs，需要理解 ZNS SSDs 的特性。现有研究[30]已经得出这些功能是如何影响 ZNS SSD 的性能，并且基于一个真实的 ZNS SSD 原型设计了一个分析工具，可以评估不同工作负载下的性能。然后，其研究了并行性、隔离性和可预测性方面的不同特征。观察结果表

明：首先，在大型单元中请求 I/O 是 ZNS SSD 中获得并行性所不可或缺的；其次，与传统 SSD 相比，工作负载可以更有效地隔离；然后，意外的性能下降不会受到监控；最后基于 LBA（逻辑块地址）的性能不同，当我们为 ZNS SSD 设计新算法时，可以有效利用 LBA。这些都是今后研究可以着重考虑的方向。不止如此，并行性所能给 ZNS 带来的优势可以应用到 GC，碎片等多个方面，这也是可以进行突破的研究内容。

参 考 文 献

[1] Nvm express – scalable, efficient, and industry standard, 2022. https://nvmexpress.org/, Accessed on 2022-04-15.

[2] Zoned namespace command set specification - nvm express, 2021.https://nvmexpress.org/wp-content/uploads/NVM-Express-Zoned-Namespace-Command-Set-Specification-1.1-2021.06.02-Ratified-1.pdf,Accessed on 2022-04-15.

[3] Matias Bjørling, Abutalib Aghayev, Hans Holmberg, Aravind Ramesh,Damien Le Moal, Gregory R Ganger, and George Amvrosiadis. {ZNS}: Avoiding the block interface tax for flash-based {SSDs}. In 2021 USENIXAnnual Technical Conference (USENIXATC 21), pages 689–703, 2021.

[4] Aayush Gupta, Youngjae Kim, and Bhuvan Urgaonkar. Dftl: a flash translation layer employing demand-based selective caching of page-level address mappings. Acm Sigplan Notices, 44(3):229–240, 2009.

[5] Theano Stavrinos, Daniel S Berger, Ethan Katz-Bassett, and Wyatt Lloyd. Don’t be a blockhead: zoned namespaces make work on conventional ssds obsolete. In Proceedings ofthe Workshop on Hot Topics in Operating Systems, pages 144–151, 2021.

[6] Hojin Shin, Myounghoon Oh, Gunhee Choi, and Jongmoo Choi. Exploring performance characteristics of zns ssds: Observation and implication. In 2020 9th Non-Volatile Memory Systems and Applications Symposium (NVMSA), pages 1–5. IEEE, 2020.

[7] Samsung introduces its first zns ssd with maximized user capacity and enhancedlifespan,2021.https://news.samsung.com/global/samsung-introduces-its-first-zns-ssd-with-maximized-user-capacity-and-enhanced-lifespan, Accessed on 2022-04-15.

[8] Matias Bjørling, Abutalib Aghayev, Hans Holmberg, Aravind Ramesh, Damien Le Moal, Gregory R. Ganger, and George Amvrosiadis. ZNS: Avoiding the block interface tax for flash-based SSDs. In 2021 USENIX Annual Technical Conference (USENIX ATC 21), pages 689–703.USENIX Association, July 2021.

[9] Kyuhwa Han, Hyunho Gwak, Dongkun Shin, and Jooyoung Hwang. Zns+: Advanced zoned namespace interface for supporting in-storage zone compaction. In 15th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 21), pages 147–162, 2021.

[10] Theano Stavrinos, Daniel S Berger, Ethan Katz-Bassett, and Wyatt Lloyd. Don’t be a blockhead: zoned namespaces make work on conventional ssds obsolete. In Proceedings ofthe Workshop on Hot Topics in Operating Systems, pages 144–151, 2021.

[11] Western Digital. ZenFS, Zones and RocksDB - Who Likes to Take out the Garbage Anyway? Technical report.

[12] Lorenzo Zuolo, Cristian Zambelli, Rino Micheloni, and Piero Olivo. Ssdexplorer: A virtual platform for ssd simulations. In Solid-State-Drives (SSDs) Modeling, pages 41–65. Springer, 2017.

[13] Myoungsoo Jung, Wonil Choi, John Shalf, and Mahmut Taylan Kandemir. Triple-a: A non-ssd based autonomic all-flash array for high performance storage systems. ACM SIGARCH Computer Architecture News, 42(1):441–454, 2014.

[14] LightNVM. Open-Channel Solid State Drives Specification. Rev. 2.0.

[15] Hu Y, Jiang H, Feng D, et al. Exploring and Exploiting the Multilevel Parallelism Inside SSDs for Improved Performance and Endurance[J]. IEEE Transactions on Computers, 2013, 62(6):1141-1155.

[16] Bae H, Kim J, Kwon M, et al. What you can't forget: exploiting parallelism for zoned namespaces. In Proceedings of the 14th ACM Workshop on Hot Topics in Storage and File Systems. 2022: 79-85.

[17] Jens Axboe. fio.

[18] ZoneFS - Zone filesystem for Zoned block devices.

[19] Myoungsoo Jung, Ellis HWilson III, and Mahmut Kandemir. Physically addressed queueing (paq) improving parallelism in solid state disks. ACM SIGARCH Computer Architecture News, 40(3):404–415, 2012.

[20] Myoungsoo Jung and Mahmut T Kandemir. An evaluation of different page allocation strategies on high-speed ssds. In HotStorage, 2012.

[21] Myoungsoo Jung and Mahmut Kandemir. Revisiting widely held ssd expectations and rethinking system-level implications. ACM SIGMET-RICS Performance Evaluation Review, 41(1):203–216, 2013.

[22] Myoungsoo Jung and Mahmut T Kandemir. Sprinkler: Maximizing resource utilization in many-chip solid state disks. In 2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA), pages 524–535. IEEE, 2014.

[23] Myoungsoo Jung, Wonil Choi, Shekhar Srikantaiah, Joonhyuk Yoo, and Mahmut T Kandemir. Hios: A host interface i/o scheduler for solid state disks. ACMSIGARCH Computer Architecture News, 42(3):289–300, 2014.

[24] Congming Gao, Liang Shi, Mengying Zhao, Chun Jason Xue, Kaijie Wu, and Edwin H-M Sha. Exploiting parallelism in i/o scheduling for access conflict minimization in flash-based solid state drives. In 2014 30th Symposium on Mass Storage Systems and Technologies (MSST), pages 1–11. IEEE, 2014.

[25] Choi G, Lee K, Oh M, et al. A New {LSM-style} Garbage Collection Scheme for {ZNS}{SSDs}[C]//12th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 20). 2020.

[26] ROSENBLUM, M., AND OUSTERHOUT, J. K. The Design and Implementation of a Log-structured File System. Transactions on Computer Systems (1992).

[27] LEE, C., SIM, D., HWANG, J.-Y., AND CHO, S. F2FS: A new file system for flash storage. In FAST (2015).

[28] Im M, Kang K, Yeom H. Accelerating RocksDB for small-zone ZNS SSDs by parallel I/O mechanism[C]//Proceedings of the 23rd International Middleware Conference Industrial Track. 2022: 15-21.

[29] Qi W, Tan Z, Shao J, et al. InDeF: An Advanced Defragmenter Supporting Migration Offloading on ZNS SSD[C]//2022 IEEE 40th International Conference on Computer Design (ICCD). IEEE, 2022: 307-314.

[30] Shin H, Oh M, Choi G, et al. Exploring performance characteristics of ZNS SSDs: Observation and implication[C]//2020 9th Non-Volatile Memory Systems and Applications Symposium (NVMSA). IEEE, 2020: 1-5.