ZNS：优势、技术和未来发展

洪子骁

1) 华中科技大学计算机科学与技术学院

摘 要 当前对闪存设备的研究几乎完全集中在传统的 SSD 上。然而，业界已经标准化并正在采用分区命名空间（ZNS）SSD[1]，它提供了一种新的存储接口，可以取代传统 SSD。因此，对传统 SSD 的持续研究错失了通过构建 ZNS SSD 来解锁系统性能的阶跃变化的机会。我对 ZNS 的优势和未来发展进行了总结，并介绍 LFS[2]和 CAZA[3]新技术。

关键词 闪存设备；分区命名空间；LFS；CAZA；

ZNS: Advantages, Technology and Development

zixiao hong

1)(Department of Computer Science, Huazhong University of Science and Technology, 430000, China)

Abstract

Research on flash devices almost exclusively focuses on conventional SSDs, which expose a block interface. Industry, however, has standardized and is adopting Zoned Namespaces (ZNS) SSDs[1], which offer a new storage interface that dominates conventional SSDs. Continued research on conventional SSDs is thus a missed opportunity to unlock a step-change improvement in system performance by building on ZNS SSDs. I summarized the advantages and future development of ZNS, and introduced the new technologies of LFS [2] and CAZA [3].

Keywords flash devices; ZNS; LFS; CAZA

1 介绍

云和数据中心服务需要对海量数据集进行低延迟访问。基于闪存的固态硬盘（SSD）提供比硬盘驱动器（HDD）低几个数量级的延迟和更高的 I/O 吞吐量，同时以比 DRAM 更低的成本提供更多容量，从而成为满足这些需求的关键。因此，SSD 用于延迟敏感型应用程序，例如缓存和键值存储现在非常普遍，即使是对延迟不敏感的应用程序（如通用文件系统）也构建在 SSD 之上，云提供商不提供 HDD 作为通用 VM 的一部分。

不幸的是，擦除块中的闪存单元只能在完全擦除后重写。闪存单元会随着每个写入和擦除周期而磨损，最终失去可靠存储数据的能力，从而限制了电池的耐用性。

在传统的 SSD 中，闪存单元及其特性隐藏在传统的块接口后面。该接口通过 SSD 上的复杂固件（即闪存转换层（FTL）实现。块接口向主机公开一个平面地址空间，该空间可以以页面粒度（通常为 4KB）进行运行写入，类似于 HDD。此接口是应用程序开发人员熟悉的，并且受主要操作系统的支持。但是，由于闪存单元无法覆盖，并且必须以擦除块粒度（通常为几兆字节）擦除，随机写入会强制 FTL 实现垃圾回收，以从日志地址空间中被覆盖的旧数据中回收空间。垃圾回收会在擦除擦除块之前将有效数据转发到过度配置的（备用）闪存空间中。这会导致写入放大，其中写入逻辑地址空间一次的字节在闪存上物理写入多次。写入放大通过使用过多的写入和擦除周期来缩短器件寿命。将大约在同一时间失效的数据放在一起是避免写入放大的关键。但是，FTL 无法访问此类数据放置所需的应用程序级信息，并且应用程序对 FTL 如何在设备上排列数据的控制有限。

大量的研究工作已经用于管理传统 SSD 块接口的不良影响。这包括在管理垃圾回收和其他 FTL 任务引起的性能降低和不可预测性方面所做的大量工作。之前的工作已经对 FTL 进行了逆向工程，以找到最适合 FTL 内部操作的访问模式。系统还经常限制闪存写入以延长闪存设备的使用寿命，因为它们的工作负载会导致高写入放大。

本文认为，系统社区今天应该停止研究传统的 SSD。研究的努力应该转移到分区命名空间（ZNS）SSD。ZNS 是一种新的 SSD 接口，在传统的块接口中占主导地位

ZNS 的优势在于接口与数据写入物理闪存的方式相匹配，同时抽象了闪存硬件的混乱细节。ZNS SSD 分为称为区域的逻辑区域。写入可以转到任何区域，但必须在区域内按顺序进行。ZNS 已成为现实：接口规范于 2020 年添加到 NVMe 标准中，ZNS SSD 可从多家供应商处获得，大型云提供商正在采用它们。在软件方面，越来越多的 ZNS 兼容文件系统、键值存储和开发框架正在消除采用障碍。

由于 ZNS 接口更匹配数据的写入方式，因此 FTL 更薄：与传统的 FTL 相比，它执行更粗粒度的地址转换（即，在擦除块粒度而不是更小的页面粒度），并且不执行垃圾收集。因此，消除了性能差异和垃圾合作的其他影响;性能建模和逆向工程 FTL 是不必要的。主机控制写入放大，并可以做出应用程序感知的数据放置和 I/O 调度决策。设备更便宜，因为不需要为垃圾回收过度配置闪存容量，并且地址转换只需要一小部分 DRAM。某些 ZNS 用例可能需要主机资源来执行传统上由 FTL 执行的任务。至关重要的是，使用 ZNS，应用程序开发人员可以选择将多少系统资源（如果有的话）用于与闪存相关的任务。在传统的 SSD 中，DRAM 和过度配置的闪存电容是固定成本，无论是否需要它们。

研究对最近关于 SSD 的系统文献进行了调查，发现只有 18%的论文不会受到向 ZNS SSD 转变的影响;23%的论文涉及 ZNS 简化或解决的问题。其余 59%的论文将受到影响或需要重新审视，一旦行业不可避免地转向使用 ZNS SSD 以获得成本和性能优势。

未来的研究应该集中在使用 ZNS SSD 使研究的系统更便宜、更快、更高效，而不是过时的传统 SSD 问题。研究提出了几个广泛的研究方向，这些方向将提高系统性能并针对 ZNS SSD 引入的特定限制和挑战。

分区命名空间 SSD 几乎在各个方面都领先着传统设备：它们更便宜，向主机公开更有用的接口，并且可能获得更好的性能。

2 背景

2.1 Flash 架构简介

闪存架构。闪存由 NAND 单元组成。一个单元格可以存储一个（SLC）、两个（MLC）、三个（TLC）、四个（QLC），或五个（PLC）位，取决于编程和保留电压电平的数量。

为了实现高容量，闪存单元以三维方式排列：二维排列在阵列中，然后堆叠。为了实现高吞吐量，读写操作利用数千个单元的并行性。

此体系结构形成一个深层的多层层次结构。多个单元形成一个页面，其中包括用于存储纠错代码的大量冗余。多个页面形成一个 block 擦除块，而 blocks 又形成 planes 平面。多个 planes 平面最终形成一个 channel 通道（或死亡）。

读写 Flash。闪存公开读/写/擦除接口。读取可以按页面粒度（通常为 4KB）进行。在写入（编程）页面之前，必须将其擦除。擦除所需的时间是编程的几倍。为了隐藏此延迟，擦除按块粒度（数十 MB）进行批处理。在擦除块中，页面按顺序编程，直到没有空页。

多个读/写操作通常安排在每个通道中的多个平面上并行发生。对多级单元进行编程涉及的步骤要多得多，制造商经常实施这种几何形状的变体以进一步提高并行性，从而将擦除块大小增加到数百兆字节。

在传统固态硬盘中，闪存转换层（FTL）隐藏了设备几何形状的复杂性和擦除块的存在。如果相邻的逻辑页在不同的时间写入，它们可能会映射到不同的块偏移量，甚至映射到不同的块和平面。FTL 负责：

将每个逻辑地址转换为平面、擦除块和页面的层次结构。

垃圾回收：当擦除包含有效和无效页面混合的擦除块（即在 Flash 上逻辑上被覆盖的页面）时，FTL 将有效页面转发到剩余可用页面的擦除块中，然后擦除旧块。

持久且一致地存储 FTL 数据结构，为断电事件做好准备。

磨损均衡：通过平衡所有擦除块之间的擦除，确保擦除块尽可能均匀地磨损。

分区命名空间 SSD：ZNS SSD 将地址空间划分为多个区域，其行为类似于擦除块：区域只能按顺序写入。已满时，必须先重置（擦除）它，然后才能接受新数据。当前写入位置使用写入指针进行跟踪。

区域可以处于六种不同的状态：空、打开、关闭、满、只读和脱机。区域从空开始，打开并写入直到满，然后重置为空状态，之后可以再次打开。只有有限数量的区域可以同时处于活动状态，因为每个活动区域都会消耗资源（例如，来自写入缓冲区）。但是，所有区域可以同时处于只读模式。通过在重置后减小区域的长度或将区域标记为脱机，可以透明地处理闪存单元故障。

区域至少与擦除块一样大。例如，在最近的工作中评估的设备使用 1GB 区域并支持 14 个活动区域。

2.2 降低块接口税

现代存储设备（如 SSD 和 SMRHDD）依赖于与块接口不匹配的记录技术。这种不匹配会导致性能和运营成本。在基于闪存的 SSD 上，可以在写入时对空闪存页进行编程，但覆盖它需要擦除操作，该操作只能在擦除块（一组一个或多个闪存块，每个块包含多个页面）的粒度下发生。要使 SSD 公开块接口，FTL 必须管理功能，例如使用随处写入方法进行就地更新、将主机逻辑块地址（LBA）映射到物理设备页面、垃圾收集过时数据以及确保擦除块均匀磨损。

FTL 会对性能和运营成本产生重大影响。为了避免就地更新的媒体限制，每个 LBA 写入都将定向到下一个可用位置。因此，主机将物理数据放置的控制交给 FTL 实现。此外，必须对较旧的陈旧版本的数据进行垃圾回收，从而导致正在进行的操作的性能不可预测性。

垃圾回收的需要在设备上分配物理资源。这要求介质超出总容量的 28%，以便暂存在物理地址之间移动的数据。还需要额外的 DRAM 来维护逻辑地址和物理地址之间的易失性映射。容量预留空间和 DRAM 是 SSD 中最昂贵的组件，导致每 GB 可用容量的成本更高。

2.2.1 现行减税策略

降低块接口税的两种主要方法已经获得了证实：具有流支持的 SSD（Stream 固态硬盘）和开放通道固态硬盘（OCSSD）。

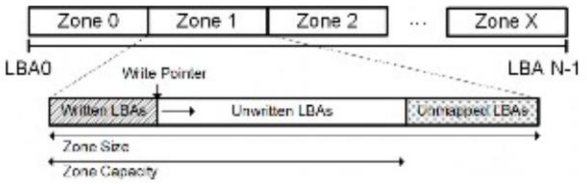

图 1：存储部门的 LBA 地址空间内的区域。每个区域的写入指针在成功写入时增加，并通过发出显式重置命令进行重置。

流 SSD 允许主机使用流提示标记其写入命令。流提示由流 SSD 解释，允许它将传入的数据区分到不同的擦除块上，从而提高整体 SSD 性能和介质寿命。流 SSD 要求主机仔细标记具有相似生存期的数据，以减少垃圾回收。如果主机将不同生命周期的数据混合到同一流中，则流 SSD 的行为类似于块接口 SSD。流 SSD 必须携带资源来管理此类事件，因此流 SSD 不会降低块接口 SSD 的成本，用于额外的媒体预留空间和 DRAM。研究比较了 StreamsSSD 和 ZNS SSD 的性能。

开放通道 SSD 允许主机和 SSD 通过一组连续的 LBA 块进行协作。OC-SSD 可以公开这些块，以便它们与媒体的物理擦除块边界对齐。这消除了设备内垃圾收集开销，并降低了媒体过度配置和 DRAM 的成本。使用 OCSSD 时，主机负责数据放置。这包括基础介质可靠性管理，如磨损均衡和特殊介质故障特征（取决于 OCSSD 类型）。与 StreamSSD 相比，这有可能提高 SSD 性能和介质寿命，但主机必须管理 SSD 实施之间的差异以保证耐用性，从而使接口难以采用并需要持续的软件维护。

研究接下来介绍的 ZNS 接口建立在 OCSSD 和 SMRHDD 的经验教训之上。它利用并兼容 ZAC/ZBC[28，29]规范中定义的分区存储模型。它增加了一些功能，以利用基于闪存的 SSD 的特性。ZNS 旨在消除 SSD 介质和设备接口之间的不匹配。它还通过避免直接管理 OCSSD 等特定于介质的特征，提供了与介质无关的下一代存储接口。

2.2.2 带区域的免税存储

分区存储模型的基本构建基块是区域。每个区域代表 SSD 逻辑地址空间的一个区域，可以任意读取，但必须写入

必须按顺序重置 10 次，并且要启用新的写入。写入约束由每个区域的状态机和写入指针强制执行。每个区域的状态机使用以下状态确定给定区域是否可写：EMPTY、OPEN、CLOSED 或 FULL。区域从 EMPTY 状态开始，在写入时过渡到 OPEN 状态，最后在完全写入时过渡到 FUL。设备可以进一步对可以同时处于 OPEN 状态的区域数量施加开放区域限制，例如，由于设备资源或介质限制。如果达到限制并且主机尝试写入新区域，则必须将另一个区域从 OPEN 转换为 CLOSED 状态，从而释放设备上的资源（如写入缓冲区）。C 丢失区域仍然是可写的，但在提供其他写入之前必须再次转换为 OPEN 状态。

区域的写入指针指定可写区域中的下一个可写 LBA，并且仅在 EMPTY 和 OPEN 状态下有效。每次成功写入区域时，都会更新其值。主机发出的任何写入命令如果（1）不从写入指针开始，或（2）写入处于 FULL 状态的区域，将无法执行。重置区域时，通过重置区域命令，区域将转换为 EMPTY 状态，其写入指针更新为区域的第一个 LBA，并且不再访问之前写入的用户数据。区域的状态和写入指针使主机软件无需跟踪写入区域的最后一个 LBA，从而简化了恢复，例如，在不正确的关闭之后。

虽然不同分区存储规范的写入约束基本相同，但 ZNS 接口引入了两个概念来应对基于闪存的 SSD 的特性。

可写区域容量属性允许区域将其 LBA 划分为可写和不可写，并允许区域的可写容量小于区域大小。这使得 ZNS SSD 的区域大小能够与 SMRHDD 引入的双次方区域大小行业标准保持一致。图 1 显示了如何在 ZNS SSD 的逻辑地址空间上布局区域。

活动区域限制对可能处于 OPEN 或 C 丢失状态的区域添加了硬限制。虽然 SMRHDD 允许所有区域保持可写状态（即 CLOSED），但基于闪存的介质的特性（例如程序故障）要求 ZNS SSD 具有此数量。
尽管 ZNS 接口增加了主机软件的责任，但研究的研究表明，各种用例可以从一组简化其采用的技术中受益。

3 优势

3.1 ZNS 每 GB 成本更低

ZNS SSD 在每个设备上需要更少的专用资源，这大大降低了它们的成本。

更少的板载内存。在传统 SSD 中，FTL 将地址映射表（逻辑到物理）和垃圾收集元数据保存在设

备的板载 DRAM 中。在 ZNS SSD 中，FTL 以区域的粒度维护地址转换，这需要最少的 DRAM。

为了估计 DRAM 开销，研究假设页面为 4KB。传统 SSD 中的优化映射表每页需要大约 4 个字节[。在当前设备上，每 TB 闪存大约需要 1GB 的板载 DRAM。在 ZNS SSD 中，FTL 将区域映射到擦除块。假设每个块和 16MB 擦除块具有类似的 4 字节开销，则只需要~256KB 的板载 DRAM。在 ZNS 上构建一个更复杂的接口需要一些主机资源。

减少过度配置。传统的 SSD 保留了很大一部分闪存单元作为备用容量。这种过度配置通常是可用容量的 7-28%。FTL 使用此备用容量来减少垃圾收集开销。SSD 无需在覆盖逻辑地址时立即回收空间，只需将新数据写入干净的闪存页面并更新广告映射即可。延迟垃圾回收还会增加擦除块中的数据在需要回收块空间之前失效的可能性。在研究的随机写入工作负载和可变预留空间因子的实验室实验中，垃圾回收的写入放大（写入的额外字节数）从没有过度配置的 15 个增加到大约 25 个。具有~25%的过度配置。过度配置会抬高 SSD 价格，因为闪存单元是设备中最昂贵的部分。ZNS SSD 不进行垃圾回收，因为区域中的擦除块在重置区域时完全失效并被擦除。因此，主机可以访问几乎所有闪存容量（保留一些容量以替换坏闪存块）。尽管某些应用程序将执行主机端垃圾回收，但将此责任转移到主机意味着系统资源可以根据应用程序的需求进行分配，而不是为所有应用程序固定。

3.2 ZNS 是一个更有用接口

传统 SSD 公开的块接口对于任何开发读取或写入硬盘驱动器的软件的人来说都很熟悉。但是，块接口抽象了太多闪存的写入约束。这使得应用程序极难调整其 I/O 模式以最大限度地提高性能和设备耐用性。ZNS SSD 公开的界面达到了一个最佳点，在易于使用和忠实地表示设备如何处理数据之间取得了平衡。编写器可以写入任何区域，但每个区域必须按顺序写入。这种精简接口使应用程序能够更好地控制闪存上的数据放置，从而更好地控制写入放大。

分区界面适合在顶部构建其他空间。例如，使用 ZNS SSD 在主机上实现块接口非常简单。此任务由 NVMe 标准中的简单复制命令辅助，该命令允许主机发出设备控制器管理的数据复制操作。使用此命令，在擦除区域之前复制前转有效数据不会使用任何 PCIe 带宽，从而实现与传统 SSD 相当的性能。在设备的接口上构建抽象会将一些计算和 DRAM 负担转移到主机上，这是 ZNS SSD 的少数缺点之一。但是，为主机购买大型 DRAM DIMM 比每个 SSD 中的多个小型嵌入式 DRAM 芯片便宜。此外，由于主机可以洞察应用程序细节，因此它可以比板载 FTL 更好地管理垃圾收集和其他数据管理任务。主机可以选择是否支付额外的主机资源成本来构建不同的接口。

ZNS 是对两个现有抽象的改进。开放通道 SSD 向主机公开设备几何形状，从而实现对数据放置的细粒度控制。分区名称-paces 根据从开放通道 SSD 中吸取的经验教训标准化了稍高级别接口。多流写入 NVMe 指令在概念上类似于 ZNS。主机使用相同的流 ID 标记相关写入，设备将每个流写入其自己的一组确定擦除块。多流是主机对传统 SSD 中数据放置的有限控制的解决方法;传统设备的高硬件成本仍然存在。

3.3 ZNS 可能会获得更好的性能

FTL 在传统 SSD 中使用的关键性能策略（缓冲写入和战略性地放置写入以利用并行性）同样适用于 ZNS 设备。虽然需要进一步的研究来验证，但 ZNS SSD 应该能够完成这些任务，并且比板载 FTL 更好。但很明显，有关应用程序的信息是接近最佳垃圾收集的关键瓶颈，而传统的 FTL 没有这些信息。另一方面，主机有权进行应用程序感知的数据放置决策。由于主机可以通过将哪些数据写入同一区域来控制将哪些数据一起擦除，因此它可以对写入放大进行精细控制。主机可以根据数据的预期到期时间将数据分组到区域中，从而显著减少写入放大。

主机还可以利用应用程序知识做出更好的调度决策。在传统的 SSD 中，垃圾收集通过干扰 I/O 来增加尾部延迟。垃圾收集也使性能不可预测，因为 FTL 使用不透明的内部算法来调度它。主机显式回收 ZNS SSD 上的空间，通过降低主机在 I/O 周围安排垃圾收集来提高性能可预测性并减少读尾延迟。早期的 ZNS SSD 实验突出了它们的性能。西部数据报告称平均读取延迟降低 60%，并且基准测试中的更高吞吐量。西部数据还报告了 2-4 倍的读尾延迟降低和 2 倍更高的写入吞吐量 RocksDB over ZNS。CMU 研究人员表明，RocksDB 的写入放大率从 5 倍

下降到 1.2 倍在 ZNS 固态硬盘上。IBM 报告了 22 倍更低的尾部延迟和 65%的应用程序吞吐量增加。

3.4 ZNS 固态硬盘采用

ZNS 设备已经从一些供应商那里上市。所有主要的 SSD 制造商都有有效的 ZNS 原型，大多数都在超大规模测试中进行了数月的测试。

一位超大规模企业与研究分享说，ZNS SSD 是部署 QLC 闪存并在未来实现显著成本节约的关键构建块。在准备此部署时，正在进行大量投资以使后端软件与 ZNS 兼容。第二个超大规模企业公开分享了将其存储软件堆栈移植到 ZNS SSD 的性能结果。

在开源世界中，Linux 支持两种本机分区文件系统（ZoneFS 和 F2FS）。此外，主流文件系统 Btrfs 和 ext4 接近在 ZNS SSD 上原生运行。在此之前（以及不久的将来的 XFS），从 4.14 版本开始，Linux 提供了 dm 分区设备映射器，该映射器在任何 ZNS SSD 之上模拟标准块设备。此外，像 Ceph 和 RocksDB 这样的大型应用程已被移植到本机使用 ZNS SSD，同时实现了显著的性能提升。Storage 性能开发工具包（SPDK）是一个用于开发内核旁路存储应用程序的框架，从版本 20.10 开始支持 ZNS。

4 重新评估 SSD 研发的重点

为了了解这种技术转变对未来系统研究的影响，研究调查了最近在操作系统和存储会议上发表的闪存和 SSD 工作。具体来说，研究感兴趣的是，在 ZNS 设备占主导地位的未来，研究是否还会进行这项研究。需要明确的是，研究不质疑论文的价值或影响。这些工作解决了传统 SSD 的实际问题，到目前为止，还没有其他价格相似的技术可以避免这些问题。然而，ZNS 改变了这种情况，研究必须相应地重新评估研究的研究议程，否则研究将不必要地推迟使用这项新技术的最大潜力。因此，研究的调查旨在量化如果系统社区继续研究传统 SSD 可能会被误导的努力。

研究调查了过去五年在 FAST，OSDI，SOSP 和 MSST 发表的论文。研究总共收集了 465 篇论文，并将其缩小到 104 篇论文，其中基于闪存的 SSD 是研究或系统实施的重要组成部分；研究手动对这些进行了分类。这不是一项详尽无遗的调查；研究的目标是对近期系统研究的目标有一个广泛的了解。研究省略了关于超低延迟 SSD（例如三星 Z-NAND）和非易失性存储器（例如 3DXPoint）的论文，因为它们与传统闪存 SSD 具有不同的系统含义。

研究为论文定义了四大类（表 1 中的列标题以粗体表示）：

简化/解决：本文的主要问题通过 ZNS SSD 解决或简化。例如，为闪存模拟器构建 FTL 或改进垃圾收集。

应用：论文解决问题的方法可能会随着 ZNS 而改变，例如，系统实现可能会改变。

研究：研究或评估的结果可能会改变。例如，系统性能可能会发生变化[18]，或者测量研究的结果可能会有所不同。

Venue	# Pubs.	Simpl	Appr	Res	Orth
FAST	126	9	8	23	8
OSDI	164	3	0	4	0
SOSP	77	2	2	2	0
MSST	98	10	7	16	10
Total	465	24	17	45	18

Table 1: Impact of ZNS adoption on existing work on flash-based SSDs. Columns are counts. *# Pubs.* indicates the total publications in the venue over the last 5 years.

无关：论文中解决的问题与 ZNS 是无关的，例如，它提出了一种闪存的低级安全技术。

分类如表 1 所示。研究发现，23%的 SSD 论文关注的是 ZNS 简化或解决的问题。这一类别的论文侧重于减轻垃圾收集的负面性能影响或管理写入放大以保持设备耐久性。其他论文对 FTL 进行逆向工程或重新设计，以减少性能可变性；这在 ZNS SSD 中要简单得多，因为 FTL 更简单。另外 59%的 SSD 论文需要改变他们的方法或重新审视结果，因为它们会随着 ZNS SSD 的变化而改变。许多具有大量闪存组件的系统需要针对 ZNS 进行重新设计。严重依赖闪存的系统在移植到 ZNS 后可能会显示出不同的耐用性或性能特征，需要重新评估。在现场或极端工作负载下描述设备行为的性能和可靠性测量研究需要重新运行，以正确地描述包含 ZNS SSD 的系统。

5 技术介绍

5.1 事件日志和共享日志

5.1.1 简介

事件日志：事件日志跟踪应用程序和系统操作。可以分析日志以了解系统性能、趋势和异常。日志也可以被视为消息总线，将事件（或事件摘要）发布给订阅者。但是，如果发生故障，则可能会丢失故障之

前的事件，这些事件可能有助于解释故障的根本原因。如果使用组追加将事件向下推送到控制器，则持久日志记录速度更快，并且最近的日志记录丢失的可能性较小。

共享日志：共享日志是共享事件日志;Scalog 将它们描述为“可以由多个客户端访问和附加的有序记录序列”。有许多高度可扩展的分布式共享日志强制执行排序保证，包括 Kafka，CORFU，Tango，ZLog 和 Scalog。

5.1.2 向 ZNS 固态硬盘的演进

如果发生故障，则紧接在该故障之前发生的事件可能不会显示在事件日志中。这些缺失的事件可能有助于解释失败的根本原因。但是，如果使用组追加和区域追加将事件快速推送到控制器，则最近的日志记录丢失的可能性较小。

让研究来看看一个共享日志系统 Scalog 如何与 ZNS SSD 一起工作。Scalog 定义了直接映射到 ZoneAppend 的 append 和 appendToShard 操作。日志段的 FIFO 复制对应于发送块集合，可能是控制器到控制器，并由接收控制器快速确认。匹配块边界，之前传输的最后一个块之后的块可以使用区域追加追加。

5.1.3 研究方向

具有弹性可扩展性的可扩展共享日志对于现代数据管理至关重要。以下是 ZNS 面临的一些有趣的事件和共享日志研究挑战：

多个编写器：对于事件日志，是让一个编写器处理所有事件，还是让多个编写器可以追加到日志的尾部更好？

计算存储：事件是否应该在生成后立即发送到计算控制器，甚至在确定事件顺序之前？是否应尽可能使用跨计算存储的合作通信来管理共享日志的复制和排序，从而最大限度地减少主机参与？

区域歧视：在最初存储事件或重组共享日志时，存储事件的区域是否可以/是否应该存在歧视（基于数据访问模式）？这是否支持对包含不太重要数据的区域进行有效的垃圾收集，同时仍支持订单要求？

5.2 CAZA

5.2.1 简介

研究提出了一种新的 Compaction-Aware ZoneA 分配算法（CAZA），该算法允许新创建的 SSTables 在未来合并后一起删除。CAZA 在 RocksDB 的 ZenFS 中实现，研究广泛的评估表明，与 LIZA 相比，CAZA 显着降低了 WA 开销。

5.2.2 研究背景

ZNS SSD 引入了强制顺序写入并不允许覆盖的区域概念。由于写入模式的限制，ZNS SSD 不会在闪存转换层（FTL）中执行垃圾整理（GC）。删除设备内 GC 通过消除写入放大（WA）来提高 I/O 性能。但是，为了释放设备内部的空间，使用 ZNS SSD 的应用程序必须执行区域清理，这是一项擦除整个区域的操作。如果有效数据保留在被选为区域清理受害者的区域中，则由于复制有效数据，WA 问题仍然存在。

尽管如此，应用程序可以做出应用程序感知的数据放置决策，通过最小化区域清理期间复制的数据量来减少 WA。为了减少此数量，具有相同生存期的数据应写入同一擦除单元（ZNS SSD 的区域）。在 ZNS SSD 上运行的应用程序可以直接确定将在其中放置数据的区域，从而减少即将执行删除时区域中存在的有效数据量。这大大减少了区域清理期间的数据复制，最大限度地减少了 WA 问题。

5.2.3 应用

RocksDBforZNS SSD 使用 ZenFS（支持 ZNS 的 RocksDB 的用户级文件系统)进行高效的数据放置，以最大限度地减少区域清理开销。启用 ZNS 的 RocksDB 是一个基于日志结构的基于树的键值存储，在 ZNS SSD 上运行。具体而言，ZenFS 采用基于生存期的区域分配算法（LIZA），该算法将具有相同生存期的数据放在同一区域中。LIZA 使用 LSM 树的以下特征预测每个数据的生命周期。在 LSM 树中，即将更新和删除的 SSTables 位于较高级别，而相对较晚更新的 SSTables 位于较低级别。LIZA 根据热度（写入频率）为 LSM 树的每个级别提供生存期，根据 SSTable 属于同一区域，并将具有相同生存期的 SSTable 放在同一区域中，从而允许在将来触发压缩时将它们一起删除。但是，由于生存期预测失败，LIZA 无法准确识别要一起删除的 SSTable，从而无法最大限度地减少 WA 问题。

LIZA 的一个关键限制是 LIZA 不考虑 SSTable 如何在 LSM 树中失效。因此，本文提出了一种新的压缩感知区域分配算法（CAZA），其设计基于以下观点：一组 SSTables 一起被删除/失效是由 LSM 树的压缩算法确定的。压缩作业选择一组在相邻级别具有重叠关键范围的 SSTable，将它们合并到一个或

多个新的 SSTable，并删除用作合并压缩输入的 SSTable。考虑到压实过程，CAZA 将新创建的 SSTable 放置在与其关键范围重叠的 SSTable 最多的区域中。然后，当将来触发压缩时，被选为合并受害者的 SSTable 将与已在同一区域中的 SSTable 合并并一起删除/失效。这样可以将区域中剩余的有效数据量降至最低，从而最大程度地减少在擦除区域之前复制有效数据的开销。

5.2.4 评估

研究提出了一种新的区域放置算法，该算法可以识别 ZenFS 中 LSM 树的压缩过程。CAZA 会将具有重叠键范围的 SSTables 放在同一区域中，以便选择用于压缩 LSM 树的 SSTables 位于同一区域内。
具体而言，CAZA 的设计考虑了 LSM 树的以下特征。首先，SSTables 仅在压缩期间被删除和失效，并相应地创建新的 SSTable。其次，在压缩过程中，选择为受害者的 SSTable 和与此 SSTable 的键范围重叠的 SSTable 将被删除并一起失效。

为了进行评估，研究通过修改 RocksDB 版本 6.14 来实现 CAZA。评估证实，与 LIZA 相比，对于写入密集型合成工作负载，CAZA 将区域清理期间产生的有效数据副本减少了多达 2 倍，WA 减少了。

6 挑战

既然 ZNS 已经标准化，设备也可供商业使用，现在是时候重新审视研究提出的关于传统 SSD 的许多问题了，这次是 ZNS，并提出关于如何最好地将这项技术集成到研究的系统中的新问题。

本节首先介绍通过与主机上的应用程序共置垃圾协作而产生的几种证明性能的机会。然后，它描述了 ZNS 新界面引入的几个挑战。

6.1 提高性能

应用程序级信息如何改进区域管理？在向纠删块和区域分配写入操作时，有关应用程序及其访问模式的信息非常宝贵。如果写入擦除块的大部分数据同时过期，则垃圾回收开销最小。软件通常可以对数据过期做出有根据的猜测，例如，同一文件中的页面比不同文件中的页面更容易被删除或覆盖在一起。在相似时间创建的文件也更有可能一起出现（例如，分析工作负载中的中间文件）。在更高的层次上，与具有不同创建者的文件集相比，由同一应用程序、容器或虚拟机创建的文件集和所有者更有可能同时过期（例如，当应用程序退出或虚拟机迁移或关闭时）。

这些关键信息从未提供给传统的固态硬盘。从历史上看，即使使用接近最优的垃圾收集算法，这种信息屏障也将 FTL 限制在次优性能。文件系统具有现成的此信息，可以将其与 ZNS SSD 一起使用;但是，当前用于 ZNS SSD 的 Linux 内核文件系统（例如 F2FS）尚未使用此信息。因此，这是一个丰富的探索主题。文件系统知识（所有者、创建者、时间戳）可以减少多少写入放大？除了文件系统之外，特定于应用程序的信息还能进一步减少多少开销？给定其他信息，理论上最优的垃圾收集算法如何变化？

应用程序应如何与区域交互？这些接口中的每一个都对性能和可用性进行了权衡：原始分区存储访问提供了对 I/O 和数据放置的最大控制;文件系统和键值存储提供的控制较少，但易于使用（例如，许多文件系统处理元数据管理和数据完整性）。即使在一类接口中也有多样性。例如，F2FS 是一个功能齐全、符合 POSIX 标准的文件系统，而 ZoneFS 将区域视为与区域本身具有相同范围的文件。原始分区存储可能会通过内核或不通过。最后，各个应用程序具有可以根据性能进行调整的配置（例如，启用 ZNS 的 RocksDB 中的活动区域计数）。

闪存的耐用性有限，这意味着使用错误的接口不仅仅是性能，正如研究在传统 SSD 和块接口中看到的那样。分区存储设备（例如 SMR HDD）在 ZNS 之前就存在，但它们没有与闪存设备相同的耐用性限制。因此，尚未充分探索以最大化 I/O 性能、可用性和设备耐用性的方式与分区闪存设备进行交互。通常，应用程序是否更喜欢使用分区接口、文件系统或其他一些 API？针对不同的应用程序和系统设置明确实现耐用性、性能和可用性权衡的最佳方法是什么？如何使试验不同接口（和接口配置）的过程尽可能无痛？

使用主机驱动的设备管理进行 I/O 调度的最佳方法是什么？在传统的 SSD 中，操作系统和应用程序都不知道垃圾收集的时间。这导致了复杂的解决方法，例如，预测对传统 SSD 的 I/O 请求的延迟，并使用多个 SSD 来对冲缓慢的请求。使用 ZNS SSD，各个区域的性能彼此独立，而不仅仅是共享带宽。

对于当前的第二代 PCIe4 固态硬盘来说已经不是限制，在不久的将来，PCIe5 产品的限制将更小。因此，主机处于完全控制之中，可以精确地安排区域擦除和维护操作。这种灵活性使新策略能够将一个目标优先于另一个目标，例如，读取延迟优先于写入延迟和写入放大。此外，这些策略可能因区域集而异，从而实现多个应用程序或租户的高效共置。应用程序开发人员可以通过哪些接口将这些目标和首选权衡传达给应用程序或操作系统？开发人员如何以有原则的方式选择这些权衡？

研究如何才能最好地利用透明的数据放置？在传统的 SSD 中，应用程序必须依靠技巧来强制 FTL 按预期放置数据。例如，大规模闪存缓存应用程序维护多个对象桶，其中每个存储桶应写入同一擦除块。在传统的 SSD 中，并发写入操作在擦除块上交错，即使它们在逻辑地址空间中相距很远。应用程序已经发展到使用 DRAM 作为缓冲区，将许多写入合并为一个非常大的写入。使用 ZNS SSD，不再需要这些缓冲区。研究如何大规模识别和修改这些应用程序以回收浪费的 DRAM？是否有适用于此类应用程序的通用抽象？

6.2 管理限制

主机应如何管理活动区域限制？当前的 ZNS SSD 限制了可同时写入的区域数量。对于现有的内核文件系统实现来说，这种限制不是问题，这些实现已经实现了足够的并行度，可以充分利用可用的闪存带宽。但是，当 ZNS SSD 在绕过内核的多个应用程序中划分时，这就会成为一个问题。一个简单的策略是为每个应用程序分配固定数量的区域以及固定的活动区域预算。无论如何，这种方法无法针对典型的突发工作负载进行扩展，因为它不允许对这种稀缺资源进行多路复用。是否有按需动态分配区域的好策略？假设制造商将在设备上资源需求适度的情况下制造具有更多区域的设备，那么足够数量的区域是多少？

是否存在工作负载在 ZNS SSD 上的表现比在传统 SSD 上差？ZNS SSD 的一个众所周知的工作负载挑战（在规范最终确定后在原型中发现）是区域的写入指针可能会受到锁争用的影响。对于写入集中在单个区域中的多写入器工作负载，例如持久队列和仅追加数据结构，这是一个问题。发生这种情况是因为规范分配了责任将写入指针移动到主机端软件。ZNS 规范的附加功能解决了此问题：追加命令，该命令不指定偏移量，并允许设备将并发写入序列化到同一区域。鉴于 ZNS 接口的新颖性和迄今为止有限的硬件部署，是否有其他工作负载在 ZNS 上的表现比传统 SSD 差，这仍然是一个悬而未决的问题。研究能否系统地测试代表性和合成工作负载，以发现是否有任何工作负载的性能比 ZNS 差？它们的性能可以通过扩展 ZNS 规范来纠正吗？

ZNS SSD 如何适应将 I/O 任务从主机卸载到专用硬件的最新趋势？在超大规模企业采用 ZNS SSD 的同时，它们也将重新负责转移到主机，它们也将 I/O 过程从主机 CPU 卸载到专用硬件：AWS 的 ASIC，微软的 ARM SoC 和阿里巴巴的 FPGA。系统设计理念中的这种明显矛盾需要学术审查。与 Facebook 用于计算卸载的加速度计类似，研究设想研究如何确定硬件堆栈的哪些部分应该负责 I/O 控制和数据路径上的哪些功能。

7 结论

虽然许多 SSD 研究都集中在传统 SSD 的问题上，但行业已经取得了进步——标准化并开始采用 ZNS。这个转折点为如何充分利用这些新兴设备开辟了令人兴奋和有影响力的研究方向。它还指出了系统社区面临的一个问题：许多研究都是针对大型云提供商等环境进行的，但通常只有那些在工业界或与行业有密切联系的人才能优先访问新兴技术和问题。这对没有这种联系的研究人员不利，但也损害了整个领域。如果接触相关问题的机会受到限制，当解决最紧迫问题的思想家的多样性受到限制时，进展就会受到阻碍。我们希望能够鼓励来自世界各地一系列机构的研究人员参与研究，并组织行业小组，作为讨论新趋势和挑战的渠道。

参考文献

[1] Stavrinos, Theano, et al. "Don't be a blockhead: zoned namespaces make work on conventional SSDs obsolete." Proceedings of the Workshop on Hot Topics in Operating Systems. 2021.

[2] Purandare, Devashish R., et al. "Append is Near: Log-based Data Management on ZNS SSDs." *12th Annual Conference on Innovative Data Systems Research (CIDR'22)*.. 2022.

[3] Lee, Hee-Rock, et al. "Compaction-aware zone allocation for LSM based key-value store on ZNS SSDs." *Proceedings of the 14th ACM Workshop on Hot Topics in Storage and File Systems*. 2022.

[4] Bergman, Shai, et al. "{ZNSwap}:{un-Block} your Swap." *2022 USENIX Annual Technical Conference (USENIX ATC 22)*. 2022.

[5] Bjørling, Matias, et al. "{ZNS}: Avoiding the Block Interface Tax for Flash-based {SSDs}." *2021 USENIX Annual Technical Conference (USENIX ATC 21)*. 2021.