基于 ZNS SSD 的高效键值对数据放置管理

汪易

[1)2)3)](华中科技大学计算机学院 武汉市 中国 430074))

摘 要

基于日志结构合并树(LSM-Tree)的键值存储由于其顺序写入特性而具有较高的 I/O(输入/输出)性能而备受关注。但是由于压缩导致的写操作过多，会导致 SSD (Solid-state Drive)的寿命缩短。因此，有一些研究旨在通过使用 ZNSS SSD 来减少垃圾收集开销;主机可自行决定数据存放位置的 SSD 盘。然而，现有的研究由于没有考虑键值数据的生命周期和热度，在性能提升方面存在一定的局限性。因此研究出一种技术通过根据键值数据的特点来安排 ssd 的空间效率和垃圾收集开销。通过对 RocksDB 的 ZenFS 进行改进，实现了该方法，性能评估结果表明，该方法的空间效率可提高 75%。

关键字：分区名称空间;Flash 转换层; FTL;键值存储;文件系统

Efficient key-value data placement management based on ZNS SSD

Yi Wang

[1)2)3)](Huazhong University of Science and Technology of Computer School, WuHan China 430074)

Abstract

Log-structured merge-tree (LSM-Tree)-based key–value stores are attracting attention for their high I/O (Input/Output) performance due to their sequential write characteristics. However, excessive writes caused by compaction shorten the lifespan of the Solid-state Drive (SSD). Therefore, there are several studies aimed at reducing garbage collection overhead by using Zoned Namespace ZNS; SSD in which the host can determine data placement. However, the existing studies have limitations in terms of performance improvement because the lifetime and hotness of key–value data are not considered. Therefore, in this paper, we propose a technique to minimize the space efficiency and garbage collection overhead of SSDs by arranging them according to the characteristics of key–value data. The proposed method was implemented by modifying ZenFS of RocksDB and, according to the result of the performance evaluation, the space efficiency could be improved by up to 75%.

Key words zoned namespace; flash translation layer; host-level FTL; key–value store; filesystem

1. 引言

近年来，键值存储由于具有高性能、灵活性和简单性等优点，被广泛应用于新兴技术(如深度学习、区块链)。特别是，基于日志结构合并树(LSM-Tree)的键值存储，如 RocksDB 和 LevelDB，可以在基于 NAND 闪存的固态硬盘上获得较高的输入/输出(I/O)性能，这是由于出位置更新导致的顺序写入特性。但是，压缩过程中会发生大量写操作，会降低 SSD 的性能和寿命。此外，在遗留块接口 ssd 的情况下，flash 转换层的垃圾收集进一步增加了写放大(FTLs)。

ZNSssd 使用了一种新的基于 zone 的接口，将 NAND 闪存区域划分为一定大小的 zone，并允许主机直接管理这些 zone。因此，ZNS SSD 可以最大限度地减少垃圾收集开销和写放大，因为数据放置直接由主机执行。目前，正在研究如何使用 ZNS SSD 最大限度地减少 LSM-Trees 的写入放大。

在 RocksDB 的情况下，ZNS ssd 可以通过 ZenFS (ZNS 的简化文件系统)使用。然而，目前的 ZenFS 并没有有效地利用 RocksDB 的信息;因此，不同热度的文件被记录在同一个区域，空间利用率不高。此外，由于当前 ZenFS 不支持垃圾收集，RocksDB

收稿日期：2023年1月18日

无法管理 ZNS ssd 上的大键值集。

因此，提出一种方法，通过优化基于 lsm－tree 的键值存储的基于区域的数据放置，在压缩过程中快速保护空区域。此外，还提出了一种考虑数据生命周期的垃圾收集技术，以最大限度地减少写放大。

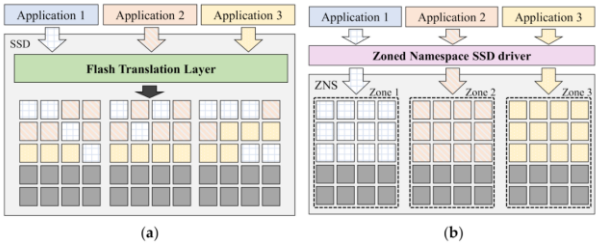
2. 背景、相关工作和挑战

2.1 ZNS SSD

ZNS SSD 是基于 ZAC/ZBC 标准提出的，用于 SMR(叠瓦式磁记录)硬盘驱动器(HDD)和开放通道 SSD。与传统硬盘不同，SMR 硬盘中的磁道不是分开的，而是重叠的。虽然这为 HDD 提供了更大的容量，但它不能提供就地更新。不允许就地更新将禁止 HDD 使用传统数据块层。因此，针对 SMR HDD 提出了新的 I/O 方法，称为分区 ATA 命令(ZAC)和分区块命令(ZBC)。

每个标准都将其空间拆分为分区的单位。每个分区的 I/O 独立于其他分区，并且分区只能接受其中的顺序写入。标准建议使用驱动器管理、主机管理或主机感知的方法来高效地管理区域。驱动器管理方法使用软件固件，例如 SSD 中的闪存转换层来支持传统数据块接口层。主机管理的方法使主机能够直接管理分区，并且仅允许在分区中执行顺序写入。因此，主机可以通过使用主机信息将数据放置在磁盘的适当位置。这使磁盘能够充分利用其性能。主机感知方法是一种混合方法，具有驱动器管理和主机管理的特点。因此，它对在分区中写入数据没有任何限制，并且主机可以管理每个分区状态。然而，这种方法伴随着一个问题，它有时会产生长尾延迟

由于这些优势，ZAC/ZBC 扩展了分区命名空间(ZNS)接口。ZNS SSD 是一种通过 NVME 使用分区命名空间接口的 SSD。如图所示，ZNS SSD 将其基于空间的单元抽象为 ZONE。主机利用此抽象分区来管理和服务 I/O。这意味着 SSD 只需要非常简单的 FTL，或者根本不需要 FTL。与 ZAC/ZBC 规范一样，ZNS SSD 分区通常也只允许顺序写入其空间。对顺序写入的限制实现了 ZNS SSD 中的匿名写入。它显著减小了逻辑到物理(L2P)映射表的大小。这些优势增强了固态硬盘的性能。



此外，ZNS SSD 使主机能够预测 SSD 的延迟。这是因为传统的 SSD 有一个复杂的 FTL，它成为供应商的一个完全的黑匣子。然而，由于 ZNS SSD 只需要非常简单的 FTL 或根本不需要 FTL，因此主机可以利用 ZNS SSD 公开的信息来比传统 SSD 更好地预测 ZNS SSD 的延迟。此外，ZNS SSD 可以选择数据放置位置，从而提高磁盘的性能。例如，如图 a 所示，FTL 随机放置常规 SSD 数据。然而，如图 b 所示，ZNS SSD 数据根据应用进行分组。让我们假设应用程序 1 生成非常密集的工作负载。在此工作负载中，传统 SSD 需要写入放大来擦除数据块。另一方面，对工作负载数据进行分组由 ZNS SSD 中的主机执行。这意味着 SSD 不需要写入放大来擦除数据块。由于这些优点，它作为下一代接口方法受到关注，并将被添加到 NVME 标准中。

2.2 ZNS SSD 挑战

ZNS 固态硬盘面临两大挑战。

（1）主机软件需要显式地处理分区，如分区重置、打开、读取、写入和分区垃圾收集。

（2）ZNS SSD 有一个独特的约束，称为顺序写入约束。像 SMR(瓦片磁记录)驱动器一样，要求按顺序写入区域中的所有数据。

（3）数据放置：为了在利用率较高的 SSD 中放置尽可能多的低级别 SST。因此，SST 存储管理需要一种动态数据布局算法。特别是，对于混合分区存储，在不影响性能可预测性的情况下正确地将 SST 放置在 SSD 或 HDD 中是至关重要的，这是分区存储设备的关键设计特征。

（4）数据迁移：跨 SSD 和 HDD 迁移 SST 是不可避免的(例如，将频繁读取的 KV 对象从 HDD 移动到 SSD)。然而，混合分区存储中的数据迁移尤其重要，因为其分区容量很大(例如，数百 MiB)。如果迁移仅移动区域的部分数据，则会导致区域内的碎片。另一方面，如果迁移按区域移动数据，则会产生大量 I/O 流量，从而干扰前台活动。

（5）数据缓存：有效的数据放置和数据迁移可以减少硬盘的读取量，但频繁读取的 KV 对象可

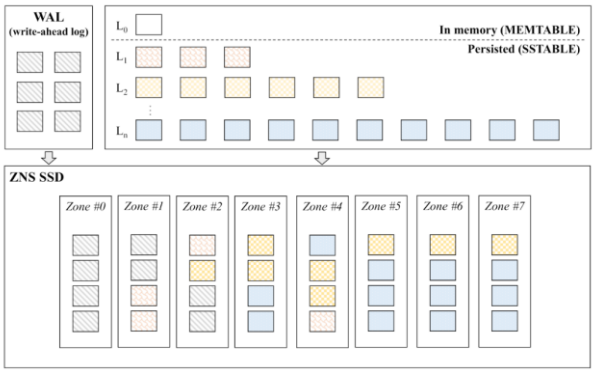
能分散在 HDD 中的 SST 上。应专门缓存此类频繁读取的 KV 对象，以实现快速读取性能。扩大内存中的高速缓存空间是一种可能的解决方案，但代价高昂。在 SSD 中缓存 KV 对象可以解决内存缓存的紧张问题。然而，由于 ZNS SSD 的只附加写入，需要根据仅附加接口来设计具体的缓存策略。此外，应该仔细平衡 SSD 空间，该空间现在由 WAL、低级别 SST 和缓存的 KV 对象共享。

2.3 RocksDB 和 LSM-Trees

Facebook 开发了 RocksDB，这是一个基于日志结构合并树(LSM-Tree)的开源键值存储。与其他键值存储不同，它支持在刷新和压缩例程中使用多线程。

LSM-Tree 减少了使用非位置更新的随机写入，并具有一个压缩例程来收集由非位置更新生成的无效数据。它们使 LSM-Tree 具有快速的写入性能，并有助于高效地利用空间。这些是 LSM-Tree 是键值存储后端的主要原因。

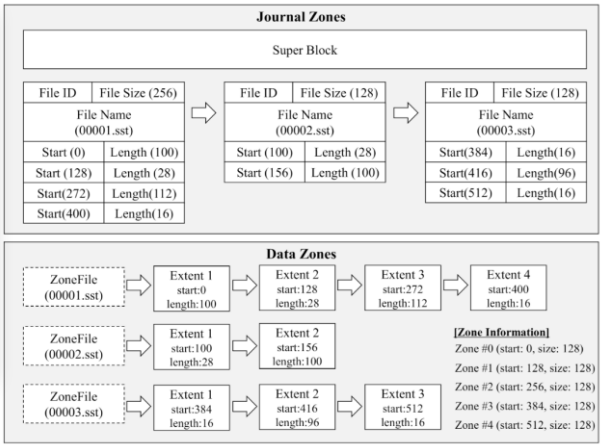
如图显示了由 Memtable、预写日志(WAL)文件和排序顺序表(SSTable)文件组成的 RocksDB 体系结构。Memtable 是存在于内存中的写缓冲区，它聚合用户写 I/O，直到达到缓冲区阈值。缓冲创建了粗粒度的写 I/O，并允许更多地利用磁盘。SSTable 由压缩结果生成。它是带有 LSM-Tree 的 RocksDB 的核心。RocksDB 压缩使用水平压缩。分级压缩限制了重叠表的数量，以实现最佳读取性能。然而，在执行排序合并时，它会遇到高写入放大问题。因此，位于级别 1 的 SSTable 由于写入放大而出现问题。



3. ZenFS

3.1 ZenFS

如图显示了 ZenFS 管理每个分区，将其划分为日志区和数据区。日志区中的区域数是在编译时静态定义的。相比之下，数据区中的区域数定义为区域总数减去日志区区域数。



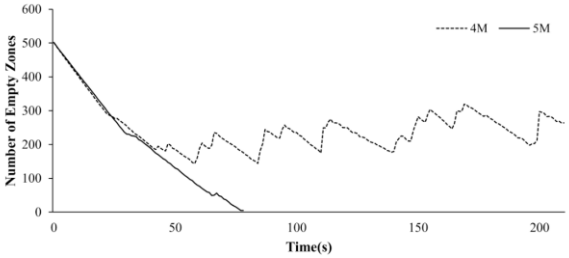
日志区包含 ZenFS 超级数据块和 zonefile 元数据。超级数据块区域具有 ZenFS 创建信息。日志区中的其余区域具有区域文件元数据，例如 WAL 和 SSTable 映射区域信息。区域文件元数据包含文件 ID、文件大小、文件名和区元数据数组。

Zonefile 是 ZenFS 处理 RocksDB 文件的抽象文件。它的元数据被上载到内存，并写入日志区，以便在系统崩溃时恢复。其数据以数据区的范围为单位写入。区具有写入区域中的区域文件数据的连续部分。在该范围中，当数据第一次被写入时，“Start”表示写入指针(WP)在区域中的位置，而“Length”表示区域中连续写入的大小。

3.2 ZenFS 的限制

研究发现，ZenFS 没有利用主机信息，也没有垃圾收集逻辑。为了验证这一事实，创建了一个 ZNS SSD 仿真环境，并评估了 ZenFS 的当前状态。该实验分别使用 400 万(约 3.1 GB)和 500 万(约 3.9 GB)键-值对进行。

实验结果如图所示。该图显示了 400 万个键-值对的插入没有任何问题。但是，发现在覆盖完成之前，有 500 万个键-值对无法执行工作负载。结果表明，ZenFS 没有有效地利用 ZNS SSD 的空间。

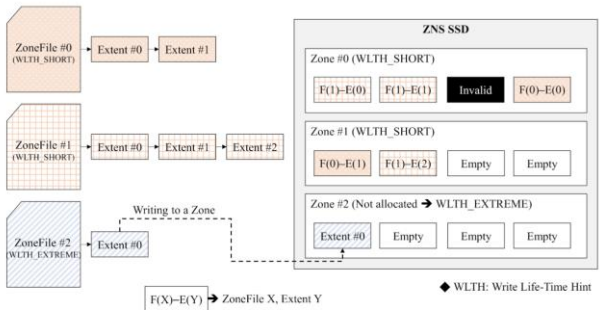


此外，根据图中插入的 400 万个键-值对结果，发现 ZenFS 在开始覆盖时只能保留少量可用区域。这是因为 ZenFS 的区域分配方法没有正确考虑区域文件的生存期，因此没有有效地回收区域。

因此,研究发现 ZenFS 不能有效地利用其空间。此外，它还要求根据数据生命周期拆分数据放置。为此，针对 ZNS SSD 中的键值存储，提出了基于生命周期的区域分配方法和垃圾回收方法。

4.方法设计

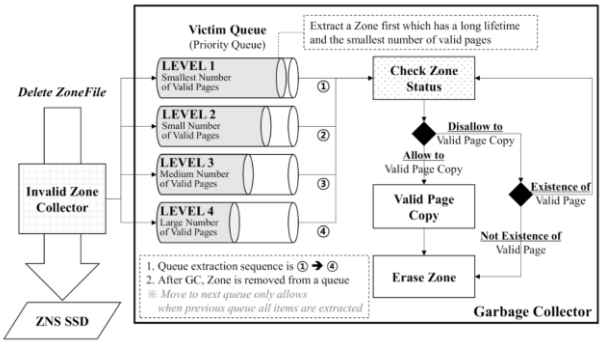
4.1 基于生命周期的区域分配方法



如图显示了根据生命周期的区域分配方法。假设在图中为 Zonefile#2 分配了一个区域来写入 Extent#0。首先，我们按顺序检查 Extent#0 中每个区域的状态。Extent#0 被排除在区域分配的候选区域列表之外，因为所有区域都已用尽。由于 Extent#1 具有空闲页面，因此我们将主机声明的其写入生命周期提示(Wlth_Extreme)与包含 Extent#0 的 Zonefile#2 的写入生命周期提示(Wlth_Short)进行比较。如果生存期相同，则分配该区域。然而，正如我们在图中所看到的，由于生存期不同，Extent#1 被排除在外。最后，Zone#2 是一个完全空的分区，未从主机分配任何写入生存期提示；因此，它被分配为用于写入 Zonefile#2 的 Extent#0 的分区。作为预防措施，无论分区顺序如何，具有与请求数据相同的生存期且具有某些空闲的分区将优先于空分区进行分配。

这种方法尽可能地将一个 Zonefile 放置在相同的区域中。此外，即使将不同的 Zonefile 放置在单个 zone 中，这也不会混合不同寿命的 Zonefile；因此，物理删除不会延迟。这有效地解决了由于将不同寿命的 Zonefile 混合到一个 ZOne 中而导致的物理删除延迟的问题。它减少了有效的数据复制量，提高了性能。

4.2 ZNS SSD 的垃圾收集器



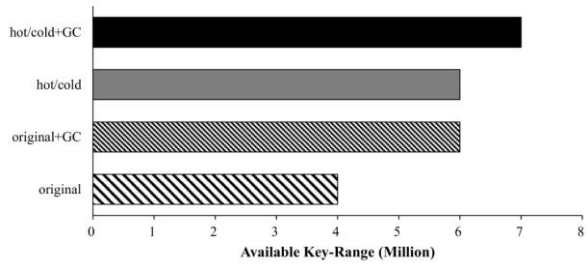
垃圾收集器的操作如图所示。当对任何 Zonefile 发出删除命令时，每个 zone 都会记录包含与 Zonefile 关联的数据的区域中的数据已失效。包含无效数据的 Zone 将来可能会受到垃圾回收；因此，无效区域收集器检查 Zone 的有效数据数量，并将它们插入受害者队列的适当位置。

受害队列由多个队列组成，每个队列具有不同数量的无效数据的。也就是说 1 级队列中的区域具有最少数量的有效数据，而 4 级队列中的区域具有最大数量的有效数据。首先选择寿命最长、有效数据最少的区域作为受害者。它使用优先级队列，因此可以快速搜索有效数据较少的区域。这降低了包含短生命周期数据或具有大量有效数据的区域受到垃圾回收的可能性。因此，它可以防止对有效数据进行不必要的复制

5. 实验

5.1 空间利用率分析

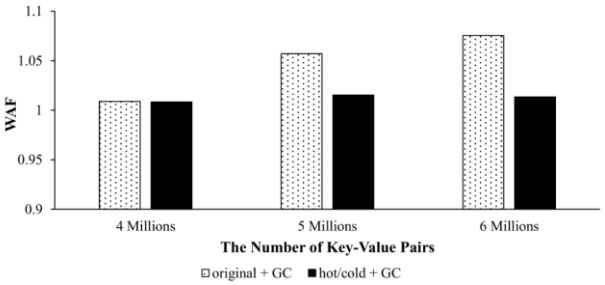
为了评估上一节的方法如何有效地利用 ZNS SSD 的空间，检查了四种不同实现中有多少键范围可用于“fillseq + overwrite”工作负载。



评估结果如图所示。已确认常规方法可用于多达 400 万个键范围。“original + GC”(只是添加到常规方法中的垃圾收集(GC)逻辑)和“hot/cold”(它没有垃圾收集逻辑，但根据生存期分配区域)使用了多达 600 万个键范围。经证实，“″hot/cold + GC”具有拟议技术的所有特征，可用于多达 700 万个关键范围，比传统方法增加了 75%。

这意味着，如果唯一键的数量超过 400 万(约 3.2 GB)，则具有 16 字节键和 800 字节的值的 8 GiB ZNS SSD 不能自由覆盖。但是，通过采用垃圾收集或基于生存期的区域分配方法之一，它可以自由覆盖高达 600 万个(约 4.9 GB)的唯一键。使用这两个键，可以自由覆盖多达 700 万个(约 5.7 GB)的唯一键。

上述评估结果表明，所提出的方法缓解了由于不同寿命的 Zonefile 混合在一个 Zone 中而造成的延迟物理删除。因此，可以更快地回收区域，并可以确保更多的空白区。该区域分配方法考虑了数据的生命周期，通过将短生命周期数据和长生命周期数据分开来最小化物理删除延迟问题。另一方面，垃圾收集是一种解决方案后的方法，它通过从包含无效数据的区域中提取长生命周期的有效数据来减少物理删除延迟。



上图表明，将垃圾收集和按生存期分配区域的方法相结合的方法对于有效利用空间非常有用。该图是对垃圾收集期间有效数据拷贝产生的额外写入进行分析的结果，同时将键范围从 400 万更改到 600 万(以百万为单位)。图表的写入放大系数(WAF)是指生成的额外写入数与现有写入数之比。

在传统方法中增加垃圾收集的方法中，晶片面积随着键范围的增加而增加。这是由于这样的问题：当将具有长生存期的有效数据复制到另一个区域时，它们在被复制到包含具有短生存期的有效数据的区域后立即被再次复制。然而，提出的方法没有这个问题。因此，即使关键范围很大，它也显示出一致的晶片数量。因此，提出的方法有助于减少不必要的写入，并导致有效的空间利用。

5.2 I/O 性能

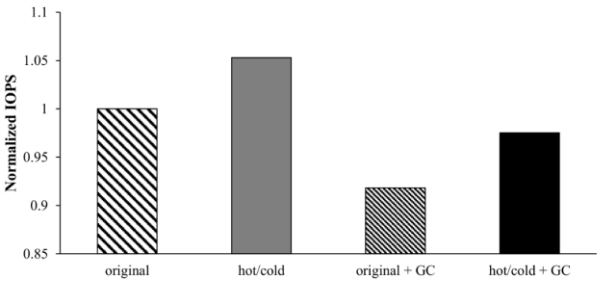


图 16 显示了 400 万个键上“fillseq + overwrite”工作负载的标准化 I/O 性能。结果表明，与传统方法相比，基于寿命的分区分配方法将 IOPS 提高了 5%。提出的的垃圾收集逻辑可以使用更多的键范围，但表明有效的页面复制会导致性能下降。在传统方法中引入垃圾收集会导致 9%的性能下降，而提出的“hot/cold + GC”技术则会导致 3%的性能下降。

产生上述评估结果的原因是区域分配延迟，没有垃圾收集的生存期的区域分配方法会比比传统方法更小的区域分配延迟。也就是说基于寿命的区域分配方法由于较少地发现空闲区域而被中断，这导致了比传统方法更好的性能

6. 总结与展望

针对 ZNS SSD 中的键值存储，提出了一种基于生存期的区域分配方法和垃圾回收方法。基于数据生命周期的区域分配方法利用了 LSM-Tree 的特点。此外，由于 ZenFS 没有垃圾收集逻辑，还增加了垃圾收集逻辑来提高 ZNS SSD 的空间利用率。根据实验结果，考虑到数据的生命周期，该方法通过分区划分和垃圾回收，更有效地利用了 ZNS SSD 空间。

参 考 文 献

[1] D. Wu, B. Liu, W. Zhao and W. Tong, "ZNSKV: Reducing Data Migration in LSMT-Based KV Stores on ZNS SSDs," 2022 IEEE 40th International Conference on Computer Design (ICCD), Olympic Valley, CA, USA, 2022, pp. 411-414, doi: 10.1109/ICCD56317.2022.00067.

[2] Oh, G.; Yang, J.; Ahn, S. Efficient Key-Value Data Placement for ZNS SSD.Appl.Sci.2021, 11, 11842. https://doi.org/10.3390/app112411842

[3] Oh, G.; Yang, J.; Ahn, S. Efficient Key-Value Data Placement for ZNS SSD.Appl. Sci. 2021, 11, 11842. https://doi.org/10.3390/app112411842

[4] Gunhee Choi, Kwanghee Lee, Myunghoon Oh, Jongmoo Choi, Jhuyeong Jhin, and Yongseok Oh. 2020. A new LSM-style garbage collection scheme for ZNS SSDs. In Proceedings of the 12th USENIX Conference on Hot Topics in Storage and File Systems

(HotStorage'20). USENIX Association, USA, Article 1, 1.

[5] M. Bjørling, A. Aghayev, H. Holmberg, A. Ramesh, D. L. Moal, G. R. Ganger, and G. Amvrosiadis. ZNS: Avoiding the block interface tax for flash-based SSDs. In Proc. of USENIX ATC, 2021.

[6] Zhong, W.; Chen, C.; Wu, X.; Jiang, S. REMIX: Efficient range query for LSM-trees. In Proceedings of the 19th USENIX Conference on File and Storage Technologies (FAST 21), Santa Clara, CA, USA, 23–25 February 2021; pp. 51–64.

[7] H. Chen, C. Ruan, C. Li, X. Ma, and Y. Xu. SpanDB: A fast, cost-effective LSM-tree based KV store on hybrid storage. In Proc. of USENIX FAST, 2021.