ZNS SSD的compaction优化综述

侯玉峰

摘　要　Zoned Namespace(ZNS) SSD是近几年新出现的基于传统SSD的存储设备，其消除了传统SSD设备端的垃圾回收机制，Over-Provisioning(OP)空间，但是ZNS SSD在主机端需要进行垃圾回收和数据放置管理，主机端变得更加复杂。因为ZNS SSD以zone为粒度进行清除的，如果其无法合适的放置数据，即使得zone中的数据同时失效，那么主机端将会面临更严重的写放大问题，所以ZNS SSD必须优化其compaction策略，尽量降低compaction带来的开销。本文展示了ZNS SSD compaction优化领域的研究进展，介绍了四种知名的compaction优化策略，分析了这四种方法的技术路线，本文旨在为研究人员提供关于ZNS SSD compaction优化领域的系统且有价值的介绍。

关键词　ZNS SSD，compaction优化

1　Introduction

块接口将存储设备呈现为固定大小的逻辑数据块的一维数组，可以以任何顺序读取、写入和覆盖[1]。最初是为了隐藏硬盘介质特性和简化主机软件而引入的，块接口适用于多代存储设备，并允许在存储设备接口的两侧进行重大创新。然而，块接口和当前存储设备特性之间存在显着的不匹配。

对于基于闪存的SSD来说，支持块接口的导致性能下降和操作成本飞速增长[1]。这些成本是由于基于接口的操作和底层闪存介质的性质不匹配造成的。尽管单个逻辑块可以被写入闪存，但介质必须以较大的单位（称为擦除块）的粒度进行擦除。固态硬盘的闪存转换层（FTL）使用大量的DRAM进行动态逻辑到物理页面映射结构，以及保留大量的硬盘介质容量（OP空间）来降低擦除块数据的垃圾回收开销。垃圾回收往往会限制吞吐量，接口不匹配会导致写放大，从而带来性能的不可预测性和高尾部延迟。

ZNS最近作为基于闪存的SSD 的新接口标准被引入[2]。ZNS SSD 将逻辑块分组为区域，而不是单个逻辑块阵列。区域的逻辑块可以按随机顺序读取，但必须按顺序写入，并且必须在重写之间擦除区域。ZNS SSD 对齐区域和物理介质边界，将数据管理的责任转移到主机。这消除了对设备内垃圾收集的需求以及对资源和容量过度配置(OP) 的需求。此外，ZNS 对主机软件隐藏了设备特定的可靠性特征和媒体管理的复杂性。

相比黑盒化的SSD，ZNS的优势显而易见，它将内部的闪存物理结构暴露给应用侧，为高效的数据管理提供了诸多可行性，其目前的生态也在逐渐完善中，有机会成为未来的主流存储设备[3]。ZNS将设备端的GC移动到主机端，虽然消除了设备端的写放大，但是为主机端带来了更大的复杂性，其主要问题为主机侧垃圾回收带来的写放大，因为Zone的大小远比块要大，如果不对Zone里的数据好好管理，其写放大带来的损失将会更加严重，因此如何优化compaction操作从而降低写放大开销是当今关于ZNS的研究热点，本文整理了近几年该方向的优秀文章，向大家介绍了该方向的最新进度。

2　Background

分区存储模型最初是为叠瓦式磁记录(SMR) HDD引入的，它的诞生是为了创建无需与块接口兼容性相关的成本的存储设备。前面详细介绍了与SSD 相关的成本，接下来继续描述分区存储的基本特征[1]。

分区存储模型的基本构建块是区域。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按顺序写入，并且要启用新的写入，必须显式重置。写入约束由每个区域的状态机和写入指针强制执行[1]。

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，在写入时转换到OPEN状态，最后在完全写入时转换到FULL。例如，由于设备资源或介质限制，设备可以对同时处于开放状态的区域的数量施加限制。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。CLOSED区域仍然是可写的，但必须再次转换到OPEN状态，然后才能提供额外的写入[3]。

区域的写指针指向可写区域内的下一个可

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。主机发出的任何写命令(1) 不是从写指针开始，或(2) 写入处于FULL 状态的区域将无法执行。当一个区域被重置时，通过reset zone命令，该区域被转换到EMPTY状态，它的写指针被更新到该区域的第一个LBA，并且之前写入的用户数据不再可访问。区域的状态和写指针消除了主机软件跟踪写入区域的最后一个LBA 的需要，从而简化了恢复[3]。

尽管跨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，并允许区域具有小于区域大小的可写容量。这使ZNS SSD的区域大小能够与SMR HDD引入的二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置区域。

活动区域限制为可以处于OPEN或CLOSED状态的区域添加了硬限制。然而SMR HDD 允许所有区域保持在可写状态（即CLOSED），但基于闪存的媒体的特性，例如程序干扰，要求该数量受限于ZNS SSD。

因为Zone有只能顺序写，写后无法更改的特征，所以当Zone区域中有一部分数据失效时，失效空间无法利用，要想进行更新操作或者主机端垃圾回收时，必须将当前Zone里的所有有效数据迁移到新的Zone里，带来极大的资源开销和性能浪费，如何避免这种开销时当前ZNS领域的研究热点[2]。

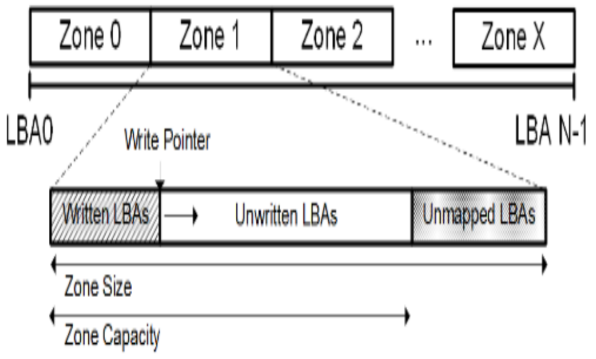


Fig. 1 存储设备LBA地址空间内的Zones,每个Zone里的WP只能递增，且被显示重置命令重置

3 ZenFS

从ZNS的描述可以发现，LSM-Tree这种追加写/异地更新的方式特别适合ZNS架构，而RocksDB是基于LSM-Tree概念的键值数据库系统，因此有人想将RocksDB适配到ZNS上，由于RocksDB不能运行在裸设备上，因此提出了ZenFS[1]，其作为存储后端集成在RocksDB中，RocksDB通过它与ZNS SSD进行交互，ZenFS的主要架构如图2所示，将在下面描述。

1) **Journaling and Data Zones** Zenfs 定义了两种类型的zones: 其中Journal Zones 用来管理文件系统的元数据，包括异常时恢复文件系统的一致性状态，维护文件系统的Superblock，映射WAL和数据文件到Zone中的Extent Map以及管理Extent，Data Zones用于存储文件内容（如SST）。

2) **Extent** 是一个大小可变、块对齐、连续的LBA，按顺序写入Data Zone，其中包含与特定标识符关联的数据。每个Zone可以存储多个Extents，但Extent不会跨越Zone。内存中的数据结构追踪Extent到Zone的映射，一旦删除了Zone中所有已分配Extent的文件，就可以重置和重用该Zone。

3) **SuperBlock** 用于标识属于当前磁盘的Zenfs，当磁盘的盘符重启或者外部插拔发生变化的时候仍然能够识别到这上面的文件系统。用来初始化Zenfs或者从磁盘异常恢复Zenfs的状态。

在RocksDB 中，数据热度从L0-Ln依次下降，为了减少写放大，ZenFS在分配Zone时，采用尽力而为的算法来选择存储RocksDB 数据文件的最佳区域。RocksDB 在写入文件之前为文件设置writelifetime 提示来分隔WAL 和SST 级别。在第一次写入文件时，会分配一个数据区域用于存储。ZenFS 首先尝试根据文件的生命周期和存储在Zone中的数据的最大生命周期来查找Zone。只有当文件的生命周期小于存储在Zone中的最旧数据时，匹配才有效，以避免延长区域中数据的生命周期。如果找到多个匹配项，则使用最接近的匹配项。如果没有找到匹配项，则分配一个空区域。如果文件填满了分配区域的剩余容量，则使用相同的算法分配另一个区域。通过这种分配方法，在进行GC时，可以极大的减少数据的Copy操作；Stream SSD考虑的也是类似的策略，根据数据特性，将其绑定到不同的Stream中，以达到减少GC的策略；但是也可以看出，如何将数据正确分类，将不同类型的IO放入到对应的Stream或着Zone内，对GC的效率至关重要。

4 ZNS+

新的存储接口需要改进软件栈。对于ZNS，我们需要修改两个主要的IO栈组件，文件系统和IO调度器。首先，必须用附加日志记录文件系统(如日志结构文件系统(LFS))替换现有的更新文件系

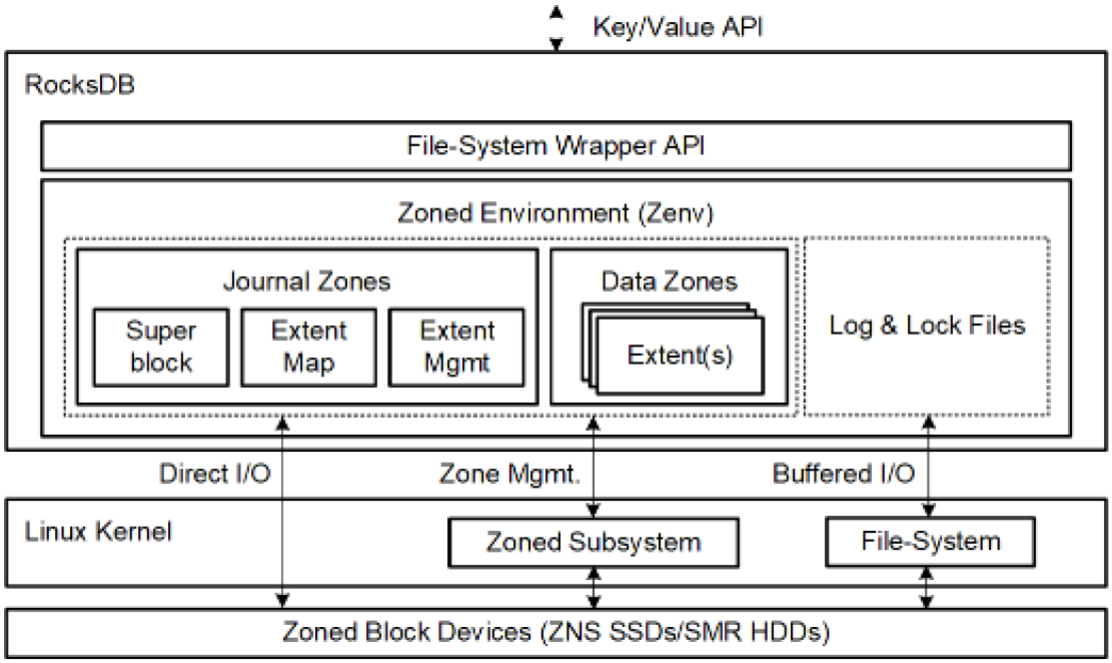


Fig. 2 ZenFS架构

统(如EXT4)，以消除随机更新。因为LFS的一个段是通过追加日志按顺序写入的，所以每个段可以映射到一个或多个区域。其次，IO调度器必须保证按顺序提交区域的写请求。

在LFS的追加日志模式下，脏段的废弃块必须通过段压缩(也称为段清理或垃圾收集)回收，将段中的所有有效数据移动到其他段，使段干净。压缩会调用大量的复制操作，特别是在文件系统利用率较高的情况下。ZNS的特征使我们利用主机端GC以换取尽少的使用设备端GC。主机端GC的开销比设备端GC高，因为主机级块复制需要IO请求处理、主机到设备的数据传输以及读取数据的页面分配。另外，段压缩需要调整文件系统元数据来反映数据重定位。此外，段压缩的数据复制操作是批量执行的，因此，许多挂起的写请求的平均等待时间很重要。因此，我们可以观察到当前的存储系统在使用ZNS时只考虑在设备侧带来的收益，而忽略主机侧引入的复杂性。为了简化SSD的设计，所有复杂的东西都传递给主机。

LFS感知的方案ZNS+支持compaction加速和compaction避免[4]，以减少主机端执行segment compaction(GC)时引起的开销。

4.1 compaction加速

4.1.1 普通段压缩

段压缩需要四个子任务:受害段选择、目标块分配、有效数据复制和元数据更新，如图3(a)所示。第一个子任务选择找到压缩代价最低的段(①)。第二个子任务从目标段分配连续的空闲空间(②)。第三个子任务通过主机发起的读写请求将受害段中的所有有效数据移动到目标段，这将在主机和存储设备之间产生大量的数据传输流量(③)。第四个子任务文件系统将修改后的元数据块和节点块写入存储，以反映数据位置的变化，然后写入检查点块(④)。我们可以将数据复制任务卸载到SSD，其他任务交由主机文件系统执行。因为设备端数据复制比主机端数据复制快。对于compaction加速,ZNS+允许主机通过zone_compaction命令将数据拷贝任务卸载到SSD上。

4.1.2 基于IZC的段压缩

图3(b)展示了在Internal Zone Compaction(IZC)模式下段压缩过程。普通端压缩任务由复制卸载所代替(❸)，即发送zone_compaction命令来传递块复制信息。因为目标数据没有加载到主机页缓存中，所以不需要分配相应的页缓存。SSD内部控制器可以有效地调度多个读和写操作，同时最大限度地提高闪存芯片的利用率。因此，可以显著降低段压缩延迟。此外，存储中的块拷贝可以利用copyback操作。

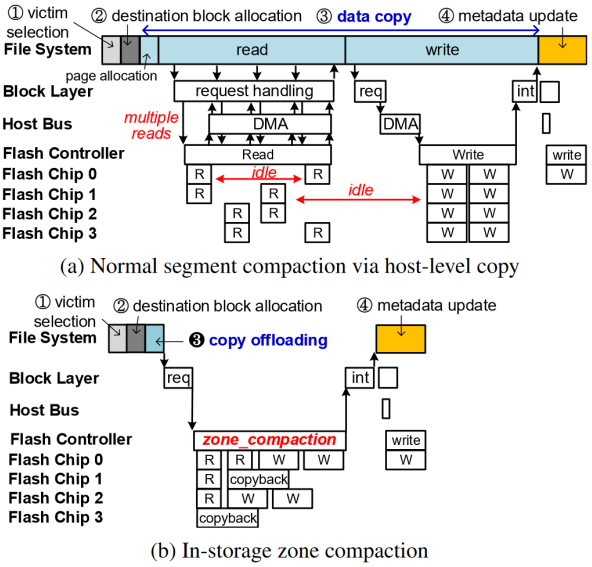


Fig. 3 ZNS上的段压缩(a)vs.ZNS+(b)

4.2 compaction避免

4.2.1 internal plugging

为了避免段压缩，LFS可以使用另一种回收方案，称为threaded logging，它通过覆盖新数据来回收现有脏段中的无效空间。它不需要清理操作，但会对段产生随机覆盖。这与ZNS的特征是相悖的。但令人宽慰的一点是threaded logging按照块地址的递增顺序访问脏段的空闲空间，因为它首先消耗块的较低地址。因此，它的访问模式是稀疏顺序的。当threaded logging重写一个端时，SSD固件可以读取写请求之间跳过的块并将其插入到主机发送的数据块中，从而达成向目标端发送密集顺序写请求的目的，这种技术成为*internal plugging*。因为plugging操作是SSD-internal的，延迟比主机端复制延迟低。

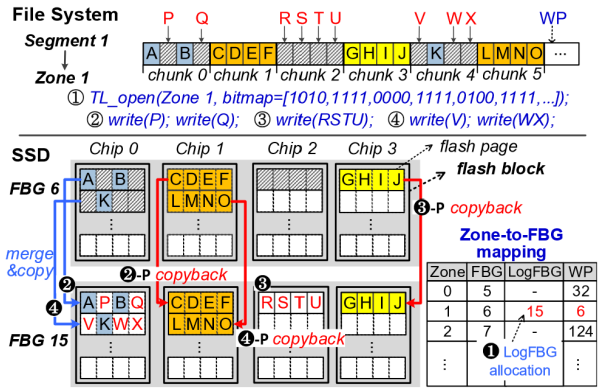


Fig. 4 threaded logging跳过块插入

图4展示了internal plugging的一个示例。段1现在映射到文件系统中的Zone 1, FBG 6分配给Zone 1，如Zone-FBG映射表所示。FBG由四个flash块组成，每个flash块位于不同的flash芯片中。主机文件系统为threaded logging分配段1，并向无效块发送写请求以重写它们，同时跳过有效块。例如，块0中A和B的块为跳过块。

4.2.2 Opening Zone for Threaded Logging

ZNS+提出一种特殊的命令，叫做TL_open(open zones,vaild bitmap)，该命令传递选择的目标zone的vaild bitmap(①)。一旦通过TL_open通知分配的段，threaded logging只重写在传输bitmap上标记的无效块。因此，SSD可以提前识别threaded logging要跳过的块，并在接下来的写请求到达之前执行插入。

4.2.3 LogFBG Allocation

由于TL_opened zone会被覆盖，所以ZNS+ SSD会重置zone的WP，并分配一个新的FBG(称为LogFBG)，将新的数据写入zone。例如在图4中,SSD为Zone 1分配一个LogFBG，FBG 15(❶)。对于TL_opened区域，该区域的数据块分布在两个FBG中，即原始FBG和LogFBG。当TL_opened区域的无效块被threaded logging覆盖时，所有有效块都被复制到LogFBG中。

4.2.4 LBA-ordered Plugging

当SSD在图4(②)中接收到两个对chunk 0的写请求时，它从FBG 6中读取跳过块A和B，并将它们与主机发送的P和Q块合并，然后在LogFBG上写入一个完整的块(❷)。在处理完对chunk 0的写请求后，SSD可以通过检查分区的valid bitmap来感知chunk 1将被跳过。为了提前为写块2的请求准备好WP，必须将跳过的块复制到LogFBG中。因此，ZNS+ SSD在处理完写请求后，如果以下逻辑块在valid bitmap中被标记为有效，则会将其复制到LogFBG中，同时调整WP(❷-P、❸-P、❹-P)。上述方法被称为*LBA-ordered Plugging*。

5 CAZA

适配于ZNS SSD的RocksDB通过ZenFS使用LIfetime-based Zone Allocation algorithm(LIZA)[1]将具有相似失效时间的数据放置在同一个zone中，从而在回收zone时最大限度地减少有效数据复制的GC开销。然而，LIZA根据LSM-Tree的层次结构级别预测每个SSTable的生命周期来分配区域，由于SSTable生命周期的不准确预测，在最小化写放大(WA)问题方面非常低效。具体来说，LIZA有两种情况无法避免:

1)在LSM-Tree的相邻级别上具有重叠key范围

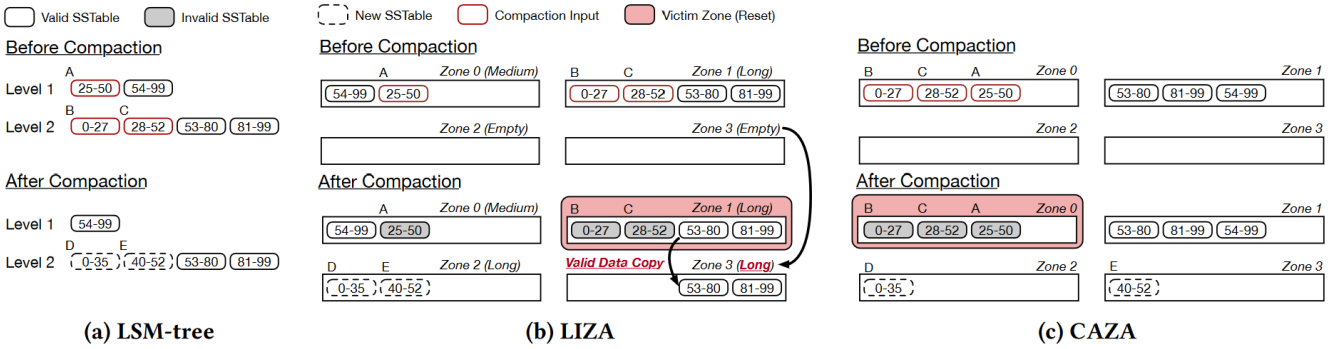


Fig. 5 展示了CAZA相比LIZA在Zone清除的高效性

的sstable可以放置在不同的区域中，并在压缩时同时失效。

2)压缩可以使具有不同生命期提示值的zone之间的sstable失效。

图5(a)和(b)展示了前面提到的发生在LIZA中的两种情况。在进行compation之后，B和C需要回收，但是根据ZenFS的策略，level2层的数据都放在zone 1中，在回收时，还有其他有效sstable存在，所以需要将zone 1中的其它数据迁移到一个空的zone中。跨区域的压缩导致压缩后每个区域的部分失效，这反过来增加了区域清理期间的数据复制开销。

LIZA最致命的限制是缺乏在LSM-Tree中如何使sstable失效的设计考虑。Compaction-Aware Zone Allocation algorithm(CAZA)是基于对compaction如何在LSM树中选择、合并和使sstable失效的密切观察而设计的[5]。因此，CAZA的设计考虑了LSM-Tree的压缩，在同一时间段执行compaction操作的sstable的无效时间是一致的，因此将他们放在同一个zone中，从而在gc时在同一个zone的数据基本都是无效的，就会有较少的数据迁移，从而极大的减少写放大；

图5(c)展示了如何解决图5(b)中LIZA存在的问题。有着重叠key范围的sstable(A,B,C)放在同一个zone 0中。在compaction后，在zone 0中的A，B，C同时失效，当触发zone清除时，可以直接将zone 0清除而不用进行任何数据迁移。下面描述了CAZA的操作：

1)假设一个新的sstable S从level n开始，CAZA在level n+1寻找与sstable S有重叠key范围的sstables($S_{overlap}$)。

2)CAZA寻找包含有$S_{overlap}$中sstable的zone集合Z。

3)将Z根据属于$S_{overlap}$的sstable数量来降序排列。

4)CAZA按序从Z中分配zone。

CAZA可能无法找到满足上述步骤的zone，考虑下述两种情况。**ZC 1:**一些sstables有着完全新的key范围。**ZC 2:**如果与新的sstable有重叠key范围的sstables所属的zones中没有多余空间。针对**ZC 1-2**，会分配新的空zone。

总的来说，sstable需要满足两个条件满足才会分配在一个zone中，1) sstable必须在相邻的level层，2) sstable之间的key必须有交叠。

6 LL-Compaction

Lifetime-leveling LSM-tree Compaction[6]面临着这样的问题：即位于不同level的sstable分配在同一个zone中，导致zone中sstable的寿命并不一致，无法同时失效，在进行GC时会导致写放大。如图6所示，每个zone在GC时都存在有效数据。

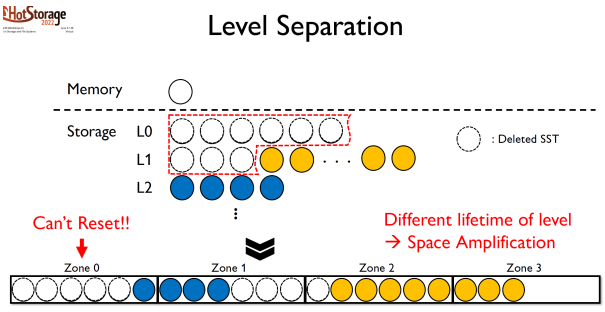
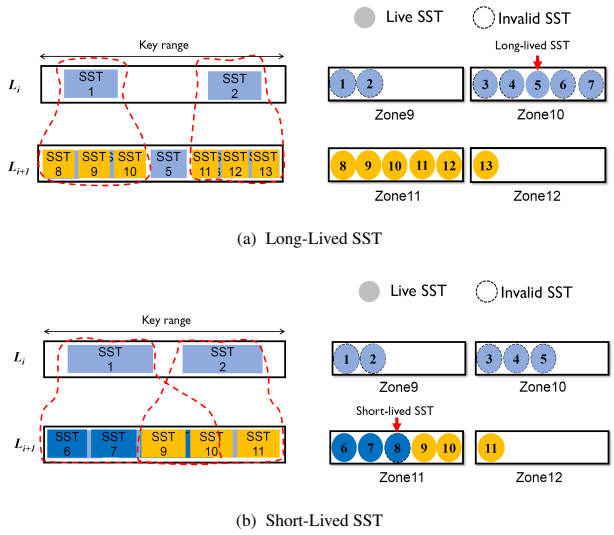


Fig. 6 不同level的sstable分配在同一zone时的情况

即使处在同一层，sstable的寿命也不一致，比如，某些sstable与目标target的key不存在重叠，所以一直不会被用来compaction，比如图7(a)中的SST 5，这种情况被称为Long-Lived SST。

还有一种情况，经过compaction操作之后会生成一个新的sstable，但是紧接着在下次compaction时，这个sstable刚好被选中（key与

目标sstable交叠），它很快被delete掉，比如图7(b)的SST 8，这种情况被称为Short-Lived SST。



(a) Long-Lived SST

(b) Short-Lived SST

Fig. 7 导致写放大的两种情况

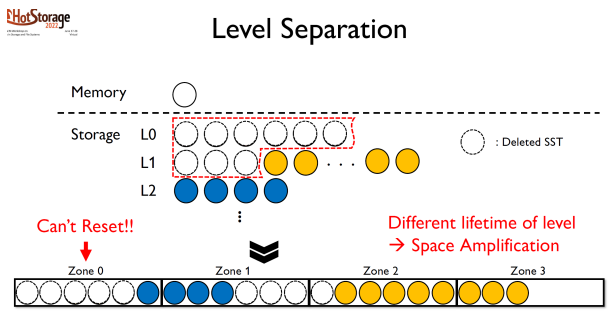


Fig. 8 不同level的sstable分配在同一zone时的情况

针对Long-Lived SST这种情况，在compaction时，必须轮询L+1层的所有的sstable，比如需要将图7(a)中的sst 5也要包含在compaction操作中。

针对Short-Lived SST这种情况，在第一次compaction时，将L+1层中与L层sstable无重叠key范围的部分分割为一个新的sstable放在一个新的空zone中，比如将图8中，第一次compaction(1,6,7)时，将SST 7中的非重叠key范围的部分分割出一个SST 11放在一个新zone中，然后第二次compaction(2,5,11)。

7 Conclusion

ZNS SSD作为近些年出现的新型存储设备，其相比传统的SSD有着更高的写吞吐量，更低的延迟和更大的容量。ZNS SSD将原本在设备端的GC移到主机端，同时其只能以zone为粒度进行GC，这导致如果没有好好设计compaction策略，其在进行GC时可能会产生更大的写放大问题。本文研究了近几年来有关优化ZNS compactio策略的几篇优秀文章，将其整理总结形成该文，希望本文能对有志于研究ZNS compaction策略的研究人员提供一些参考。

参考文献

[1] Bjørling M, Aghayev A, Holmberg H, et al. ZNS: Avoiding the Block Interface Tax for Flash-based SSDs[C]//2021 USENIX Annual Technical Conference (USENIX ATC 21). 2021: 689-703.

[2] Purandare D R, Wilcox P, Litz H, et al. Append is Near: Log-based Data Management on ZNS SSDs[C]//12th Annual Conference on Innovative Data Systems Research (CIDR'22). 2022.

[3] Stavrinos T, Berger D S, Katz-Bassett E, et al. Don't be a blockhead: zoned namespaces make work on conventional SSDs obsolete[C]//Proceedings of the Workshop on Hot Topics in Operating Systems. 2021: 144-151.

[4] Han K, Gwak H, Shin D, et al. ZNS+: Advanced zoned namespace interface for supporting in-storage zone compaction[C]//15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21). 2021: 147-162.

[5] Lee H R, Lee C G, Lee S, et al. Compaction-aware zone allocation for LSM based key-value store on ZNS SSDs[C]//Proceedings of the 14th ACM Workshop on Hot Topics in Storage and File Systems. 2022: 93-99.

[6] Jung J, Shin D. Lifetime-leveling LSM-tree compaction for ZNS SSD[C]//Proceedings of the 14th ACM Workshop on Hot Topics in Storage and File Systems. 2022: 100-105.