# 海量信息存储报告

论文题目：《SHRD: Improving Spatial Locality in Flash Storage Accesses by

Sequentializing in Host and Randomizng in Device》，选自FAST 2017会议，作者为Hyukjoong Kim, Dongkun Shin, Yun Ho Jeong, and Kyung Ho Kim。

## 1.摘要：

当前时代，随着闪存技术的不断发展，闪存的每bit价格不断降低，使得大容量的固态硬盘的应用越来越广泛，尤其是在企业应用中，常常作为大容量存储器来存储信息。但是固态硬盘的设计本质导致其有些固有的本质特征和缺陷。第一、固态硬盘的随机写性能相对而言比较差。但是大型企业的存储系统中必然存在大量的随机写操作，因此，这一点也是需要去攻克的。第二、在固态硬盘中的DRAM大小是和固态硬盘的大小成比例的。在固态硬盘中存在一个FTL部件，这个FTL部件为上层应用提供了一个通用接口，使得固态硬盘和传统机械硬盘在上层应用看来是一样的。这个FTL部件会在DRAM中维持一个地址映射表。目前，有很多方法来解决地址映射表的大小问题，但是会恶化固态硬盘的随机写效率。

结合上面两点要解决的问题，作者在本文中提出了一种新型的地址映射方法SHRD。这个方法会将一系列随机写操作在host上进行序列化，再在SSD中进行随机化。这个方法不同以前的方法，它在固态硬盘中进行随机化的时候只需要对映射表进行一系列修改，而不需要copy操作，因此会比以往的方法效果优秀。

## 2.背景和要解决的问题：

当前，固态硬盘飞速发展，其容量不断增加，价格也不断降低，应用范围也越来越广泛。首先，固态硬盘有很多区别于机械硬盘的特征。写前擦除的限制使得固态硬盘不能对block中的一页进行随时覆盖，而必须先将整个block中的有效数据转移到其他的空闲block上，在对这个block进行擦除，才能对这个页再进行写操作。因为这个特质，FTL在SSD中起到了很多关键的作用，FTL会先将将要覆盖原数据的新数据覆盖到新的空闲页中，将原数据状态变为无效，同时当原来块中的无效数据太多之后就会触发垃圾回收机制（GC），对这个块进行擦除操作，使之变得可以再利用。在FTL中，其会维持一个地址映射表，将上层应用发送而来的逻辑地址转换为物理地址。当前很多的映射表都是页级映射，即DRAM中维持一个从所有逻辑地址映射到物理地址的表，虽然这个方法简单方便，但是也存在很大的问题，当固态硬盘的容量有8TB的时候，就会需要8GB的DRAM去存放这张表，同时大DRAM的操作还会非常耗能，因此为了解决这个问题，有人提出了DFTL方法，这个方法的不同之处在于其维持了一个映射表的Cache，而整张映射表则存储在SSD中，按照访问的时间局限性原则和空间局限性原则，一个地址在短的时间内很有可能再次被访问，一个连续的地址空间也有可能被访问，因此Cache里只需要存储当前需要的地址映射表即可。如果上层访问请求有很大的时间局限性，那么DFTL的性能就和传统页级映射的性能不相上下。如果上层访问有很大的空间局限性，那么DFTL可以将连续的地址映射装入DRAM，也会提高命中率，使得访问效率变高。但事实证明，在大多数的情况下，很多访问的时间局限性和空间局限性特征不明显，尤其是随机读写，而DFTL方法不能很好的处理这类问题。（在这篇文章中，作者在基于DFTL的方法上进行了改进，主要来解决随机写的问题。）

经过作者的调查，随机写操作大概占了整个的70%，为了解决这类问题，有很多基于日志结构的文件系统被提了出来，比如说NILFS、F2FS，还有copy-on-write文件系统也被提出来，比如btrfs等。但是这些方法会触发大量的垃圾回收机制，同时也会存在文件系统和SSD的垃圾回收机制的重叠。不过在解决这类问题中，之前报告的文章《Application-managed flash》就是解决文件系统和SSD系统的垃圾回收机制重叠问题，该论文采用的思想是把SSD的FTL层中的大部分功能直接移植到上层应用中，使得重叠消失，但这个方法会改变上层应用的改变，可能会导致不兼容的情况产生。

另一个解决方法是将随机写操作请求转换为一个序列写操作。这类方法有ReSSD和LSDM等。这类方法在存储空间中划分出一块特殊的地方用来存储序列化的写操作的data。当这部分的数据占满空间后，则会把这些数据按照原逻辑地址，重新写入到存储空间的数据区。这类方法会导致重新写的操作，这会造成固态硬盘的操作拥挤，同时最后还是将随机序列写入了磁盘，没有从本质上去避免这个问题。

在本文中，作者提出的SHRD方法是基于DTFL-SSD的，在这个方法中，作者也将随机写操作进行了序列化，并写入了一个指定的空间，这个过程和前者相似，但是在当该特定空间满的时候，只需要修改地址映射表的相关项，而不需要重新copy，因此相比前者的方法，有更好的性能。

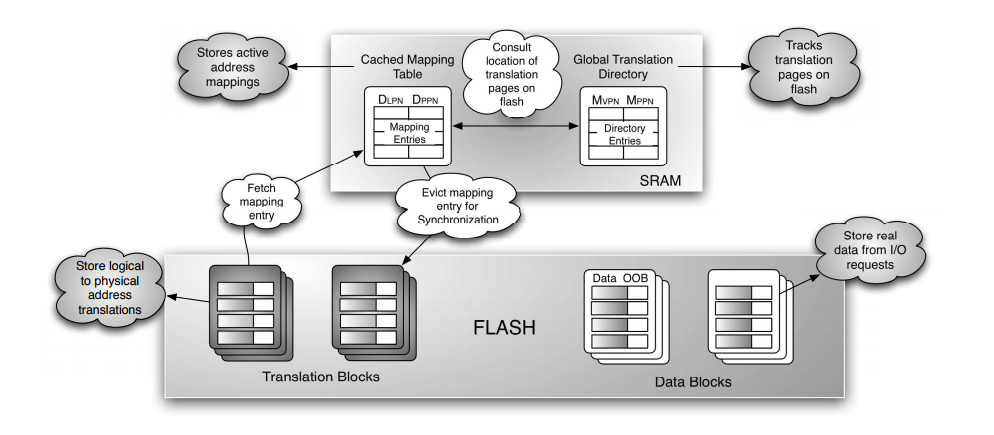
## 3.方法基础：DFTL

这个方法是基于DFTL的，因此在这个部分需要介绍一下DFTL。

首先介绍一下页级映射方法的相关操作。

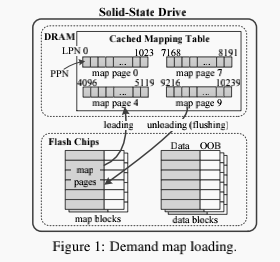
在页级映射方法中，页级映射FTL会在DRAM中维持一张L2P表（logical address 2 physical address）为了处理写请求，FTL分配一个active flash block，然后将写请求中的数据写入active flash block中，同时更新DRAM中的地址映射项，当这个active block写满了用户数据后，FTL就把DRAM中被修改的地址映射项写入闪存，成为map block，顾名思义，就是专门存放地址映射的数据块。若在DRAM中的地址映射数据还未写入map block时产生了断电情况的话，在SSD中的power-off recovery（POR）操作就会扫描所有的active blocks，并且重新建立表映射项，因为block中的每一page都会有一个OOB部分，可以存储很多其他信息，如ECC（校验）、LPN（逻辑块号）等，通过逻辑块号和active block对应的物理块号，再将信息写入map block即可。

下面介绍一下DFTL。



如上图所示，这个方法主要是为了减小mapping table的大小。在DRAM中维持一个CMT表，这个CMT表只有整个地址映射表的一部分，在GTD表中存放的是每个映射项所对应的map page在flash中的位置，当请求在CMT中miss时，就会利用GTB到Translation blocks中调出相对的映射项。

但是在本文中，和原DTFL有一些不一样，在原DFTL中，CMT中存放的是一条一条映射项，但在本文中，则存放的是一页一页的映射项，如下图所示：



在这幅图中，map blocks中必须以page为单位进行读，因此，读写请求没有在CMT中找到相应项后，会从map blocks中找到并以page为单位直接写入CMT，因此CMT中的page中的地址映射在逻辑地址上是连续的，因此当请求具有一定的空间局限性和时间局限性的时候，命中率会大大的提高。同时dirty map page也要整个写入map blocks中。那么这个问题也会暴露出来，当处理随机写的时候，CMT会连续将胀页写入map blocks，同时读出相应的地址映射页，这会造成多余的存储阻塞。同时这也会降低SSD的并行功能。当多个请求同时造成CMT中的miss的时候，而miss的map都在同一个芯片中的时候，那么这些请求将会等待这一个芯片一个一个将map读出，这显然会降低SSD的并行功能。

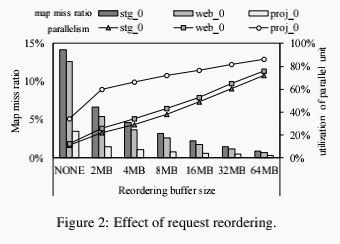
## 4.SHRD方法主要思想：

造成SSD随机写的功能相对较差的原因有三种：

1. mapping-table handling overhead：如果采用DFTL方法，那么随机请求将会造成大量的CMT miss，就会产生大量的map blocks的读写，即使SSD将所有的map导入到DRAM中，当处理随机写的时候，由于SSD写的特质，必然会产生大量的胀页，而这些胀页还要写入map blocks。
2. Request-handling overhead：大量小请求的产生会增加主机和SSD之间的阻塞。
3. Cost of GC：随机写会产生大量的需要回收的块，这些块都需要进行擦除。

在这篇文章中，主要是为了解决前两个问题。

作者做了一个实验，如下图所示：

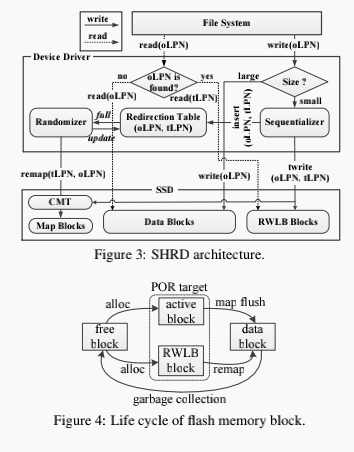


作者利用了一个请求缓存，在上层应用产生一系列请求的时候，先将这些请求存入缓存，当缓存满了之后，将这些请求按照请求的逻辑地址进行排序，然后再将这些请求发送给SSD，测得map miss ratio和utilization of parallel chips如上如所示，随着缓存大小的增大，map miss ratio不断减小，而utilization of parallel chips则不断上升。这个现象说明虽然工作负载在很短的一段时间内产生的请求的空间和时间的局限性很小，但是在很长一段时间内，产生的请求的空间和时间的局限性就会慢慢变大，这就解释了为什么上图中随着buffer的不断增大，而CMT的命中率不断减小，并行的利用率不断增大的现象。

作者从这个现象得到启发，但是作者并没有把数据存在缓存中，而是仅仅把映射项存入host memory中，数据则会直接写入SSD中。为了将请求的数据进行重新排序，SHRD将写请求的数据写入一个SSD中预定的空间，称为RWLB，这个过程就称为序列化。同时，这个RWLB是逻辑意义上的，并不是固定在SSD中的某个空间，只是有一个连续的逻辑页号而已。在写入这一系列序列化的数据时，将数据的原始的逻辑页号按照RWLB的连续页号重新分配，使其是按照顺序递增的页号，这样写入的时候，就会有很大的空间局限性，写入时就不会造成大量的map miss。同时在host 上维持一个原逻辑页号到重新分配的页号之间对应关系的表。当RWLB空间满了的时候，再利用host上的对应表将按照原逻辑页号排序后的对应表项发送给SSD，SSD再做相关处理。这里讲述的不是很具体，下面再来详细介绍。

## 5.SHRD Scheme:

SHRD包括一个装在host系统上的SHRD设备驱动和一个支持SHRD的SSD固件，具体的框架如下图所示：

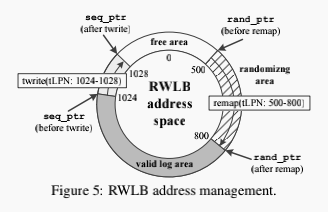


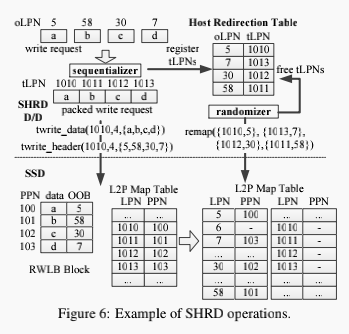
文件系统向下层发送一个write请求，这个请求中包含数据块的逻辑页号，大小，还有数据指针。SHRD在接收到这个请求后，根据其数据的大小来判断是否要将其进行序列化，当然在本文中采取的方法是数据块越小，越需要进行序列化，因此也会设置一个阈值来判断是否需要进行序列化。在序列化之后会进行一系列操作，并将数据写入RWLB blocks中，详细的操作在下面说明。

上图的循环图说明了在SHRD中每个block的生命周期，active block 和RWLB block都是data block和free block的中间状态，前面我们提到了在页级映射中的active block的作用，这里RWLB block在某种程度也是相同的，都可以在掉电，但是映射项还没有写入map block 的时候起恢复的时候。

### 5.1 Sequentializing in Host:

这一部分介绍Sequentalizing操作。如下图所示：





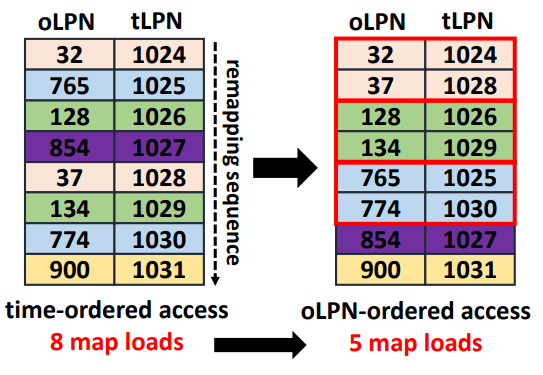
如上图所示，当SHRD接受到足够的需要序列化的请求后，就会对这些请求按照请求顺序重新分配一个LPN。在图中，假设来了对应原逻辑页号为5,58,30,7的请求，其对应的数据分别为a,b,c,d。Sequentiallizer觉得已经可以集合起来一起发送给SSD的时候，其会按照这四个请求到来的顺序，按照一定顺序分配一个逻辑页号，称为tLPN，这个分配是按照RWLB的逻辑页号来分配的。RWLB的逻辑空间大小是一定的，其逻辑页号也是一定，也是如图5所示的循环方式，按照图5所示，以一种递增的顺序，给5,58,30,7重新分配相对应的tLPN，在图6中，按顺序分配了1010,1011,1012,1013。当sequentializing部分做好前面的操作后，就会准备把数据写入RWLB blocks中，首先，其会发送一个twrite\_header命令，将tLPN作为逻辑地址，并且将原LPN即（oLPN）作为OOB中的内容发送，这里为什么分两个命令我也不是很清楚，这需要了解SATA的接口编程了。再发送twrite\_data命令，将数据a,b,c,d发送给SSD，这时候SSD是按照将数据写入RWLB block中，就如同前面提到的页级映射表把数据写入active block中。然后把相关修改写入CMT中，当某个RWLB block写满数据时，CMT会将所有修改的映射项写入map block，而map block中存储的LPN-PPN就是RWLB的tLNP和对应的物理页号，而不是原始逻辑页号，即（oLPN）。当CMT将映射项写入page map后sequentializer则会把序列中的原始逻辑页号和重新分配的逻辑页号的对应关系存入host系统的重定位表中，这个表也是为了在对读操作的时候进行重定位。

### 5.2 Read Redirection:

这个过程相对而言容易一点，当文件系统发送了一个读请求的时候，首先，SHRD的设备驱动会在host系统的重定位表中查找是否存在该原始逻辑页号的映射，若不存在，则直接用该原始逻辑页号作为查找地址送入SSD。若存在，则应该把重定位表中的tLPN用以替换oLPN，然后将其送入SSD中进行查找。在这里，我们可能会怀疑这里增加了查找过程，会增加操作时间，在文章中，作者采用红黑树的形式存储这些信息，其查找时间复杂度大概为O(log n)，因此这个的影响不大。

### 5.3Randomizing in Device

当RWLB地址空间被消耗的快没了的时候，这个时候就需要进行Randomizing操作，这个操作，实际上就是把RWLB转换为data blocks。如图6所示，这个时候，SHRD设备驱动会向SSD发送一个remap的指令，而这个remap中则包含需要进行随机化的RWLB的存储空间，如图5所示，整个RWLB的存储空间是固定的，也是一个循环的过程，rand\_ptr则指向需要序列化的的逻辑空间的起始位置，规定一个固定的长度，从这个起始位置开始的后面的固定长度的空间都是需要进行随机化的。首先在host系统的重定位表上找到tLPN-to-oLPN的映射项集合，并且按照oLPN的顺序进行排序，然后再用remap指令发送给SSD。在修改CMT的时候，由于所有的数据是按照oLPN进行排序的，因此大大的提高了其空间局限性，下面有一个详细的例子：



在这个例子中，如果不进行排序，那么CMT就会产生8个miss，同时从map blocks中导入8个map，而在按照oLPN进行排序之后，只产生了5个miss，也只需要从map block中导入5个map，从而提高了性能。

在SSD接受到remap命令之后，就会修改CMT中的相应项，同时，胀页也会重新写入存储在SSD中的map blocks，如图6所示，在L2P表中原来是tLNP-PPN的关系，在进行remap之后就变成了oLNP-PPN关系，如果整个RWLB block中所有的页都进行了这样的转换，那么这个RWLB block就会变为一个普通的data block，那么垃圾回收机制就可以对这个块进行操作。同时，在这里也不需要任何类似前面所提到的ReSSD和LSDM等方法的copy操作，而只需要对L2P进行相关的修改，也提高了相应的效率。

### 5.4Power-off-recovery and Garbage Collection

在这部分，将会讲到如何进行掉电处理及垃圾回收。

掉电处理：

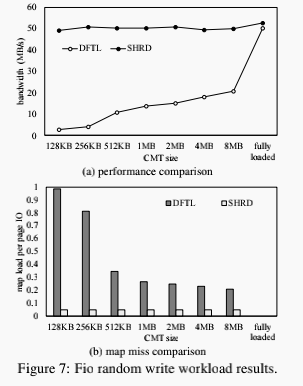
若host系统突然掉电，那么必然损失了系统中的重定位表，即损失了原逻辑页号和tLPN的对应关系。这时POR会扫描所有的RLWB blocks，因为RLWB blocks中的每个页的OOB中存储着其对应的原逻辑页号，那么原逻辑页号和物理页号的对应关系就能重新建立起来，通过查找L2P表，那么oLPN和tLPN的对应关系就能重新建立起来，但是在这里不需要重建重定位表，直接对RLWB blocks进行随机化操作就行了。

垃圾回收：

当RWLB blocks变为普通的存储数据的data blocks之后，就有可能被选择为回收块，垃圾回收需要将块中所有剩下的有效数据全部都copy到新的块中，而这就会去修改CMT，但是由于原RWLB blocks中存储的都是随机写的内容，那么就会造成在CMT中的很多miss，为了解决这个问题，作者采用了 map logging technique技术。这个时候，当CMT产生miss的时候，不需要将L2P中的整个map page导入到CMT，而是将新的映射项写入DRAM中的一个map log table，这个也只需要占用一点点DRAM的空间。当map log table满了的时候，将table中所有的项按照LPN进行排序，然后集体写入CMT，这个时候，由于是拍完序列的，那么就增加了其空间局限性，就会减少CMT的miss几率。

## 实验结果及阅读心得

作者做了一系列实验。作者利用不同的workload，这些我就不详细介绍了。在下图7a中，作者用SHRD和DFTL的随机写功能进行了对比，x轴是CMT表的大小，显然可见SHRD在不同CMT大小的对比中，性能比较稳定，且一直比DFTL好。在图7b中，比较了不同CMT大小下，不同方法的map miss ratio。显然易见，SHRD的miss ratio一直都比DFTL小，性能更优越。



阅读心得：

在这篇文章中，我觉得其重要思想就是利用了随机写在短时间内不具有很强的空间和时间上的局限性，但是当时间段很长的时候，其时间和空间的局限性就能体现出来，所以采用了缓存的策略，集合一段时间内的随机读写，再利用逻辑页号进行排序，突出其时间和空间的局限性，从而达到减少CMT中频繁读写map page的操作，同时也能解决大量小型写操作造成的拥挤问题。

但是这篇文章主要集中解决随机写产生的CMT的miss问题，和大量小型写操作造成的拥挤，而没有解决随机写所产生的垃圾回收的问题，这也是这篇文章的一个缺陷之处。