《海量存储技术》课程读书报告
小尾闪存：一种可以几乎完美消除 NAND 固态硬盘中由于垃圾回收导致的尾部延迟的方法

夏海峰　　学号
武汉大学计算机学院　　*2017202110067*

摘要

固态硬盘作为一种近几年在消费者市场十分火爆的商品，其优秀的性质吸引了众多关注与先关研究。在这里，我仅仅以一个简单的方式，先简单介绍了固态硬盘的发展历史和先关的一些技术内容，然后结合一篇FAST' 2017[8] 会议论文及其相关论文。文章提出来了一种全新的，被称为 TTFlash 的固态设备。这一全新的的固态硬盘通过绕过四种全新的方法（页面阻断垃圾回收、垃圾回收轮转、允许垃圾回收读取、允许垃圾回收写入）来降低固态硬盘的尾部延迟。这些都是基于固态硬盘内部三个部件的进步：主控芯片、基于奇偶校验的RAIN和电容式背板RAM。但依赖于页面内的使用回拷操作。这一全新的设备在实际会使用是表现出几乎不出现垃圾回收机制。

1　引言

近些年来，随着人们对存储的要求的不断提高，特别是随机读写等对 PC 响应速度起着至关重要作用的参数，使得 SSD （Solid State Drive）收到了广泛的关注与研究。SSD 市场和 SSD 的相关技术取得了长足的发展。固态硬盘（Solid State Drives），用固态电子存储芯片阵列而制成的硬盘，由控制单元和存储单元（FLASH芯片、DRAM芯片）组成。固态硬盘在接口的规范和定义、功能及使用方法上与普通硬盘的完全相同，在产品外形和尺寸上也完全与普通硬盘一致。被广泛应用于军事、车载、工控、视频监控、网络监控、网络终端、电力、医疗、航空、导航设备等领域。 SSD 作为一种重要的存储介质，相较于普通的 HDD 磁盘，SSD 具有读写速度快的优势。特别是在系统启动等对随机读写操作要求较高的场景， SSD 相较于普通机械硬盘的优势就充分体现出来。

1.1　SSD 的发展历史

SSD 的最早起源最早可以追溯到上世界五十年代，当时有两种相近的技术： magnetic core memory 和 charged capacitor read-only storage (CCROS)。这些辅助单元出现早真空电子管计算机时代。直到上世纪七十年代至八十年代之间，SSD 作为半导体存储介质被用在 IBM 的超级计算机上，但是由于高昂的造价与维护费用，这项技术在当时并未得到广泛的应用。世界上第一款真正意义上的固态硬盘出现于1989年，不过由于其价格过于高昂因此在当时只限应用于非常特别的市场比如医疗、工作以及军用市场。实际上虽然当时其1M大小的闪存换算下来的价格已经达到了3500美元，但是其性能却要远低于当时的普通硬盘产品，不过

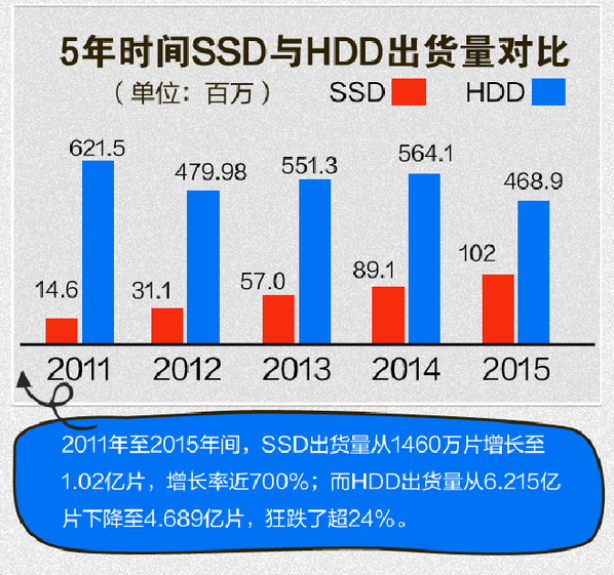

图 1: SSD 和 HDD 近五年出货量变化对比图示

凭借其独有的特性却使得闪存硬盘在军用、航空以及医疗领域获得了长足的发展。自从07年七月IBM在其刀片式服务器上部署 SanDisk SSD，固态硬盘再一次走进人们的视线。

早期的固态硬盘读写速度低，比当时普通的机械硬盘还慢。但是固态硬盘经久耐用、防震抗摔。因为全部采用了闪存芯片，所以SSD固态存储器内部不存在任何机械部件，这样即使在高速移动甚至伴随翻转倾斜的情况下也不会影响到正常使用，即是设备发生意外掉落或与硬物碰撞时能够将数据丢失的可能性降到最小。除此之外，固态存储器工作时非常安静，没有任何噪音产生。得益于无机械部件及闪存芯片发热量小、散热快等特点，SSD固态存储器因为没有机械马达和风扇，工作时没有噪音。

1.2 SSD 的市场发展及前景

固态硬盘真正进入消费者市场还得到2008年，更早期的 SSD 都没能表现出足够的市场竞争力，与当时同时代的产品，比如机械硬盘等。但是近些年来，技术的革新带来了制程的不断转换，固态硬盘从SLC到MLC再到现在TLC，每次变革都会推动SSD市场向前迈进一大步。特别是 TLC 颗粒，TLC颗粒技术使得SSD容价比不断降低，直接推动了SSD市场的发展，用户可以买到相对超值的产品。

如图1所示 [1]，固态硬盘近几年的市场需求还是的增长是十分迅速的。

1.3 SSD的构成

SSD主要由主控制器，存储单元，缓存（可选），以及跟HOST接口（诸如SATA，SAS, PCIe等）组成。主控制器:每个 SSD 都有一个控制器(controller)将存储单元连接到电脑，主控器可以通过若干个通道（channel）并行操作多块FLASH颗粒，类似RAID0，大大提高底层的带宽。控制器是一个执行固件(firmware)代码的嵌入式处理器。主要实现如下功能：

1. 错误检查和纠正(ECC)

2. 磨损平衡(Wear leveling)

3. 坏块映射(Bad block mapping)

4. 缓存控制

5. 垃圾回收(垃圾回收, Garbage Collection)

存储单元: 虽然有某些厂商推出了基于更高速的 DRAM 内存的产品，但 NAND 闪存依然是最常见的，占据着绝对主导地位。在存储颗粒方面，低端产品一般采用 MLC(multi-level cell) 甚至 TLC(Triple Level Cell) 闪存，其特点是容量大、速度慢、可靠性低、存取次数低、价格也低。高端产品一般采用 SLC(single-level cell) 闪存，其特点是技术成熟、容量小、速度快、可靠性高、存取次数高、价格也高。

1. SLC(Single Level Cell), 单层单元 :就是在每个存储单元里存储 1bit 的数据，存储的数据是0还是1是基于电压阀值的判定，对于 NAND Flash 的写入(编程)，就是控制 Control Gate 去充电(对 Control Gate 加压)，使得浮置栅极存储的电荷够多，超过4V，存储单元就表示 0(已编程)，如果没有充电或者电压阀值低于4V，就表示 1(已擦除)。

2. MLC(Multi-Level Cell)，多层单元: 就是每个存储单元里存储 2bit 的数据，存储的数据是”00”,”01”,”10”,”11”也是基于电压阀值的判定，当充入的电荷不足3.5V时，就代表”11”，当充入的电荷在3.5V和4.0V之间，则代表”10”，当充入的电荷在4V和5.5V之间，则表示”01”，当充入的电荷在5.5V以上，则表示”00”。同时由前面的图可以看到，MLC 相比 SLC 虽然使用相同的电压值，但是电压之间的阀值被分成了4份，可以想象这样就直接影响了性能和稳定性。

3. TLC(Triple Level Cell)，三层单元: 就更加复杂，因为每个存储单元里存储 3bit 的数据，所以它的电压阈值的分界点就更细致，导致的结果也就每个存储单元的可靠性也更低。

2 SSD 相关的重要技术

2.1 磨损平衡

磨损平衡(Wear leveling)，简单说来就确保闪存的每个块被写入的次数尽可能相等的一种机制。通常情况下，在 NAND 块里的数据更新频度是不同的：有些会经常更新，有些则不常更新。很明显，那些经常更新的数据所占用的块会被快速的磨损掉，而不常更新的数据占用的块磨损就小得多。为了解决这个问题，需要让每个块的编程(擦写)次数尽可能保持一致：这就是需要对每个页的读取/编程操作进行监测，在最乐观的情况下，这个技术会让全盘的颗粒物理磨损程度相同并同时报废。

磨损平衡算法分静态和动态两种。动态磨损算法是基本的磨损算法：只有用户在使用中更新的文件占用的物理页地址被磨损平衡了。而静态磨损算法是更高级的磨损算法：在动态磨损算法的基础上，增加了对于那些不常更新的文件占用的物理地址进行磨损平衡，这才算是真正的全盘磨损平衡。简单点说来，动态算法就是每次都挑最年轻的 NAND 块来用，老的 NAND 块尽量不用。静态算法的主要思路，就是把长期没有修改的老数据从一个新的 NAND 块里面搬出来，重新找个最老的 NAND 块放着，这样年轻的 NAND 块就能再度进入经常使用区。尽管磨损均衡的目的是避免数据重复在某个空间写入，以保证各个存储区域内磨损程度基本一致，从而达到延长固态硬盘的目的。但是，它对固态硬盘的性能有不利影响。

2.2 垃圾回收

垃圾回收(GC, Garbagecollection) 当整个SSD写满后，从用户角度来看，如果想写入新的数据，则必须删除一些数据，然后腾出空间再写。用户在删除和写入数据的过程中，会导致一些Block里面的数据变无效或

者变老。Block中的数据变老或者无效，是指没有任何映射关系指向它们，用户不会访问到这些FLASH空间，它们被新的映射关系所取代。比如有一个Host Page A，开始它存储在FLASH空间的X,映射关系为A->X。后来，HOST重写了该Host Page，由于FLASH不能覆盖写，SSD内部必须寻找一个没有写过的位置写入新的数据，假设为Y，这个时候新的映射关系建立：$A->Y$，之前的映射关系解除，位置X上的数据变老失效，我们把这些数据叫垃圾数据。随着HOST的持续写入，FLASH存储空间慢慢变小，直到耗尽。如果不及时清除这些垃圾数据，HOST就无法写入。SSD内部都有垃圾回收机制，它的基本原理是把几个Block中的有效数据（非垃圾数据）集中搬到一个新的Block上面去，然后再把这几个Block擦除掉，这样就产生新的可用Block了.

另一方面，由前面的磨损平衡机制知道，磨损平衡的执行需要有“空白块”来写入更新后的数据。当可以直接写入数据的“备用空白块”数量低于一个阀值后，SSD主控制器就会把那些包含无效数据的块里的所有有效数据合并起来写到新的“空白块”中，然后擦除这个块以增加“备用空白块”的数量。

目前， SSD 主要有三种垃圾回收策略：

1. 闲置垃圾回收：很明显在进行垃圾回收时候会消耗大量的主控处理能力和带宽造成处理用户请求的性能下降，SSD 主控制器可以设置在系统闲置时候做“预先”垃圾回收(提前做垃圾回收操作)，保证一定数量的”备用空白块”，让 SSD 在运行时候能够保持较高的性能。闲置垃圾回收的缺点是会增加额外的”写入放大”，因为你刚刚垃圾回收的”有效数据”，也许马上就会被更新后的数据替代而变成”无效数据”，这样就造成之前的垃圾回收做无用功了。

2. 被动垃圾回收：每个 SSD 都支持的技术，但是对主控制器的性能提出了很高的要求，适合在服务器里用到，SandForce 的主控就属这类。在垃圾回收操作消耗带宽和处理能力的同时处理用户操作数据，如果没有足够强劲的主控制器性能则会造成明显的速度下降。这就是为啥很多 SSD 在全盘写满一次后会出现性能下降的道理，因为要想继续写入数据就必须要边垃圾回收边做写入。

3. 手动垃圾回收：用户自己手动选择合适的时机运行垃圾回收软件，执行垃圾回收操作。

可以想象，如果系统经常进行垃圾回收处理，频繁的将一些区块进行擦除操作，那么 SSD 的寿命反而也会进一步下降。由此把握这个垃圾回收的频繁程度，同时确保 SSD 中的闪存芯片拥有更高的使用寿命，这确实需要找到一个完美的平衡点。所以，垃圾回收是影响 SSD 性能的重要机制之一。

3 垃圾回收机制导致的尾部延迟

文章的主要解决的问题是，试图通过改进现有的固态硬盘的垃圾回收机制来消除延迟造成的性能下降。这篇文章之前的文章[2] [6]都是试图通过减少垃圾回收的次数来提高性能。对于现行的普通的固态硬盘，其在进行垃圾回收操作时，这一过程往往是阻塞式的，即固态硬盘在进行垃圾回收操作时，时无法进行其他的 I/O 操作的。垃圾回收机制的这一阻塞式特性，是垃圾回收对固态硬盘的性能会产生重大影响的根本原因。而这篇文章则打算改变垃圾回收这一过程的阻塞式的性质，借此来提高固态硬盘的性能。这种全新的固态硬盘技术被称为 TTFLash (Tiny Tail Flash)。

在这里，我们提出了两个显示垃圾回收串联影响的实验，这激发了我们的工作。每个实验都是在2014年末的128GB三星SM951上运行的，它可以承受70“KWPS”（70K的4KB随机写入/秒）。

在图二中,我们运行了一个执行16 KB随机读取的前台线程，同时还有后台线程，在三个实验中，以1,2.5和5 KWPS（远低于最大值70 KWPS）注入4 KB随机写入噪声。我们测量李，每读取16 KB前景的延迟。图1a描绘了 L_i 的 CDF，清楚地表明更频繁的垃圾回收（来自更强垃圾回收诱发的，而不是排队延迟，我们进行了相同的实验，但是现在使用了随机读取噪声（1,2,5和5 KRPS），读出的噪声结果被绘制成

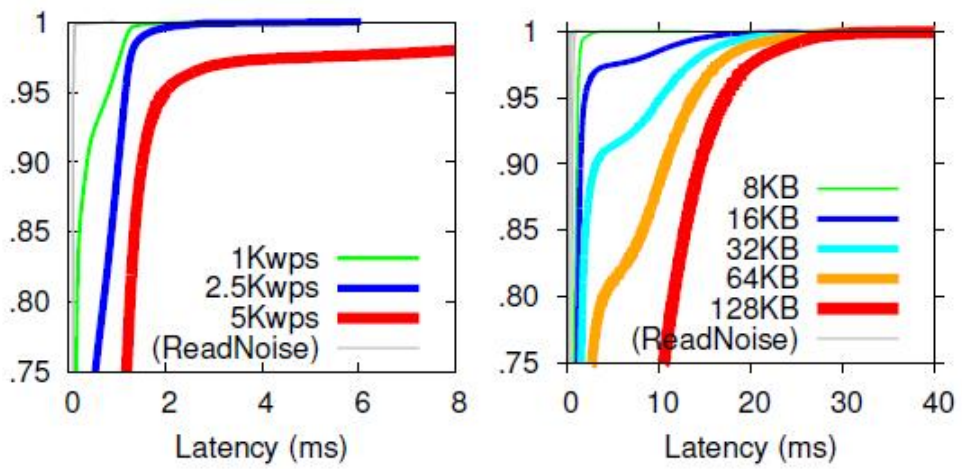

图 2: 垃圾回收导致的尾部延迟

三条重叠的细线，标记为“ReadNoise，这代表了一个完美的无垃圾回收情景，如图2所示，噪音为5KWPS，读取操作分别比无垃圾回收情景分别慢90倍，95倍和99倍，在有垃圾回收是慢了分别15倍，19倍和96倍。图2b中，我们保留了5-KWPS的噪声，现在改变了前景随机读取的I / O大小（5个实验中的8,16,32,64和128 KB）。假设，读取的2倍应该只消耗2倍的延迟，但是这个数字表明垃圾回收在更大的读取中会产生更多的尾部延迟，例如，在第85个百分点处，64-KB读取比32-KB读取要慢4倍，问题的核心是：如果一个大读取的单个页面被垃圾回收阻塞，则整个读取不能完成；随着读取大小的增加，其中一个页面被垃圾回收阻塞的概率也增加。与具有5-KRPS噪声（标记为“ReadNoise”的五条细灰线）相同的实验相比，该模式更明显。因为闪存读取延迟通常比写入延迟快20倍，为了更公平的实验，所以我们还读取了20倍更强烈的读取噪声，以及读取噪声20倍的读取噪声。发现结果是相似的。

4 TTFlsh 的设计

现在介绍 TTFlash 的设计。 TTFalsh 是全新的固态硬盘结构，可以保证设备具有和不发生垃圾回收的固态硬盘具有相近的性能表现。使用如下四个关键的策略来达从所有层面上消除垃圾回收的效果：

1. 设计一个无阻塞的控制器和通道协议，将垃圾回收上的任何资源阻塞推给受影响的页面。我们称这种细粒度的架构阻塞垃圾回收。
2. 利用基于 RAIN 奇偶校验的冗余，并将其与垃圾回收信息结合起来，主动重新生成在页面级垃圾回收阻塞的读取。这一过程称为允许垃圾回收读取。
3. 以轮转的方式去安排垃圾回收的时间，确保每个页面上最多只有一个垃圾回收在执行。一个奇偶校验只能“去尾”。称这一机制为垃圾回收轮转。
4. 使用电容器支持的写入缓冲器可以快速持久地完成写操作，从而使其能够以和垃圾回收并存的方式，在垃圾回收完成以后被逐出闪存页面。这一过程被称为垃圾回收缓冲。

4.1 页面阻塞垃圾回收

控制器和通道阻塞垃圾回收由于其硬件实现简单而经常被采用；垃圾回收本质上是一个回拷命令

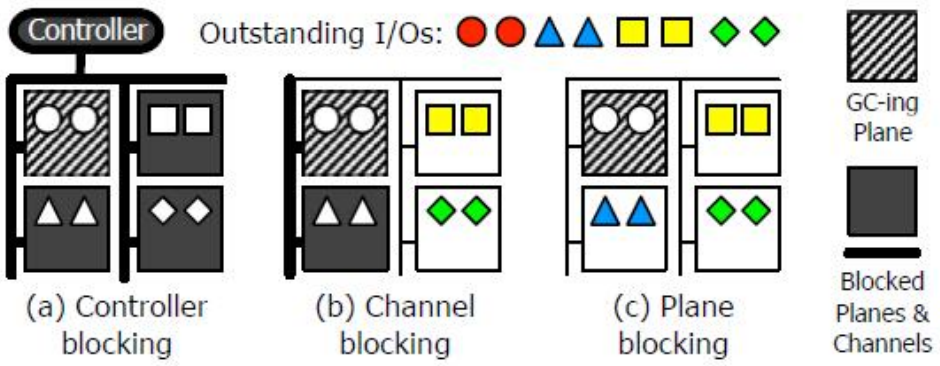

图 3: 不同级别的垃圾回收阻塞

的for循环。然而，这种简单性导致严重的尾部延迟，因为独立的页面被不必要地阻挡。对于较大的I / O，通道阻塞并不比控制器阻塞垃圾回收好；由于每个大型I / O通常都是跨多个通道分条的，因此一个垃圾回收繁忙通道仍会阻塞整个页面的 I / O，否定了SSD并行性的好处。此外，随着SSD容量的增加，在同一个通道中会有更多的页面被阻塞。更糟糕的是，垃圾回收时期可能会很长。复制64个有效页面（25％有效）的垃圾回收将导致54毫秒（64×840μs）被阻塞的通道，这可能使数百个其他I / O不可用。原本只需要少于100μs的快速的读取操作现延迟了增长了几个数量级（s）[1] 。

为了减少这种不必要的阻塞，我们引入了页面阻塞垃圾回收。如图3c所示。这里，唯一被垃圾回收阻塞的I / O是那些处于垃圾回收的页面（用 ○记）。所有未处于垃圾回收状态的页面都可以进行 I/O 操作。包括垃圾回收页面的同一通道中的I / O。作为附注，页面封闭垃圾回收可以互换地定义为芯片封闭垃圾回收；在本文中，我们使用1个页面/芯片来简化演示。为了实现这个概念，控制器必须执行细粒度的I / O管理。为了说明，让我们考虑四个垃圾回收步骤。在TTFLASH中，控制器CPU /线程发送闪存到寄存器的读/写命令（步骤1和3）后，不会空闲等待（分别为40μs和800μs），直到下一步可执行为止。（请注意，在一个通用的实现中，由于简单的for循环和需要访问通道来检查页面，控制器是空闲的。对于页面阻塞的垃圾回收，在步骤1和3（发送读取/写入命令）之后，控制器创建一个将来的事件，标记完成时间。控制器可以可靠地预测页面内读/写命令将完成多长时间（例如，分别平均40和800μs）。总而言之，使用页面封锁垃圾回收，TTFLASH与其他未完成的I / O的页面内复制和信道使用重叠。如图3c所示，在页面内复制（分条/ 垃圾回收页面）期间，控制器可以继续为相应通道（▲ I / O）中的其他非垃圾回收页面提供I / O服务。

页面阻塞垃圾回收可能会释放数百个以前阻塞的I / O。然而，还有一个未解决的垃圾回收阻塞问题和一个新的分支。未解决的垃圾回收阻塞问题是，在垃圾回收完成之前，垃圾回收页面的I / O（图3c中的＃个标签）仍然被阻塞；换句话说，只有页面屏蔽，我们不能完全去除垃圾回收阻塞。页面阻塞的新分支是垃圾回收操作的一个潜在的延长。当垃圾回收页面准备接受另一个命令时（步骤2和4结束），由于重叠，控制器/通道可能仍处于为其他I / O服务的中间位置。例如，在垃圾回收读取完成（步骤1）后，控制器不能正好开始垃圾回收写入（步骤3），同样，下一个垃圾回收读取（步骤1）在上次垃圾回收写入后800 μs 才能开始。如果垃圾回收时间延长，垃圾回收页面的I / O将被阻塞更长时间。幸运的是，上面的两个问题可以用RAIN和垃圾回收容忍的读取来掩盖。

4.2 RAIN

为了防止I / O阻塞到垃圾回收页面，我们利用了最近流行的数据完整性标准 RAIN 。 RAIN 在 SSD 中

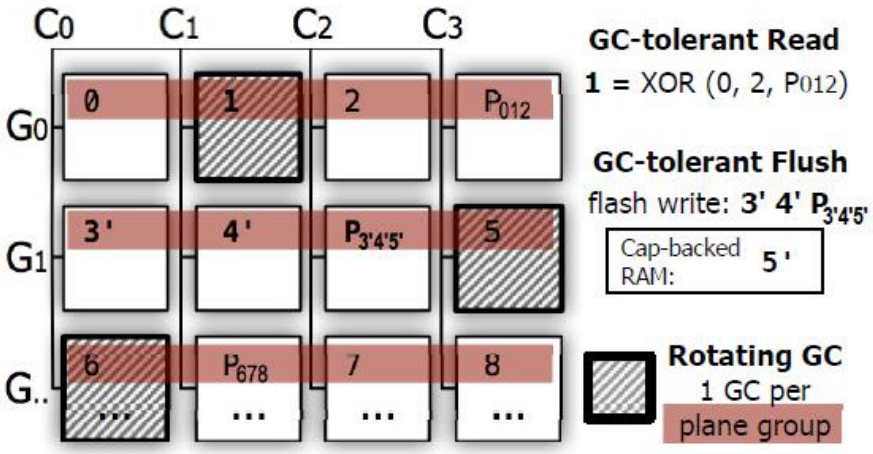

图 4: TTFlash 的结构

引入了奇偶校验页的概念。就像基于磁盘的RAID的发展一样，许多 RAIN 布局已经被引入[5] [7] ，但主要集中在数据保护，写入优化和磨损均衡上。相反，我们设计了一个RAIN布局，也是针对尾巴宽容的。本节简要介绍了我们的基本RAIN布局，足以了解它如何实现垃圾回收容错读取；我们稍后会讨论更高级的布局，以及磨损均衡问题。

图4 显示了文章中的 RAIN 的布局。为了描述的简单，仅使用了四个通道（C_1 - C_3）。

关于FTL设计（LPN-PPN映射），有两种选择：动态还是静态。动态映射（LPN可映射到任何PPN）通常用于加速写入（灵活目标）。但是，在现代SSD中，写入延迟问题被电容器支持的RAM（内存）所吸收。因此，写入分散在多个渠道。其次，个别页面独立时动态映射效果良好；但是，RAIN页面是条带化的。通过动态映射，条带中的页面可以放置在一个通道的后面，这样就不能充分利用通道的并行性。

鉴于以上原因，我们创建一个页面级的混合静态动态映射。静态分配策略是：（a）LPN静态地映射到一个平面（例如，图4中的LPN 0到平面G_0C_0），（b）N-1个连续的LPN和它们的奇偶性形成条带（例如，$012P_{012}$），（c）将条纹页映射到一个平面组内的通道上的平面（例如，G_0 中的$012P_{012}$）。稍后，我们将展示所有这些对于支持垃圾回收容忍读取（和绕过垃圾回收都是至关重要的。

动态分配策略是：在每个平面/芯片内部，一个LPN可以动态映射到任何PPN（数十万个选择）。覆盖同一个LPN将被重定向到同一平面上的空闲页面（例如，覆盖LPN0可被引导到G0C0平面内的任何PPN）。为了防止奇偶校验通道瓶颈（类似于RAID-4奇偶校验磁盘瓶颈），我们采用具有少许自定义布局的RAID-5。首先，我们将这组通道当作RAID-5组处理。例如，在图4中，P_{012}和P_{345}处于不同的通道，对角线方式。其次，由于SSD平面形成了具有磨损问题的二维布局（G_iC_j）（与磁盘的“扁平”LPN不同），我们需要确保热平价页面均匀分布。

4.3 允许垃圾回收的读取操作

我们可以轻松支持允许垃圾回收的读取操作（GC-Tolerant Read，GTR）。对于全条带读取（使用N-1个通道），GTR很简单：如果由于正在进行的垃圾回收而无法读取页面，则通过从另一个平面读取奇偶校验来快速重新生成页面内容。在图4中，给出LPN 0-2的全条带读取，并且如果LPN 1暂时不可用，则通过读取奇偶性（P012）来快速再生内容。因此，全带读取不受正在进行的垃圾回收的影响。产生的等待时间比等待垃圾回收完成快一个数量级。奇偶校验计算开销对于N8只需要少于3 µ s，附加奇偶校验只需要最少40 +100 µ s

（读+传输延迟），不会引入太多的争用。

对于部分条带读取（其中$R < N - 1$的R个页面），允许垃圾回收的写操作将总共产生N-R个额外读取;最坏的情况是当R = 1时。这些N-R额外的并行读取将增加每个N-R通道的争用，这可能需要为其他未决的I / O服务。因此，如果TGCtoComplete¿ B×（40 + 100）μs（其中B是N-R额外读取中繁忙信道的数量（对于非繁忙信道额外读取是空闲的）），我们只执行额外的读取。根据我们的经验，这个简单的策略能有效和公平地削减垃圾回收尾部延迟，而不引起重大的争论。在另一端，总是执行额外读取的“贪婪”方法导致高频道竞争。我们强调的是，不像尾部容忍的投机执行，通常被定义为一个可能实际上不需要的优化任务，允许垃圾回收的读取操作是肯定的，而不是投机性的;控制器确切知道何时何地垃圾回收正在发生以及完成多久。GTR是有效的，但有一个局限性：当一个页面组中的多个页面同时执行垃圾回收时，它不工作，这是我们用绕过垃圾回收处理的。

4.4 允许垃圾回收的缓冲

到目前为止，我们只处理读取尾部。写入更复杂（例如，由于写入随机性，读取和修改奇偶校验更新以及需要耐久性）。为了处理写入的复杂性，我们利用闪存行业大量使用电容支持的RAM作为一个持久的写缓冲区（简称“cap-backed RAM”）。为防止数据丢失，根据电源掉电后的电容器放电周期调整RAM大小;大小可以从几十到几百MB，这种设备被称为“超级电容”。

我们采用容量有限的RAM来快速吸收所有的写入。当缓冲区占用率高于80％时，将会运行后台缓冲以清除某些页面。当缓冲区已满（例如，由于大量写入），前台缓冲将运行，这将阻塞传入的写入，直到释放一些空间。这里要解决的问题是，当驱逐的页面必须发送到垃圾回收平面时，前景缓冲可能会导致写入尾巴。为了解决这个问题，我们引入了GC-tolerant flush（GTF），它可以确保页面驱逐不受垃圾回收阻塞的影响，这在旋转垃圾回收的情况下是可能的。例如，在图4中，属于3’，4’和P3’4’5’的页面可以从RAM中逐出到闪存，而5页’驱逐被延迟直到目的地平面完成垃圾回收。凭借经过验证的旋转垃圾回收，GTF可以每隔N个页面逐一排除N-1页，而不会被阻塞。因此，尚未被缓冲的页面所需的最小RAM空间很小。

对于部分条带写入，我们执行通常的RAID读取 - 修改 - 写入驱逐，但仍然没有被垃圾回收阻塞。让我们设想一下图4中第7’和8’页更新的最坏情况。新的奇偶校验应该是$P_{67'8'}$，这需要先读第6页。尽管页面6不可访问，但是可以通过读取旧页面P_{678},7和8来重新生成，之后页面7’，8’和$P_{67'8'}$可以被驱逐。我们注意到，如此昂贵的奇偶校验更新是非常罕见的，因为我们优先将全条带脏页面逐出到非垃圾回收ing平面，然后用GTF将全页面大部分排除到非垃圾回收ing平面。接下来，我们将部分条带脏页面驱逐到非垃圾回收ing平面，最后将部分条带页面驱逐到GTF的大部分非垃圾回收ing平面。与其他集中于减少写入放大的驱逐算法[35]相比，我们的方法增加了垃圾回收尾巴容忍。

5 实现

1. ttFlash-Sim (SSDSim): 为了便于器件级的准确的延迟分析，我们首先在SSDSim [4] 中实现了TTFLASH，这是一个最近流行的仿真器，其精度已经在真实的硬件平台上得到验证。我们使用SSDSim由于其纯净的设计。我们通过将2482 LOC添加到SSDSim来实现所有TTFLASH功能。这涉及对普通版本（6844 LOC）进行重大修改（+ 36％）。（523 LOC），RAIN（582），旋转垃圾回收（254），垃圾回收耐受读取（493）和写入（630行）。

2. ttFlash-Emu (“VSSIM++”): 为了运行Linux内核和文件系统基准测试，我们还将TTFLASH移植到基于QEMU / KVM的平台“VSSIM”，这个平台“有助于实现SSD固件算法” [9]。 VSSIM模拟RAM磁

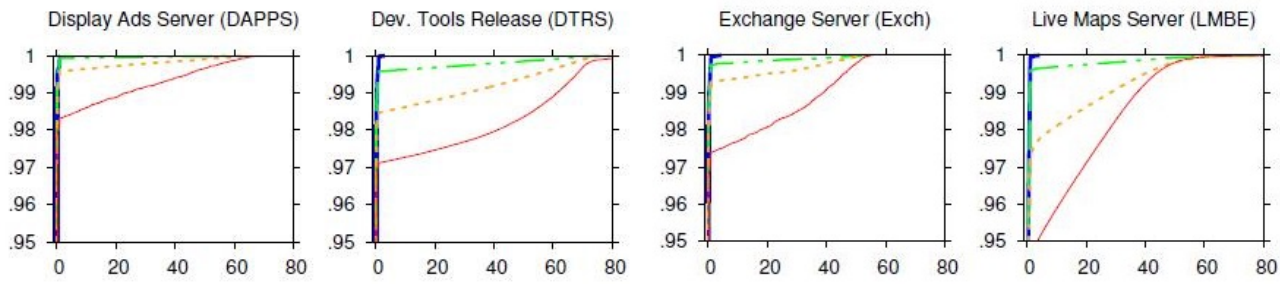

图 5: 尾部延迟

盘上的NAND闪存延迟。不幸的是，VSSIM的实现基于5年前的QEMU-v0.11 IDE接口，仅提供1 0K IOPS。此外，由于VSSIM是单线程设计，因此它基本上模仿了控制器阻塞SSD（垃圾回收下1K IOPS）。

这些限制导致我们做出重大改变。首先，我们将VSSIM的单线程逻辑迁移到QEMU AIO模块中的多线程设计，这使我们能够实现通道阻塞。其次，我们将这个新设计迁移到最近的QEMU版本（v2.6），并将其连接到PCIe / NVMe接口。我们的修改，我们称之为“VSSIM ++”，可以承受50K的IOPS。最后，我们将TTFLASH特性移植到我们称之为ttFlash-Emu的VSSIM ++中，总共变化为869个LOC。

6 评价

6.1 主要结论

我们现在进行评估，显示TTFLASH显着消除垃圾回收阻塞（第6.1节），比最先进的先发性垃圾回收（第6.2节）和其他垃圾回收优化技术（第6.3节）提供更稳定的延迟，并且不显着增加超过RAIN开销的P / E周期。

我们评估两个实现：ttFlash-Sim（在SSDSim上）和ttFlash-Emu（在VSSIM ++上），如第5节中所述。对于ttFlash-Sim评估，我们使用6个来自Microsoft Windows Server的实际块级别跟踪图5的数字标题。默认情况下，对于每个跟踪，我们选择了最频繁的时间段（除了6分钟的TPCC跟踪）。我们使用 filebench 对 ttFlash-Emu评估，其具有的六个特性如图8的x轴所示。

硬件参数:ttFlash-Emu使用相同的参数其SSD容量仅为48 GB（受机器限制） DRAM）。我们使用一台配备2.4GHz 8核英特尔至强处理器E5-2630-v3和64-GB DRAM 的机器进行模拟。该模拟和模拟SSD驱动器预热工作量相同。

小的尾部延迟：图5显示了在ttFlash-Sim上运行的六个跟踪驱动实验的读取延迟的 CDF。请注意，我们只显示读取延迟;写入延迟快速而稳定，因为所有的写入都被cap-backed RAM 所吸收。如图5所示，基本方法（“Base”=具有通道阻塞和最佳FTL 且没有RAIN的默认SSDSim）表现出较长的尾部延迟。相比之下，因为我们逐个添加每个TTFLASH特征：+ PB，+ GTR和+ RGC），所以观察到显着的改进。当所有的特征被添加（RGC + GTR + PB）时，尾部的微小延迟接近无GC的情况，我们稍后解释。图6绘制了相同实验的平均潜伏期。该图强调，虽然TTFLASH和Base的等待时间在第90百分位相似（图5），但Base的长尾潜伏期严重影响其平均潜伏期。与 Base 相比，TTFLASH的平均潜伏期要快2.5-7.8倍。

被垃圾回收阻塞的 I/O ：为了显示加速后SSD内发生的事情，我们计算被GC阻塞的读取I / O的百分比（“％GC阻塞的I / O”），如图7所示。同样重要的是，强调GC阻塞的I / O会填满设备队列，造成排队延迟，阻止新的主机I / O进入设备，我们将其视为“％队列阻塞的I / O”。该图有两个部分：％GCblocked（垃圾

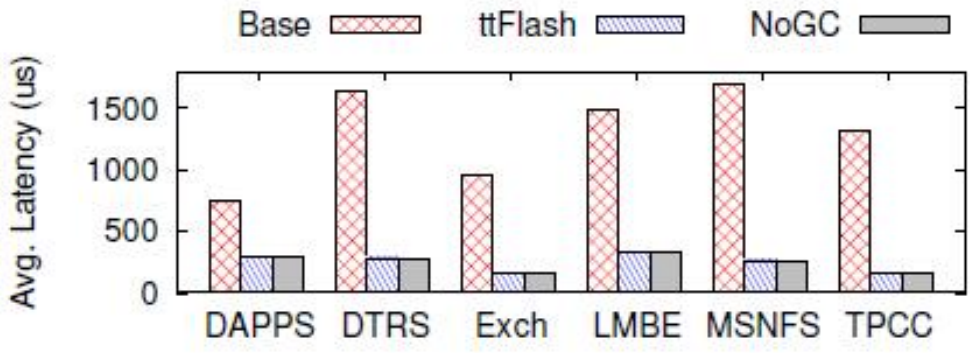

图 6: 平均延迟

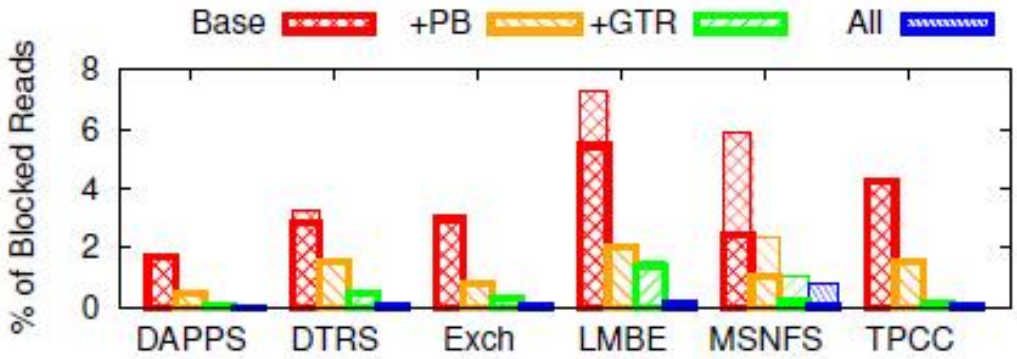

图 7: 垃圾回收阻塞的I / O 操作占比

回收阻塞）（底部，粗体边缘）和％queue-blocked（队列阻塞）的I / O（顶部），用一个小的水平边界划分。图7显示，对于没有允许垃圾回收的 Base，2-5％的读数被垃圾回收阻断。由于它们进一步导致排队延迟，总共有2-7％被阻塞的I / O不能被服务。随着每个TTFLASH功能的添加，更多的I / O被解锁。所有功能（“所有”栏）的阻塞I / O只有0.003-0.05％，除了MSNFS（0.7％）。为什么它不是0％的唯一原因是，对于非满读读取，只有当剩余时间短于额外读取的开销（如§4.3所述）时，TTFLASH才会等待垃圾回收完成。尽管只是暂时的，我们仍然把这些I / O计算在内。

6.2 TTFlash 与垃圾回收

垃圾回收可以通过更好的FTL管理，特殊编码，新颖的写入缓冲方案或基于SSD的日志结构文件系统进行优化和降低。例如，与基本方法相比，Value locality将擦除次数减少了65％[3]，闪存感知的RAID减少了40％[4]，BPLRU减少了41％，eSAP增加10-45％，F2FS增加10％，LARS增加50％，FRA增加10％，SFS为7.5×，WOM编码为33％。与这些工作相反，我们的方法是根本不同的。我们不关注减少垃圾回收的数量，而是消除垃圾回收操作的阻塞性质。随着还原，即使垃圾回收计数减少了多次，它也只会使垃圾回收引起的尾部潜伏期缩短，但不会消失（例如，如图5所示）。尽管如此，上述技术对于延长SSD寿命至关重要，因此与TTFLASH垂直。

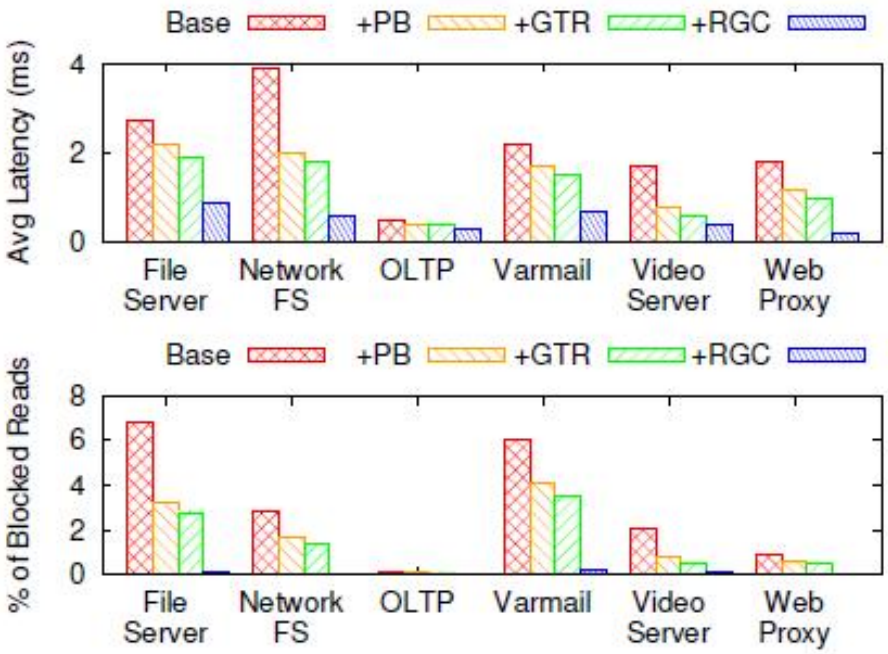

图 8: Filebench 在 ttFlash-Emu 上的测试结果：上下两幅图分别表示了读操作与由于垃圾回收阻塞的读操作占比

7 结论

SSD技术在过去几年发生了很大的变化，更快，更强大的闪存控制器是执行复杂逻辑的电缆;基于奇偶校验的RAIN已成为数据保护的标准手段;而电容支持的RAM是解决写入效率低下的事实上的解决方案。在我们的工作中，我们以前所未有的方式利用这些技术的组合。这反过来使我们能够构建诸如页面封闭垃圾回收，旋转垃圾回收，垃圾回收耐受读取和缓冲等新技术，这些技术共同为垃圾回收引发的尾部延迟的关键问题提供了强有力的解决方案。

参考文献

[1] DEAN, J., AND BARROSO, L. A. The tail at scale. *Commun. ACM 56*, 2 (2013), 74–80.

[2] GUPTA, A., PISOLKAR, R., URGAONKAR, B., AND SIVASUBRAMANIAM, A. Leveraging value locality in optimizing NAND flash-based ssds. In *9th USENIX Conference on File and Storage Technologies, San Jose, CA, USA, February 15-17, 2011* (2011), pp. 91–103.

[3] GUPTA, A., PISOLKAR, R., URGAONKAR, B., AND SIVASUBRAMANIAM, A. Leveraging value locality in optimizing NAND flash-based ssds. In *9th USENIX Conference on File and Storage Technologies, San Jose, CA, USA, February 15-17, 2011* (2011), pp. 91–103.

[4] HU, Y., JIANG, H., FENG, D., TIAN, L., LUO, H., AND ZHANG, S. P. Performance impact and interplay of SSD parallelism through advanced commands, allocation strategy and data granularity. In *Proceedings of the 25th International Conference on Supercomputing, 2011, Tucson, AZ, USA, May 31 - June 04, 2011* (2011), pp. 96–107.

[5] IM, S., AND SHIN, D. Flash-aware RAID techniques for dependable and high-performance flash memory SSD. *IEEE Trans. Computers 60*, 1 (2011), 80–92.

[6] KIM, H., AND AHN, S. BPLRU: A buffer management scheme for improving random writes in flash storage. In *6th USENIX Conference on File and Storage Technologies, FAST 2008, February 26-29, 2008, San Jose, CA, USA* (2008), pp. 239–252.

[7] KIM, J., LEE, J., CHOI, J., LEE, D., AND NOH, S. H. Improving SSD reliability with RAID via elastic striping and anywhere parity. In *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), Budapest, Hungary, June 24-27, 2013* (2013), pp. 1–12.

[8] YAN, S., LI, H., HAO, M., TONG, M. H., SUNDARARAMAN, S., CHIEN, A. A., AND GUNAWI, H. S. Tiny-tail flash: Near-perfect elimination of garbage collection tail latencies in NAND ssds. *TOS 13*, 3 (2017), 22:1–22:26.

[9] YOO, J., WON, Y., HWANG, J., KANG, S., CHOI, J., YOON, S., AND CHA, J. VSSIM: virtual machine based SSD simulator. In *IEEE 29th Symposium on Mass Storage Systems and Technologies, MSST 2013, May 6-10, 2013, Long Beach, CA, USA* (2013), pp. 1–14.

Notes

[1]取自2016年数据