第47卷第10期 中国科学技术大学学报 Vol.47,No.10
2017年10月 JOURNAL OF UNIVERSITY OF SCIENCE AND TECHNOLOGY OF CHINA Oct. 2017

文章编号:0253-2778(2017)10-0869-09

一种变粒度的闪存地址映射方案

樊 进,谭守标,陈军宁

(安徽大学电子信息工程学院,安徽合肥 230039)

摘要:闪存转换层最重要的功能是地址映射.地址映射需要同时具备高性能和低内存占用.基于需求的页级映射方案DFTL能有效节约内存且具有页级映射方案特有的灵活性,但是这种方案中,每个缓存槽只能存储一条映射记录,当缓存大小一定时,缓存中的映射记录数量有限;此外,这种方案没有考虑到请求的空间局部性.因此这种方案中,缓存命中率较低,DFTL需要频繁访问闪存来读取映射记录,降低了系统性能.于是针对命中率低的问题,提出了一种基于需求的变粒度映射方案VGFTL.这种方案可显著提高缓存的命中率.实验表明,VGFTL缓存平均命中率达到89.85%,远高于DFTL的45.46%,块擦除次数以及平均响应时间这两项指标优于DFTL,其性能接近纯页级映射方案.

关键词:闪存;闪存转换层;地址映射;变粒度映射

中图分类号:TP316 **文献标识码**:A doi:10.3969/j.issn.0253-2778.2017.10.010

引用格式:樊进,谭守标,陈军宁. 一种变粒度的闪存地址映射方案[J]. 中国科学技术大学学报,2017,47(10):869-877.

FAN Jin, TAN Shoubiao, CHEN Junning. A variable granularity-based mapping scheme [J]. Journal of University of Science and Technology of China, 2017,47(10):869-877.

A variable granularity-based mapping scheme

FAN Jin, TAN Shoubiao, CHEN Junning

(*School of Electronics and Information Engineering, Anhui University, Hefei* 230039, *China*)

Abstract: Address mapping is the most important function of flash translation layer (FTL), and it should have both high performance and low memory footprint. Although demand-based address mapping scheme (DFTL) has relatively high performance and low memory footprint, it has problems. First, it is a page-level mapping scheme. Each cache slot stores only one mapping record, and each mapping record stores only one physical page number and the corresponding logical page number, so that the cache , at a fixed size, can only store a limited number of mapping records. Second, each mapping record itself cannot exploit the spatial locality of the request. Thus, in the DFTL scheme, since the cache hit ratio is low, DFTL has to frequently access the mapping pages in the flash memory to read the mapping records, which significantly reduces the performance of the system. A new scheme named VGFTL (a mapping scheme of variable granularity-based flash translation layer) was proposed, which could significantly increase the cache hit ratio. Experimental results show that the average hit ratio of cache has reached 89.85% in VGFTL scheme, which is much higher than that of the DFTL scheme, 45.46%. VGFTL can significantly

收稿日期:2017-01-10;**修回日期**:2017-05-03

作者简介:樊进(通讯作者),男,1976年生,硕士/高级工程师.研究方向:嵌入式系统.E-mail:fanjin@ahu.edu.cn

reduce the number of block erasures and system response time compared to DFTL, and is close to pure page-level mapping scheme in performance.

Key words: NAND flash; flash translation layer; address mapping; variable granularity-based mapping

0 引言

闪存具有功耗低、发热小、重量轻、尺寸小等的优点.随着闪存容量瓶颈的突破、性能的优化以及成本的下降,闪存已经变成非常重要的数字存储介质.然而,闪存的异地更新、块擦写次数有限等特性使得现有的文件系统无法直接管理闪存.闪存存储系统一般使用闪存转换层(flash translation layer,FTL)将闪存模拟成传统块设备.地址映射是闪存转换层中最重要的功能,它使用映射记录保存供文件系统使用的逻辑地址和闪存上的物理地址之间的对应关系.为了实现快速寻址,全部或部分映射记录被缓存在内存中.因此,如何构建高效的地址映射方案并减少缓存对内存的占用成为一个重要问题.

闪存转换层目前主要使用三类地址映射方案[1]:纯页级映射[2]、块级映射[3-4]、混合映射[5].纯页级映射方案的地址映射单位为页,和闪存的读写单位一致,一个逻辑页可以映射到任意物理页,这种映射方案非常灵活,读写效率高,但是由于其映射粒度太小,导致映射表太大.块级映射的映射粒度大,以块为映射单位,逻辑地址包含逻辑块号和块内偏移量两个部分,映射表中只保存逻辑块到物理块之间的对应关系,内存占用小,但是这种映射方案不够灵活,更新数据前需要迁移块内多页数据并擦除块,性能很低.混合映射方案将大部分物理块作为数据块,用块级映射管理,少量物理块作为日志块用于存储更新数据,用页级映射管理.混合映射的映射表占用内存较小,但是带来了合并操作,因此性能不佳.

以上三种映射方案中.纯页级映射的性能最好,但是映射表占用内存大.为了解决这一问题,有学者提出了一种基于需求的页级映射方案(DFTL)[6],这种方案中,完整的映射表存储在闪存中的映射页中,而部分最新使用过的映射记录缓存在内存.DFTL采用最近最少使用算法(LRU)作为缓存的置换策略.缓存大小可通过配置缓存槽(Slot)数量来决定.因此,DFTL能有效减少内存占用并保持页级映射的高性能.但是被缓存的映射记录有限,导致DFTL需要频繁读写闪存来获取和回写映射记录,这些附加操作影响了DFTL的性能.

本文提出了一种基于需求的变粒度映射方案,这种方案的映射粒度不再是页,而是通过合并多条连续映射记录后产生的一条映射条目,缓存槽存储的是映射条目而非映射记录,因此其粒度大于或等于页,每个槽存储的映射条目所包含的页级映射记录数量不等.通过上述方式使得缓存中映射信息的存储密度大大增加,在缓存大小不变的前提下,增加了其中映射记录的数量,提高了缓存的命中率;另外,每条映射条目包含的都是逻辑地址上连续的多条映射记录,提高了系统的空间局部性能.实验结果表明:与DFTL相比,本文提出的方案在各种负载输入下,缓存的平均命中率达到89.85%,远高于DFTL的45.46%,块擦除次数以及平均响应时间这两项指标优于DFTL,接近纯页级映射方案.

1 地址映射方案

NAND型闪存由物理块组成,每个物理块由数量一定的物理页组成,块是擦除操作的基本单位,页是读写操作的基本单位.被擦除后的块被称为空闲块,其中的页被称为空闲页.

闪存有先擦后写的特性,即当物理页被写入数据后无法直接更新其中的数据,需要擦除该页所属的块后才能再次写入数据.我们期望闪存像磁盘一样采用本地更新,更新一页数据需要读取该页中所有有效数据并修改内容,然后擦除该页所在物理块,最后将修改后的多页数据回写到该块.由此可见,闪存的本地更新效率非常低下而且会造成物理块因被频繁擦写而快速老化.闪存更新数据采用异地更新,更新某个逻辑页的数据时,更新数据写入到其他空闲页中.异地更新方式造成管理闪存时必须采用地址映射机制记录逻辑地址到物理地址之间的对应关系.映射机制建立在映射表基础之上,为了加快寻址速度,映射表中全部或部分记录被缓存在内存中.

来自文件系统的逻辑地址通过地址映射被翻译成闪存中物理地址,地址映射方案根据映射粒度主要分3类:页级映射、块级映射和混合映射.

纯页级映射的映射粒度是页,和闪存的读写单

位一致，一个逻辑页可以映射到任何物理页，其映射记录中存储的是逻辑页号和对应的物理页号.这种映射方案非常灵活，性能最好，但是纯页级映射产生的映射表很大需要占用大量内存，不适合管理大容量闪存.块级映射的粒度是块，该方案将逻辑地址分为块号和块内偏移量，一个逻辑块对应一个物理块，其映射记录中存储的是逻辑块号和对应的物理块号，因此映射表很小.但是，块级映射在更新某个逻辑页的数据时，需要迁移该数据所在物理块上的所有有效数据到其他空闲块，触发大量的垃圾回收，系统性能很低.混合映射方案中，物理块在功能上被分为数据块和日志块，数据块用块级映射表寻址，日志块用于存储更新数据，用页级映射表寻址.为了使页级映射表不占用太多内存，混合映射方案中分配的日志块数量有限，因此当日志块耗尽时，需要将数据块和日志块中存储的有效数据进行合并，降低了系统性能.Gupta 等人提出的基于需求的页级地址映射 DFTL 是一种新型的页级映射方案，该方案保持了纯页级映射方案的优点，同时通过缓存部分映射记录解决了纯页级方案中映射表占用大量内存的问题.

DFTL 中，物理块在功能上被分为两类：数据块和映射块.数据块用于存储文件系统可访问的用户数据，映射块用于保存映射表.映射块中的页称为映射页，每个页保存了一定数量的映射记录.DFTL 在内存中建立了一个全局转换表(global translation directory，GTD)用于寻址映射页.DFTL 在内存中还建立了用于存储映射记录的缓存(cached mapping table，CMT)用于缓存被频繁使用的页级映射记录.由于 DFTL 在内存中只需要维护 GTD 和 CMT，因此内存占用较少.同时，CMT 采用 LRU 算法进行置换，因此，DFTL 的时间局部性性能较好.

在基于需求的页级地址映射方案中，为减少内存占用，被缓存的映射记录数量固定且数目很少.当来自文件系统的请求没有命中缓存时，需要产生额外的对映射页的操作：读取闪存中的映射页提取映射记录插入缓存，当缓存已满，则在插入前需要置换出最久没有使用到的映射记录，如果该映射记录在缓存中的生存周期内被修改，则需要被回写到闪存，回写闪存需要先读取该记录所在的映射页内容，修改内容后再异地写入到其他空闲页.综上，请求未命中缓存时，最好的情况是产生 1 次读映射页的操作，最坏的情况会产生 2 次读映射页和 1 次写操作.这些额外的操作影响了系统的性能.Xie[7] 提出了 ASA-FTL，同样采用的是页级映射，但是对冷热数据进行了鉴别并分别处理.Zhou[8] 提出的 TPFTL 采用了两级 LUR 队列来组织缓存中的映射记录，通过提高缓存的命中率来减少页映射记录的回写次数.Wei[9] 提出 Z-MAP 将闪存分为块映射区、页映射区和日志区，页级映射区存储随机数据，块映射区存储存储连续数据，日志区较小，作为写缓存，所有写入数据首先写入日志区，经过一段时间后判定日志区数据的连续性，然后写入块或页映射区，这种方案进一步减少了映射表大小，但是会不可避免地产生类似于混合映射方案中的合并操作.由此可见，缓存的命中率是影响系统性能的最重要因素.为了提高缓存的命中率，本文设计了一种新的地址映射方案，命名为基于变粒度的闪存转换层方案(variable granularity-based flash translation layer，VGFTL).

2　基于需求的变粒度映射方案

本文通过对映射记录特性的分析，提出了一种基于需求的变粒度映射方案 VGFTL.VGFTL 利用映射记录之间的连续性，合并连续的多条映射记录产生一条映射条目，缓存置换的单位不再是映射粒度为页的映射记录，而是映射粒度更大的映射条目，在稍微增加内存占用的情况下，增加了缓存中映射信息的密度，等同于增大了缓存的容量，提高了缓存的命中率.

此外，由于映射条目包含的是逻辑地址上相邻的一到多条映射记录，因此请求的空间局部性得到了满足，进一步提高了缓存的命中率.

这种方案通过增加缓存中映射信息的密度来提高缓存中映射信息的总量，因此在缓存大小一定的条件下，这种方案不会因增多映射信息而减少缓存中缓存槽的数量，因此 VGFTL 可有效利用请求的时间局部性.

2.1　映射记录的连续性

页级映射方案中，映射记录存储逻辑页 lpn 和物理页 ppn 之间的对应关系，可表示为 $lpn \rightarrow ppn$.

n 条映射记录 $\{lpn_i \rightarrow ppn_i, lpn_{i+1} \rightarrow ppn_{i+1}, lpn_{i+2} \rightarrow ppn_{i+2}, \cdots, lpn_{i+n-1} \rightarrow ppn_{i+n-1}\}$，满足对任意 $0 \leqslant k \leqslant n$ 有 $lpn_{i+k+1} - lpn_{i+k} = 1$，则称这 n 条

映射记录为相邻的映射记录.如果 n 条相邻的映射记录存储的 ppn 满足对任意 $0 \leqslant k \leqslant n$ 有 $ppn_{i+k+1} - ppn_{i+k} = 1$,则称这 n 条映射记录是连续的.n 条连续的映射记录可以合并为一条映射条目:

$$\{lpn_i \rightarrow ppn_i \mid \text{length} = n\}.$$

其中,n 为映射条目长度.如图1所示,4条连续的映射记录可合并为1条映射记录.

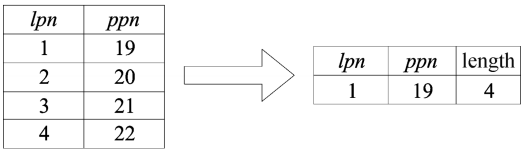

图1 连续映射记录合并成一条映射条目

Fig.1 Multiple mapping records are merged into mapping entry

2.2 二级合并机制

本文在内存中建立映射信息的缓存CMT,每个缓存槽存储1条映射条目.此外,在内存中建立计数表(count table,CT),CT中的记录和CMT中的条目一一对应,用于记录映射条目的状态和长度length.CT中每条记录占用1个字节,最高位用于表示条目状态,其余7位用于存储映射条目中映射记录的数量,因此一条映射条目可最多包含128条连续的映射记录.

CMT中映射条目是通过预合并和插入合并等二级合并机制产生的.

请求的目标总是一段逻辑地址区域,逻辑地址区域按页对齐可被分割成 L 个目标逻辑页,与此相对应,请求可以被分割成 L 个单位请求,每个单位请求的目标是一个逻辑页.

预合并发生在以下两种情况下:

(1)写请求到来时:写请求被分割成 L 个单位请求,数据被写入 L 个物理页,产生 L 条映射记录,这些映射记录是连续的或分组连续的,此时可以进行预合并,这种预合并称为对写请求的预合并;

(Ⅱ)请求到来且没有命中CMT时:VGFTL将接收到的请求分割成 L 个单位请求,如果某个单位请求没有命中缓存,则需要读取闪存中的映射页获取映射记录,在读取映射页后,可以将读取的目标映射记录和其周边的映射记录进行比较,如果连续,可以进行预合并,这种预合并称之为对映射页内容的预合并.

插入合并发生在将预合并生成的映射条目插入缓存的过程中.如果预合并生成的映射条目与缓存中现有的映射条目在逻辑地址空间上存在交集或连续的情况,则需要通过插入合并来处理.下面对预合并和插入合并的实现方式作详细说明.

2.2.1 预合并

(Ⅰ)对写请求的预合并

闪存转换层总是持有一个物理块作为写缓存块,当写缓存块写满后,闪存转换层会在闪存上寻址一个空闲块作为新的写缓存块.写请求的数据按页对齐被分割成 L 个页,写入时,按逻辑页编号从小到大将数据逐页写入写缓存块.闪存的特性决定了向物理块写入数据时必须按照块内的页编号从低到高写入数据,因此当写缓存块中空闲页数量 $F \geqslant L$ 时,写请求产生的映射记录是连续的,可以预合并成一条映射条目;当 $F < L$ 时,请求产生的映射记录是分组连续的,每个分组中的映射记录可以预合并成一条映射条目.

图2为写请求产生预合并的方法示例.假设每个块包含8个物理页,每个物理页编号需要4个字节存储.图2(a)中写请求1包含4页数据,当前写缓存块有6个空闲页,4页数据写入完成后,产生4条映射记录.在DFTL方案中,需要向缓存中插入4条映射记录,占用内存空间32个字节,而合并后产生1条映射条目,映射条目的长度用1个字节存储,占用内存9个字节.图2(b)中写请求2包含12页数据,当前写缓存块3有2个空闲页,则 $lpn1 \sim lpn2$ 两页数据写入写缓存块3,产生第1组共2条映射记录,合并后产生1条映射条目;$lpn3 \sim lpn10$ 写入闪存转换层分配的新写缓存块5,产生第2组共8条映射记录,合并后产生1条映射条目;$lpn11 \sim lpn12$ 写入闪存转换层再次分配的新写缓存块8,产生第3组共2条映射记录,合并后产生1条映射条目.在DFTL方案中,写请求2需要向缓存中插入12条映射记录,占用内存空间96个字节,而合并后产生的3条映射条目,占用内存27个字节.

(Ⅱ)对映射页内容的预合并

完整的页级映射表存储在闪存中,映射记录按照其中存储的逻辑页号 lpn 顺序存储在映射页中.当请求没有命中缓存时,闪存转换层需要从映射页中读取映射记录.在读取时,可以合并该映射记录周边的、符合连续性判定条件的多条映射记录.

图3为读取映射页时进行预合并的方法示例.请求1没有命中缓存,需要读取映射页3,假设每页可存储8条映射记录,8条映射记录被读取出来后,以映射页中 $lpn = 27$ 这条映射记录为起点,向前检

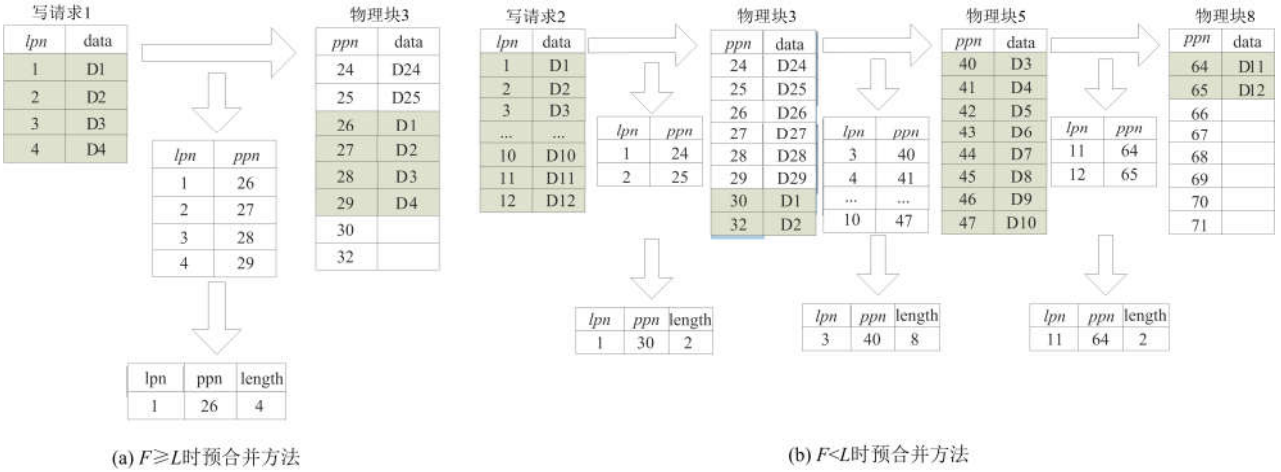

图 2　写预合并方法

Fig.2　Multiple mapping records are merged into mapping entry when FTL deals with writing request

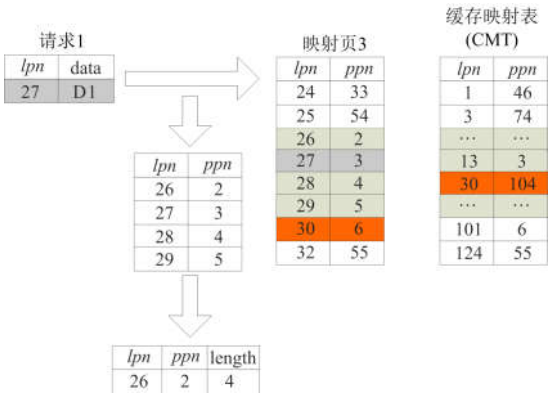

图 3　读取映射页时进行预合并

Fig.3　Multiple mapping records are merged into mapping entry when FTL reads mapping page

索映射记录，检索到 $lpn=26$ 这条映射记录可以和 $lpn=27$ 这条映射记录合并，而 $lpn=25$ 这条映射记录不符合连续性判定条件，该方向检索结束.以映射页中 $lpn=27$ 这条映射记录为起点，向后检索映射记录，直至 $lpn=29$ 这条映射记录，共 2 条记录可以合并，$lpn=30$ 这条映射记录已经存在于缓存中，因此不能被合并进映射条目.通过合并，4 条连续的映射记录可合并为 1 条映射条目.

2.2.2　插入合并

缓存 CMT 采用分段 LRU 算法，用两个定长的表来实现，一段是热记录表，一段是换出候选表，热记录表中存储的是最近使用过的映射条目，而换出候选表存储的是较久没有使用过的映射条目.预合并产生的映射条目需要插入 CMT，在插入前要判断该条目和 CMT 中既有的映射条目在逻辑地址上是否存在交集.为了降低判断的复杂度，VGFTL 方案中，热记录表和换出候选表中存储的映射条目都是按条目的 lpn 的值按从小到大顺序存储，检索 CMT 中的条目时采用二分法搜索算法，其时间复杂度为 $O(\log n)$，插入新条目时采用的是折半插入排序，其时间复杂度为 $O(n^2)$.

读取映射表时，预合并产生的映射条目不会和 CMT 中既有的映射条目产生交集，但有其包含的映射记录集可能和既有的映射条目包含的映射记录集符合连续性判别条件，因此需要在插入时判断是否可以再次合并成为一条映射条目.如图 4(a)，逻辑首地址为 1132、长度为 3 的映射条目插入缓存时，可以与缓存中逻辑首地址为 1135 的条目进行合并，产生新的映射条目，新映射条目逻辑首地址和物理首地址分别为 1132 和 517，映射条目长度变为 7.如图 4(b)所示，逻辑首地址为 1139、长度为 3 的映射条目插入缓存时，会与缓存中的首地址为 1135 长度为 4 的映射条目合并，合并后的映射条目依然原位保存在缓存中.

写请求产生的映射条目和 CMT 中的条目在逻辑地址空间上可能存在交集，如图 4(c)所示.当逻辑首地址为 1136 的映射条目插入 CMT 时，该条目和 CMT 中的逻辑首地址为 1135 的映射条目在逻辑地址空间上有交集，需要进行重新组合，组合后产生 3 个新映射条目，其中逻辑首地址为 1135 的条目原位存储于 CMT，而其他两个条目此时处于游离状态，如果此时 CMT 中有超过两个空槽，则可以将游离状态的两个映射条目直接插入 CMT.如果 CMT 中空槽小于两个，则需要首先换出最久没有访问过的映射条目，然后将游离状态的新条目插入 CMT.

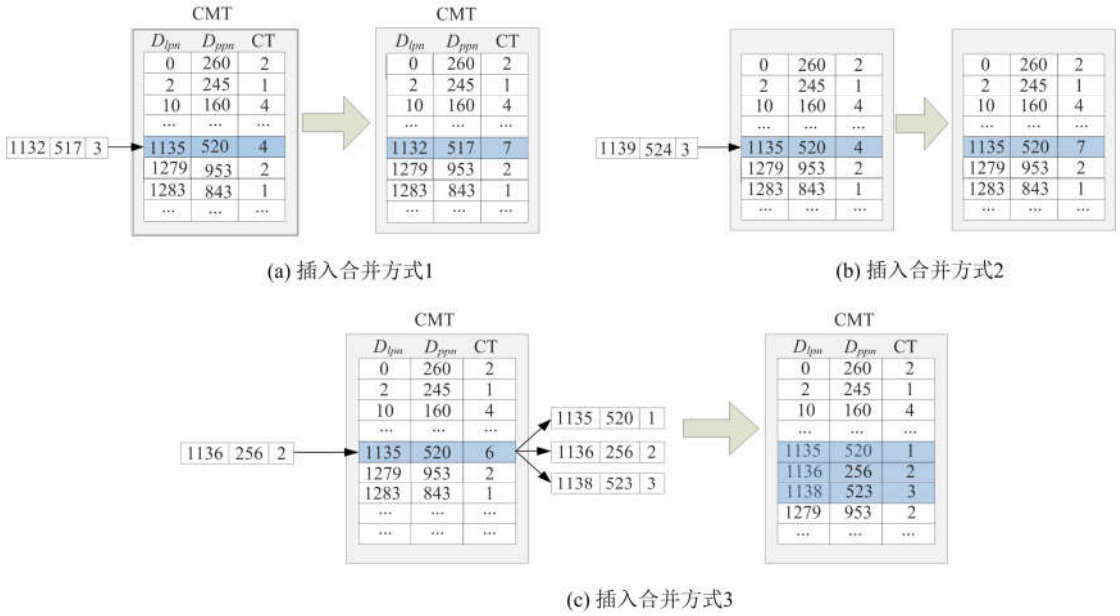

图 4 插入合并

Fig.4 Multiple mapping records are merged into mapping entries when FTL insert new entry to CMT

2.3 内存开销分析

VGFTL 在内存开销的组成上与 DFTL 稍有不同，两者都在内存中建立了映射信息缓存 CMT 和全局转换表 GTD.为了存储 CMT 中映射条目的长度和状态，VGFTL 方案中增加了计数表 CT，CT 中的每条记录占 1 个字节，CT 的表项数量和 CMT 的缓存槽数量相同并一一对应，当 CMT 中有 N 条缓存槽时，CT 的内存开销为 N 个字节.为了不增加内存开销，当映射记录用 M 个字节存储时，可将 VGFTL 方案中 CMT 的缓存槽数量减少为 DFTL 方案中的 $M/(M+1)$，这样使得 VGFTL 和 DFTL 的内存开销保持相同.虽然 VGFTL 方案中 CMT 的缓存槽略有减少，但是其 CMT 中实际缓存的映射记录数量能提高数倍，使得 VGFTL 在性能上依然优于 DFTL.

2.4 地址转换流程

算法 2.1 是 VGFTL 的地址转换算法.在 VGFTL 的地址转换算法中，将读写请求分开处理.

对于写请求，根据请求的长度(请求可被分割成逻辑页的数量)、当前写缓存块中空闲页数量以及每个物理块包含的物理页数量 PagesPerBlock 将请求分割成 n 组，形成 n 条映射条目，然后逐一处理每个分组，在处理分组中的每个单位请求时，如果某个单位请求没有命中 CMT，则根据该单位请求的目标 lpn 查找 GTD 获取目标映射记录所在的映射页，读取该映射页内容.在 VGFTL 算法中，首先对映射页内容的预合并产生映射条目，再将包含有目标映射记录的映射条目插入 CMT，这种处理可提高后续的单位请求命中 CMT 的几率.在 DFTL 的地址转换算法中，插入 CMT 的是从映射页内容中查找到的单个映射记录，如果后续的单位请求仍然没有命中 CMT，则需要再次重复查找 GTD、读取映射页的步骤，

对于读请求，则将请求分割成单位请求后，逐一处理单位请求，对于没有命中 CMT 的单位请求，同样会在读取映射页后进行预合并，插入 CMT 的是预合并后的映射条目，提高了后续的单位请求命中 CMT 的概率.

算法 2.1 VGFTL 地址转换过程

```
Input:请求的起始地址(lpn),请求长度(size),请求类型(type)/*type==0 为写请求,1 为读请求*/
Output:NULL
If type == 0 then
    /* 将请求分为 n 组,产生 n 条映射条目 */
    n ← split_write_request()
    /* 处理每一组 */
     for i from 0 to n-1
      /*组内的单位请求的逻辑页编号形成 lpns[0…m],处理其中的每个单位请求*/
       for j from 0 to m-1
         if 单位请求 lpns[j]命中 CMT then
           CMT_hits++
         else
```

```
            flash_hit++
            /*从映射页中读取并预合并映射记录产生映射条目,映射条目插入 CMT*/
            handle_hit_flash(lpns[j])
          end if
        end for
        for j from 0 to m-1
          send_flash_request(lpns[j])/* 数据写入 flash*/
        end for
        /*映射条目和 CMT 中既有条目在逻辑地址上进行比较,可能会产生多个游离条目*/
        Detached_items[0…d] ← check_cmt(lpns[i],lpns_count[i])
        /*将游离条目插入 CMT */
        for k from 0 to d-1
            insert_item_to_cmt (Detached_items [k])
          end for
        end for
    else
      while size ≠ 0 do
        if 单位请求命中 CMT then
          CMT_hits++
        else
          flash_hit++
          handle_hit_flash(lpn)
        end if
        send_flash_request(lpn)/* 从 flash 读出数据*/
        lpn++
        size--
      end
    end if
```

3　实验与分析

本文的实验仿真平台是基于 DiskSim3.0[10] 扩展而来的 FlashSim[11] 仿真器.FlashSim 模拟了闪存的读取、写入、擦除 3 种基本操作,能给出不同闪存转换层的性能分析报告.FlashSim 内置了纯页级映射、混合映射方案 FAST 和 DFTL 等 3 种算法的实现.本文在 FlashSim 中添加了 VGFTL 的实现,配置了一个 16 GB 的 NAND 闪存存储系统.具体配置参数如表 1 所示.

本文选取 4 种具有不同特性的负载作为仿真输入,这些负载都是现实环境下的各种系统的 I/O 请求集,为闪存转换层研究者广泛使用.负载的基本情况见表 2.其中,WebSearch[12] 负载是某个流行的搜索引擎的 I/O 请求的记录,以读操作为主.Financial[11] 负载来自两个大型金融机构的 OLTP 应用 I/O 请求的记录.Exchange Server[13] 负载是 Exchange Server 在 24 小时内的磁盘访问记录集.MSR Cambridge[12] 负载记录了一周内 Microsoft Research Cambridge 的企业服务器上的磁盘 I/O 请求.

表 1　实验中的闪存参数

Tab.1　Parameters of the flash memory in experiment

参数	设置值
闪存容量/GB	16
页大小/kB	2
每块页数量	64
垃圾回收阈值(最少空闲块数量)	10
每个映射页能存储的映射记录条数	512
逻辑页编号或物理页编号占用字节数	3
读页延迟/μs	32.725
写页延迟/μs	101.475
块擦除时间/ms	1.5

表 2　负载特性

Tab.2　Specification of workloads

负载	请求数	平均请求大小/kB	写请求/%
Financial1	5 334 983	9.84	76.84
Financial2	3 699 194	7.78	17.65
WebSearch	1 320 675	29.01	0
Exchange	3 108 094	56.35	92.82
MSR Cambridge	3 723 808	29.93	35.13

实验比较了 VGFTL、DFTL、纯页级映射方案的性能,包括 3 个性能指标:缓存命中率、擦除次数、系统平均响应时间.

本文通过定时采样的方法确定不同时刻 CMT 中的一级映射记录数量,每处理 10 000 次请求进行一次采样.如图 5(a)所示,在 VGFTL 方案中,缓存映射条目所表达的映射记录数量比 DFTL 有了大幅度的增加.Financial1 和 Financial2 这两种负载平均请求长度较小,因此缓存中平均映射记录数量增幅稍小,是 DFTL 的 2 倍以上.对于 WebSearch、Exchange 和 MSR Cambridge 这 3 种请求长度较大的负载,缓存中平均映射记录数量达到 DFTL 的 6 倍以上.

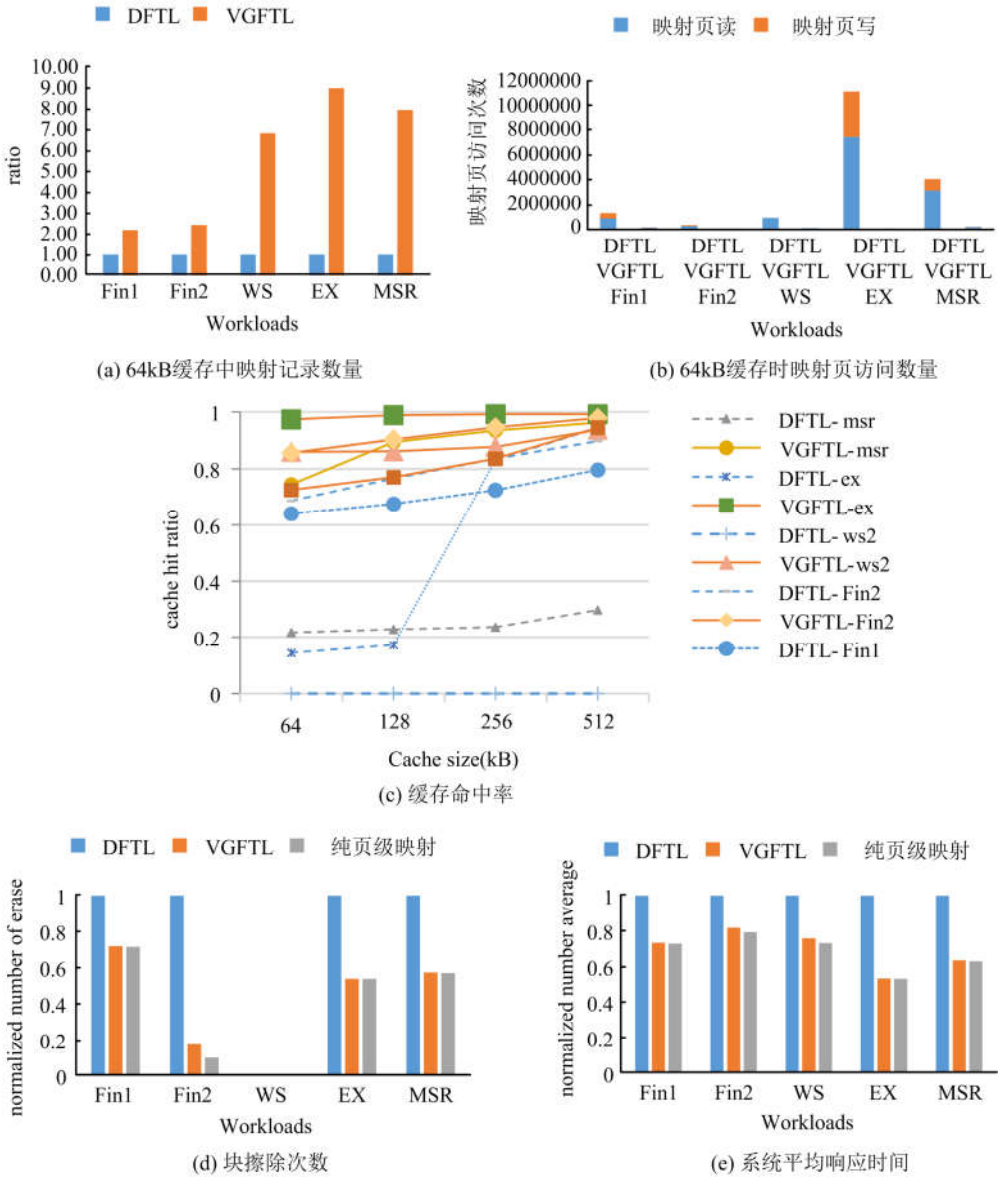

图 5 实验结果

Fig.5 Experimental results

3.1 缓存命中率

DFTL 和 VGFTL 两种方案中，GTD 消耗内存都为 48 kB.对于 DFTL，其缓存是指 CMT.对于 VGFTL，其缓存包括 CMT 和 CT，VGFTL 中增加了 CT，需要占用一部分内存.为了在相同内存开销下进行性能比较，在缓存大小分别被设定为 64 kB、128 kB、256 kB、512 kB 时，VGFTL 的缓存槽的数量分别设定为 9362、18724、37449、74898，而 DFTL 中缓存槽的数量分别设定为 10922、21845、43690、87381.VGFTL 下的缓存槽数量为 DFTL 下的缓存槽的 85.71%.采用变粒度映射方案的 VGFTL 通过缓存映射条目可以在缓存中存储更多的映射记录，因此其命中率在各种负载输入下都高于 DFTL.如图 5(c)，VGFTL 的命中率平均达到 89.85%，DFTL 的缓存命中率平均为 45.46%.纯页级映射方案的缓存命中率总是 100%，在图中没有标出.

3.2 擦除次数

命中率的高低决定了映射页被读写的次数，映射页的写入次数越多，产生的垃圾数据越多，垃圾回收次数也越多，闪存块被擦除的次数越多越好.图 5(d)是缓存大小为 64 kB 时各种负载输入下产生的擦除次数，VGFTL 在各种负载输入下，其擦除次数都小于 DFTL，接近纯页级映射的擦除次数.

3.3 平均响应时间

VGFTL 向缓存中插入映射条目时，可能会因待插入条目和缓存中既有的多个条目之间在逻辑地址空间上存在冲突，需要进行重新合并，进而产生多个游离态的映射条目.此时如果缓存中已无空槽位供插入，则 VGFTL 需要在换出候选表中一次性选择多个条目换出.由于缓存中映射记录数量大幅度增加、映射条目使具有空间局部性的请求在缓存中直接寻址的几率变大，这些优点使得 VGFTL 的缓存命中率更高，缓存换出的频率大大降低，如图 5(b)所示.在各种负载下，VGFTL 对映射页的访问次数都远少于 DFTL 访问映射页的次数，读写映射页的次数变少，即访问数据时产生的额外操作变少，而且写映射页的次数变少使得整体的擦除次数也会随之减少，这些因素的减少会大大缩短系统平均响应时间.图 5(e)是缓存大小为 64 kB 时各种负载输入下的系统平均响应时间.VGFTL 方案中的系统平均响应时间比 DFTL 的更小，接近纯页级映射方案中系统平均响应时间.

4 结论

本文提出了一种变粒度映射方案.通过增大映射粒度，每个缓存槽中可存储一到多条映射记录，提高了缓存中映射信息的密度.在相同的内存开销下，缓存中可以缓存更多的映射记录；由于每个槽中可存储一到多条逻辑地址相邻的映射记录，请求的空间局部性得到了满足.以上特点使得缓存的命中率得到大幅度提升，显著减少了读写映射页的次数.实验证明，VGFTL 的系统性能超过 DFTL，接近纯页级映射.

参考文献(References)

[1] CHUNG T S, PARK D J, PARK S, et al. A survey of flash translation layer [J]. Journal of Systems Architecture, 2009, 55(5-6): 332-343.

[2] XU G X, LIN F Y, XIAO Y P. CLRU: A new page replacement algorithm for NAND flash-based consumer electronics [J]. IEEE Transactions on Consumer Electronics, 2014, 60(1): 38-44.

[3] CHOUDHURI S, GIVARGIS T. Performance improvement of block based NAND flash translation layer [C]// Proceedings of the 5th IEEE/ACM International Conference on Hardware/Software Codesign and System Synthesis. Salzburg, Austria: ACM, 2007: 257-262.

[4] QIN Z W, WANG Y, LIU D, et al. Demand-based block-level address mapping in large-scale NAND flash storage systems[C]// Proceedings of the 8th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis. Scottsdale, USA: ACM, 2010: 173-182.

[5] CHO H, SHIN D, EOM Y I. KAST: *K*-Associative sector translation for NAND flash memory in real-time systems[C]// Proceeding of the Design, Automation & Test in Europe Conference & Exhibition. Nice, France: IEEE, 2009: 507-512.

[6] GUPTA A, KIM Y, URGAONKAR B. DFTL: A flash translation layer employing demand-based selective caching of page-level address mappings[C]// Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems. Washington, USA: ACM, 2009: 229-240.

[7] XIE W, CHEN Y, ROTH P C. A low-cost adaptive data separation method for the flash translation layer of solid state drives[C]// Proceedings of the International Workshop on Data-Intensive Scalable Computing Systems. Austin, USA: ACM, 2015: No.3, 1-8.

[8] ZHOU Y, WU F, HUANG P, et al. TPFTL: An efficient page-level FTL to optimize address translation in flash memory [C]// Proceedings of the 10th European Conference on Computer Systems. Bordeaux, France: ACM, 2015: No.12, 1-16.

[9] WEI Q S, CHEN C, XUE M D, et al. Z-MAP: A zone-based flash translation layer with workload classification for solid-state drive [J]. ACM Transactions on Storage, 2015, 11(1): No.4, 1-33.

[10] BUCY J S, GANGER G R. The DiskSim simulation environment version 3.0 reference manual[BE/OL]. [2017-10-18], http://www.pdl.cmu.edu/DiskSim/.

[11] A simulator for various FTL schemes [BE/OL]. Department of Computer Science and Engineering, Pennsylvania State University, [2017-10-18], http://csl.cse.psu.edu/? q=node/322.

[12] Websearch trace and OLTP trace from umass trace repository[BE/OL]. [2017-10-18], http://traces.cs.umass.edu/index.php/Storage/Storage.

[13] Block traces from SNIA [BE/OL]. [2017-10-18], http://iotta.snia.org/traces/list/BlockIO.