

OAFTL: 一种面向企业级应用的高效闪存转换层处理策略

綦晓颖¹ 汤显^{1,2} 梁智超¹ 孟小峰¹

¹(中国人民大学信息学院 北京 100872)

²(燕山大学经济管理学院 河北秦皇岛 066004)

(87qixiaoying@ruc.edu.cn)

OAFTL: An Efficient Flash Translation Layer for Enterprise Application

Qi Xiaoying¹, Tang Xian^{1,2}, Liang Zhichao¹, and Meng Xiaofeng¹

¹(School of Information, Renmin University of China, Beijing 100872)

²(School of Economics and Management, Yanshan University, Qinhuangdao, Hebei 066004)

Abstract NAND flash based devices usually introduce a software firmware called flash translation layer (FTL) to simulate the flash memory like a block device. FTL is critical to the performance of flash-based devices. Most existing FTL algorithms work normally in embedded systems. However, they behave poorly when there are frequent random accesses in enterprise applications. In this paper, we propose an operation aware flash translation layer (OAFTL) for enterprise-scale storage devices based on page-level mapping scheme. OAFTL manages the entries cached in RAM according to read/write operations separately. Besides, OAFTL supplies a log page for translation page to relieve frequent updates of translation information to improve performance. The experiment result shows that our OAFTL algorithm works effectively for enterprise workload. In our experiments, OAFTL improves the total performance by more than 20 percent compared with the existing methods.

Key words NAND flash; flash translation layer; page-level address mapping; storage management; SSD

摘要 基于 NAND 闪存的存储设备通过引入闪存转换层来对闪存芯片进行封装,使得闪存存储设备像普通块设备一样使用. 闪存转换层算法的性能很大程度上决定了闪存设备的存储性能,已有方法尽管可以在嵌入式环境下正常工作,但当应用到随机访问频繁的企业级应用环境中时存在访问性能低的问题. 提出了一种面向企业级应用的闪存转换层算法 OAFTL,该算法基于页级地址映射,根据访问操作的类型来组织映射项信息,通过为映射页保留日志信息来缓冲频繁修改的映射信息,以提高闪存读、写性能. 实验结果表明,提出的 OAFTL 算法能够有效地适应企业级工作负载,同已有方法相比,综合读写性能提升了 20% 以上.

关键词 NAND 型闪存; 闪存转换层; 页级地址映射; 存储管理; 固态硬盘

中图法分类号 TP311.13

磁盘作为一种主流的二级存储介质,已广泛应用于个人电脑和企业级数据存储等领域. 虽然磁盘

存储市场每年的市场份额高达 340 亿美元^[1],其机械寻道的特性限制了其 IO 性能的快速提升. 在随

机访问密集型企业级应用中使用磁盘存储介质时,但其存取性能表现并不理想.和磁盘相比,闪存是一种纯电子设备,具有高速、高吞吐、低功耗、抗震、小巧轻便等优点^[2].早在2006年,图灵奖得主 Jim Grey 就明确指出“就像磁盘取代磁带一样,闪存将会取代磁盘”^[3].随着闪存容量的不断增长和价格的不断下降,其应用领域已经逐步扩展到随机访问密集型企业级应用环境.

不同于磁盘存储器,闪存存储器有其自身的特性.磁盘中的数据可以原地更新,而闪存不支持原地更新操作,需要在更新前进行擦除,即更新某个页中的数据时,必须对该页所在块进行擦除操作,由于写操作代价较高,频繁的写操作会很大程度上影响系统的整体性能.为屏蔽闪存更新前擦除的特性,基于闪存的存储设备中通常会引入闪存转换层^[4-5](flash translation layer, FTL)进行逻辑地址到物理地址的转换.一般的,闪存存储设备中包含一个控制器、小容量的 RAM 和多个闪存芯片. FT L 主要在控制器中实现,其主要设计思想是利用较小的 RAM 提供高速的读写性能. FT L 是闪存存储设备中的核心组件之一,它通过对闪存芯片进行封装来实现向上层文件系统提供读、写接口的目的,从而使得闪存存储设备像普通的块存储设备一样工作.因此, FT L 算法的设计策略很大程度上决定了闪存存储设备的工作性能.

目前 FT L 算法的研究工作已有很多,典型的算法有 BAST^[6], FAST^[7], SuperBlock^[8], LAST^[9] 和 DFTL^[10]. 这些 FT L 算法在实际应用中存在以下一些问题.

1) 闪存空间利用率低. 已有的 BAST, FAST, SuperBlock, LAST 算法都是基于混合级地址映射策略,且引入了日志块的思想,其问题是日志块所占空间太小^[11],当日志块满时需要进行垃圾回收操作,进而引发昂贵的擦除操作. 因此,即使闪存中有很多空闲的数据块,部分数据的频繁更新操作仍然会引发垃圾回收操作,进而造成闪存的空间利用率低.

2) 随机访问性能不高. 目前,大部分的 FT L 算法对顺序访问的处理性能较好,而对随机写操作的处理性能较差. 例如,在随机写频繁的情况下,典型的 FAST 算法将不同数据块的更新页写入同一个日志块,当日志块满时就会触发垃圾回收操作. 回收日志块时需要合并日志页与对应数据页的数据,导

致垃圾回收的代价过高,并且降低随机写操作的性能.

3) 无法适应企业级应用环境. 企业级应用通常具有如下特点:海量数据存储、数据的并发访问、数据持久化等. 其中,数据的并发访问特性会引发大量的数据随机访问操作. 现有的 FT L 算法^[6-9]大部分是面向嵌入式和通用系统,随机访问性能不好. 虽然 DFTL 算法^[10]的设计初衷是面向企业级应用,但其读写性能并不高.

针对以上问题,我们提出了一种高效的闪存转换层策略. 该策略基于页级地址映射机制,通过区别读操作与写操作,在 RAM 中分别组织不同操作类型的映射表,以快速响应上层请求. 针对实际中频繁更新某些映射项信息的问题,提出一种为映射页添加日志的更新策略,该策略通过缓冲频繁更新的映射项信息来达到减少闪存擦除次数、提高闪存访问性能的目的,进而从整体上提高闪存设备的读写性能,使其可以适用于数据密集型计算的企业级应用环境.

1 背景及相关工作

1.1 闪存

根据闪存内部物理构造的不同,闪存芯片主要分为两类: NOR 型闪存和 NAND 型闪存^[5]. NOR 型闪存芯片具有可靠性高、随机读取速度快的优势,但擦除和编程操作速度较慢,被广泛应用于直接执行程序代码的应用中,例如 BIOS、移动电话和磁盘驱动器的控制内存. 而 NAND 型闪存可提供高密度的存储性能,且擦除和写入速度也很快,具有单位成本效益高和芯片兼容性更好等优点. 高密度的 NAND 型闪存适用于大容量的数据存储. 本文的工作主要针对于 NAND 型闪存.

NAND 型闪存芯片由很多块组成,每个块由多个页(64 或 128 等)组成,一个页又包含数据区和备用区,备用区主要用于存储管理信息和纠错信息等. 根据块的大小, NAND 型闪存又可分为两种类型: 小块闪存和大块闪存^[12]. 由于大块闪存的密度更高,价格更低廉,越来越多的闪存厂商使用它作为存储单元,例如三星推出的 K9WAG08U1A 芯片^[13]. 一般来说,大块闪存介质中,一个页包含 2 KB 的数据区和 64 B 的备用区. 有一点需要注意,在大块闪存中,一个块中的页应该保证顺序写. 表 1 描述了大块闪存芯片的主要性能参数.

Table 1 Performance of Large Block NAND Flash

表 1 大块 NAND 闪存的性能参数

Parameter	Read	Write	Erase
Unit/KB	2	2	128
Time/ μ s	25	200	1500

NAND 型闪存有其独特的物理特性. 首先, 闪存的基本操作有 3 种: 读、写和擦除. 其中, 读和写操作以页为单位, 擦除操作以块为单位, 一个块通常包含 64 个页. 闪存的读、写、擦除操作代价各不相同, 擦除操作花费的时间远远大于读操作所需时间. 例如 SAMSUNG 公司 K9K8G08U0A 闪存芯片^[13] 读操作需 25 μ s, 写操作需 200 μ s, 擦除操作需 1.5 ms. 其次, 闪存中的数据更新前需要进行擦除, 即更新某个页中的数据时, 必须对该页所在块进行擦除操作. 最后, 闪存中每个块的擦除次数有限, 一般为 10 万次至 100 万次. 通常使用磨损平衡技术将擦除操作均匀分布在所有的数据块中, 以避免产生某些块因磨损度过高而失效的问题.

1.2 闪存转换层

由于闪存的物理特性和磁盘截然不同, 为了便于上层文件系统的访问, 提供类似于磁盘的读写接口, 同时为了屏蔽闪存更新前擦除的特性, 基于闪存的存储设备一般都会引入闪存转换层^[14]. 闪存转换层在系统中的层次如图 1 所示. 闪存转换层主要用于封装闪存芯片, 向上层文件系统提供读、写接口, 使得闪存设备像磁盘块设备一样使用. 闪存转换层的主要功能是实现逻辑地址到物理地址的转换; 提供垃圾回收机制, 循环使用闪存空间; 提供磨损平衡机制, 延长闪存的使用寿命^[5]. 目前已有很多工作研究 FTL 算法, 根据地址映射粒度的不同, 现有的 FTL 算法可分为 3 类: 页级地址映射算法、块级地址映射算法和混合级地址映射算法^[15-16].

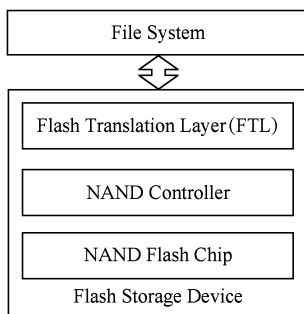


Fig. 1 Flash translation layer in flash storage device.

图 1 闪存转换层在系统中的层次

页级地址映射可把一个逻辑页映射到闪存的任何一个物理页. 该策略映射信息管理的灵活度高, 闪存空间利用率高, 垃圾回收负载小. 它的缺点为映射信息多, 占用 RAM 空间大, 且映射表大小与闪存的容量成正比.

块级地址映射只是把一个逻辑块号映射到一个物理块号上, 逻辑块内的偏移地址等于物理块内的偏移地址. 因此, 块级地址映射表只需要很小的 RAM 空间存储映射信息. 与此同时, 这样就会导致地址映射的灵活性消失, 闪存空间利用率下降.

为解决页级和块级地址映射的问题, 作为一种折中方案, 混合级地址映射算法应运而生. 该方法根据块的用途把闪存块分为两类: 数据块和日志块. 数据块使用块级地址映射策略, 日志块使用页级地址映射策略. 目前, 大部分的 FTL 算法都是使用这种混合式算法, 它们之间的区别主要在于数据块和日志块的组织方法.

1.3 典型的 FTL 算法

闪存转换层的性能直接影响了闪存存储器的性能. 目前对闪存转换层的研究有很多, 下面列出了几个典型的 FTL 算法, 并把它们作了相应的比较.

典型的基于块映射的 FTL 算法是替换块算法^[4], 当数据块进行更新时, 需要把新数据写入到新分配块对应的偏移位置, 再次进行更新时继续分配一个新块, 仍旧将更新数据写入相同的偏移位置. 该算法是一种原始的 FTL 算法, 它的缺点是空间利用率太低, 读写性能不高, 且数据的拷贝和擦除操作比较频繁.

典型的基于页级地址映射的 FTL 算法是 DFTL 算法^[9]. 它把所有的映射信息都存储在闪存上, RAM 中只缓冲最近经常使用的映射项信息. DFTL 算法有效提高了闪存的空间利用率, 但其读写操作性能并不高.

基于混合级地址映射的算法有 BAST, FAST, SuperBlock, LAST. Kim 等人在 2002 年提出了 BAST 算法^[6], 该算法在一定程度上改进了替换块算法. BAST 算法中一个数据块对应一个日志块, 同一数据块的更新操作按照顺序写入与其对应的日志块中. BAST 算法提高了闪存的空间利用率. 然而, 在随机写频繁的情况下, 日志块的利用率就会很低, 并且会引起频繁的块擦除操作, 导致随机访问性能降低. 为解决日志块利用率低的问题, Lee 等人在 2007 年提出了 FAST 算法^[7]. 该算法中一个日志块可存储所有数据块的更新数据, 这就大大提高了日志页

的空间利用率. 随之带来的问题是一个日志块中可能包含多个数据块的更新页, 进行垃圾回收时会导致日志块与多个数据块的合并操作, 其代价是高昂的. 而且由于日志块的数量只占闪存很小的比重, 当日志块不足时就会引发垃圾回收操作, 合并操作消耗时间较长, 影响运行效率. 2006 年 Kang 等人提出了 SuperBlock 算法^[8], 该算法将多个相连的逻辑块组织成一个超级块, 超级块内部采用页级地址映射算法, 采用 3 级地址转换策略, 并把转换信息存储在闪存页的备用区域. 该算法充分利用了实际环境中数据访问的局部性特征, 降低了垃圾回收的代价. 但超级块的粒度要根据实际应用不断调优, 且地址转换信息受限于页面备用区的大小. 因此, 该算法的灵活度太小, 不能满足不同应用的特殊要求. 2008 年 Lee 等人提出了 LAST 算法^[9], 该算法试图解决 FAST 算法带来的问题, 充分利用了访问序列的时间局部性与空间局部性, 把闪存划分为两个不同的区域, 根据访问类型的不同数据被存储到不同的区域. 该算法引入一个局部探测器来识别访问类型, 但是, 对于局部小的顺序写该探测器不能作出有效判断. 上述算法主要应用于嵌入式设备或通用计算机系统.

通过对已有 FTL 算法的分析比较得知, 上述算法都在不断地提高闪存空间的利用率, 大部分的算法主要适用于小容量数据存储规模, 而对企业级存储应用来说, 高效的 FTL 算法相对较少. 因此, 本文中我们致力于设计一种面向企业级应用的高效的闪

存转换层策略, 不仅要提高闪存的空间利用率, 而且还要降低垃圾回收的代价.

2 OAFTL 算法

现有的 FTL 算法大部分都是基于混合级地址映射策略, 因此不可避免地会产生日志块与数据块的合并操作, 其代价是非常昂贵的. 此处, 我们提出了一种基于页级地址映射的 FTL 算法, 该算法根据操作类型的不同分别组织地址映射信息, 我们把它称之为 Operation Aware FTL, 简称 OAFTL. OAFTL 消除了日志块的概念, 进行垃圾回收时也不需要进行日志块与数据块的合并操作. 该算法在闪存中开辟一块空间, 用于保存所有的映射信息. RAM 中只缓冲频繁使用的映射项, 即逻辑地址到物理地址的转换信息. 根据访问操作类型的不同分别组织 RAM 中的映射表. 该策略充分利用企业级应用中数据分布的局部性原理, 可以在 RAM 中缓冲足够的映射项, 以满足上层文件系统发出的读写请求.

2.1 OAFTL 系统架构

OAFTL 算法使用页级地址映射策略, 所有的映射信息都存储在闪存介质上, 为有效地管理页级地址映射信息, 我们把多个映射项组织到一个页中. 其中近期频繁使用的映射项缓冲在 RAM 中, RAM 中的映射表被分为两个子表, 我们称之为读操作映射表(read cache mapping table, RCMT)和写操作

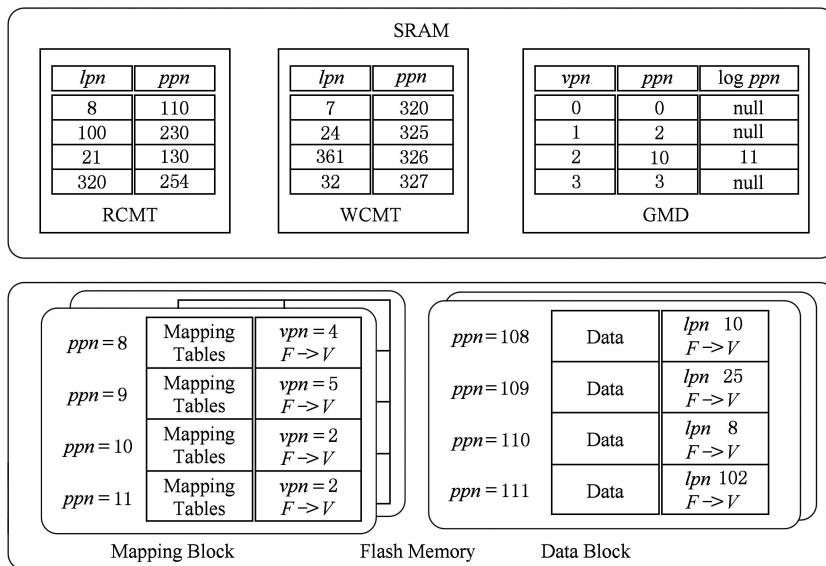


Fig. 2 OAFTL architecture.

图 2 OAFTL 系统架构

映射表(write cache mapping table, WCMT). 读操作映射表用于缓冲读操作的映射项信息, 写操作映射表用于缓冲写操作的映射项信息. 通过引入读操作映射表和写操作映射表, 我们可以区分加载到 RAM 中的映射项是否被修改: 读操作映射表中的项没有被修改, 写操作映射表中的项被修改. 除此之外, 我们为每个转换页增加一个日志页, 来缓解频繁更新映射项带来的负载.

根据存储信息的不同, 闪存上的页可分为两种: 数据页和映射页. 数据页中存储数据信息, 映射页中存储逻辑地址到物理地址的映射信息. 其中, 一个映射页中可存储多个映射项. 闪存上存储数据页的块称为数据块, 存储映射页的块称为映射块. 在 RAM 中, OAFTL 维护两张映射表, 分别是读缓冲映射表和写缓冲映射表. 除此之外, 还要维护一个全局映射目录(global mapping directory, GMD). 当上层文件系统请求的逻辑页号不在 RCMT 或 WCMT 中时, 需要根据 GMD 到映射页中查找映射信息, 并读入 RAM. OAFTL 的主要架构及数据结构如图 2 所示.

2.2 地址转换算法

在 OAFTL 算法中, 根据操作类型的不同, 逻辑到物理地址的转换过程也各不相同. 在本小节中, 我们将分别介绍如何处理读操作和写操作.

1) 读操作. 如果上层文件系统发出读命令, OAFTL 首先扫描写缓冲映射表, 再扫描读缓冲映射表. 如果请求的映射项存在于 WCMT 或 RCMT 中, 那么可根据映射项信息直接寻址, 找到请求的页(行②~⑥). 如果请求的映射项不在 WCMT 或者 RCMT 中, OAFTL 将会检查 RCMT 是否已满. 如果 RCMT 已满, 我们将利用 LRU 算法^[16]的思想在 RCMT 中选择一个最近最不经常使用的映射项驱逐出 RCMT 表(行⑧~⑩). 然后 OAFTL 将会根据 GMD(全局映射目录)读取映射信息, 并把新获取的映射项添加到 RCMT 中(行⑫~⑬). 最后, 根据映射项信息获取请求的页(行⑯).

算法 1. OAFTL(Request, LPN).

/* Request——请求的操作类型, LPN——请求的逻辑页号 */

- ① if(Type of Request==READ) then
- ② if(LP_N hits WCMT) then
- ③ Get PPN from WCMT;
- ④ else
- ⑤ if(LP_N hits RCMT) then

- ⑥ Get PPN from RCMT;
- ⑦ else
- ⑧ if(RCMT is full) then
- ⑨ Select victim entry using LRU algorithm;
- ⑩ Remove the victim entry from RCMT;
- ⑪ end if
- ⑫ Load the entry with LP_N into RCMT by GMD;
- ⑬ Get PPN by the entry;
- ⑭ end if
- ⑮ end if
- ⑯ Serve the Request by PPN;
- /* Type of the Request==WRITE, 写操作 */
- ⑰ else
- ⑱ if(LP_N hits RCMT) then
- ⑲ Move the entry from RCMT into WCMT;
- ⑳ else
- ㉑ if(LP_N hits WCMT) then
- ㉒ Follow algorithm line ㉑ to ㉓;
- ㉓ else
- ㉔ if(WCMT is full) then
- ㉕ Evict victim entries using Algorithm 2;
- ㉖ end if
- ㉗ end if
- ㉘ end if
- ㉙ Serve the Request with new PPN;
- ㉚ PPN in WCTM with LP_N=new PPN;
- ㉛ Store this mapping information(LP_N, PPN) to WCMT;
- ㉜ end if

算法 2. WCMT 中淘汰映射项的算法.

- ① Select the least recently used mapping entry in WCMT;
- ② LP_N←Logical page number of the victim entry;
- ③ if LP_N had a log page then
- ④ Update the translation page;
- ⑤ Update the GMD;
- ⑥ Evict the victim entries out of WCMT;
- ⑦ else
- ⑧ Allocate a log page;

- ⑨ Write the entries into log page;
- ⑩ Update the GMD;
- ⑪ Evict the victim entries out of WCMT;
- ⑫ end if

2) 写操作. 如果上层文件系统发出写命令, OAFTL 首先扫描 RCMT, 若请求的映射信息存在于 RCMT 中, 则该映射项从 RCMT 转移到 WCMT 中(行⑬~⑭), 写请求立即被执行, 即把数据写入到一个新页中, 同时修改映射项信息. 如果 RCMT 没有请求的映射项, OAFTL 会扫描 WCMT 的内容, 若命中, 写请求被执行(行⑮~⑯). 如果请求的映射项既不在 RCMT 中又不在 WCMT 中, 则 OAFTL 会检查 WCMT 是否已满, 若 WCMT 已满, OAFTL 将会选择 WCMT 中一组的映射项并把它们写入到闪存中(行⑰~⑱), 此处映射项的选择也是依照 LRU 算法. 之后, OAFTL 把数据写入到一个新的页中, 并且根据逻辑地址与物理地址生成一个新的映射项, 添加到 WCMT 中(行⑲~⑳).

WCMT 中的缓冲策略: 为避免映射页的频繁更新, OAFTL 为映射页提供一个日志页用于缓冲频繁更新的映射项. 算法 2 描述了如何在 WCMT 中选择牺牲项. 首先, 选择 WCMT 中最近最少使用的映射项(行①~②). 其次, 检查最近最少使用项所在的映射页是否有日志页(行③), 如果有日志页, OAFTL 直接更新逻辑物理转换页信息, 并把映射项驱逐出 WCMT(行④~⑦). 如果没有日志页, OAFTL 将会为该逻辑物理转换页分配一个日志页, 并把更新的映射项信息写入日志页中(行⑧~⑩). 执行完上述操作之后, OAFTL 更新 GMD, 并且把牺牲项淘汰出 WCMT(行⑪~⑫).

2.3 比较和分析

前面讲述了 OAFTL 的主要思想, 下面我们将从块利用率、垃圾回收、随机写操作性能等方面进行定性分析和比较.

1) 块利用率. 在混合级地址映射算法中, 闪存上的块分为数据块和日志块, 已有数据的更新操作只能被日志块吸收. 如果上层文件系统只频繁更新一部分数据, 则即使数据块中尚有很多空闲页, 仍然会触发日志页与数据页的合并操作, 此时闪存块的利用率并不高. 而对于页级地址映射算法来说, 更新操作可被闪存上所有的空闲数据页吸收, 提高了闪存的空间利用率. 因此, 基于页级地址映射的 OAFTL 算法能够提供较高的闪存空间利用率.

2) 垃圾回收. 一般的混合级地址映射算法进行

垃圾回收时, 需要将日志块与数据块进行合并操作, 进行一次合并操作需要把有效数据写入新块, 擦除日志块以及相关的数据块, 此时需要进行 $N(N \geq 1)$ 次块擦除操作才能产生一个新块. 而对于页级地址映射算法来说, 选择一个包含有效页最少的块, 拷贝有效页到新块, 擦除原始块, 即只需 $N=1$ 次擦除操作便产生一个新块, 降低了垃圾回收的代价. 因此, 基于页级地址映射的 OAFTL 算法的垃圾回收代价小于混合级地址映射算法.

3) 随机写操作性能. 闪存的随机写操作性能较差, 一般的思路是把随机写转换成顺序写. 对于混合级映射算法来说, 大量的随机写会导致一个日志块中吸收多个数据块的更新页, 进行垃圾回收时会带来昂贵的合并操作. 而在页级地址映射算法中就不会出现上述问题, 随机写以添加的方式写入闪存中的空闲数据页中, 从而大大提高了随机写操作的性能. OAFTL 算法充分利用了这一点, 把随机写转化为顺序写, 只需要更新映射项信息, 提升了随机写性能.

4) OAFTL 与 DFTL 对比: OAFTL 与 DFTL 都是基于页级地址映射的算法, 其主要不同之处是 OAFTL 把 RAM 中的映射表按照请求的操作类型分别组织, 这样可以更加迅速地定位到请求的地址转换信息, 减少查找逻辑地址映射项带来的额外代价, 提高整体的读写性能. 另外, OAFTL 利用了日志的思想, 为每个映射页提供一个日志页, 能够有效地缓解频繁更新映射项所带来的额外代价. 即某个映射页中的映射信息被部分更新时, 先分配一个空闲页保存更新的信息, 也就是上面所述的日志页, 这样可以避免更新原始页引发块擦除操作所带来的额外负担. OAFTL 通过牺牲空间以达到提高效率的目的, 总体上提高了闪存读写性能. 可以说, DFTL 算法是一种改进的页级地址映射算法, 而 OAFTL 算法又改进了 DFTL 算法.

3 实验结果与分析

本节我们将通过一系列的实验来验证 OAFTL 对企业级工作负载的有效性. 此处的实验中, 我们将根据企业级应用的特点, 选择 3 个典型的数据模型来生成数据集, 并把 OAFTL 算法同 FAST, DFTL 以及理想的页级地址映射算法作对比.

3.1 实验环境

本实验中使用的 PC 配置为: Intel Core 2 Pentium

4 Duo CPU 2.83 GHz, 2 GB 内存. 操作系统为 Windows XP. 由于在闪存存储设备(如 SSD)中,其内部的 FTL 层对用户是不开放的,因此在实际的硬件上实现 FTL 层变得非常困难. 为了比较不同 FTL 算法的性能,此处我们选择使用闪存模拟器^[17]进行实验. 在该模拟器中实现了 FAST 算法、DFTL 算法、理想的页级地址映射算法以及本文提出的 OAFTL 算法,所有的 FTL 算法都是用 C++ 语言实现. 在该实验中,闪存配置如下:容量 16 GB,页大小 2 KB,每个块中包含 64 个页. 由于在实际的 SSD 中, RAM 大小对于用户来说是保密的,此处我们设定 RAM 大小恰好满足 FAST 算法的要求.

实验中我们使用了 3 种典型的数据模型和测试标准,根据测试的数据模型生成数据集,用以测试不同的 FTL 算法对企业级应用的性能影响. 表 2 列出了实验中所使用的测试集的主要数据特征. OLTP 应用基本特征是顾客的原始数据可以立即传送到计算中心进行处理^[18],并在很短的时间内给出处理结果,因此在实时事务处理过程中会涉及很多小的随机写操作,测试该数据集的响应时间能够很好地衡量系统性能. 在实际应用环境中,很多统计数据都符合正态分布模型,因此,我们根据正态分布模型生成数据集. 上层请求访问的逻辑地址符合正态分布模型,其中正态分布中的概率密度函数如式(1)所示:

$$f(x; \mu, \sigma) = \frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}, x \in \mathbb{R}. \quad (1)$$

TPC-H 是指商业智能计算测试,它是 TPC 的重要测试表准之一,主要用来模拟真实商业的应用环境. 商业智能计算测试是对现实中商用计算需求的全面模拟,包括模拟真实商业交易数据库的动态查询. 因此根据 TPC-H 模型生成的测试数据集能够很好地反映商业应用中的 IO 流操作.

Table 2 Characteristics of Enterprise Test Data Set

表 2 企业级测试集特征

Data Set	Average Request Size/KB	Read Request/%	Sequential Request/%
OLTP	2	10	2
GD	2	20	10
TPC-H	2	90	20

实验中,我们把 OAFTL 算法同 FAST, DFTL 以及理想的页级地址映射算法作对比. 其中, FAST 算法是一种基于混合级地址映射的 FTL 算法; DFTL 是一种基于页级地址映射的 FTL 算法; 理想的页级地址映射算法把所有的映射项信息都存储于

RAM 中,在该情况下,我们假定 RAM 足够大. 然而,在实际的闪存存储设备中, RAM 是一种昂贵的资源,容量很小,不足以存储所有的页级地址映射信息. 因此,在 OAFTL, DFTL 以及 FAST 算法中,我们假定 RAM 大小只满足 FAST 算法所需的内存. 本实验主要通过运行不同工作负载的 IO 流来评估各个 FTL 算法的性能.

我们选择如下标准来评价 FTL 算法的性能: 闪存的擦除次数、数据集的运行时间. 这是因为闪存中擦除操作的代价是非常昂贵的,通常用擦除次数的多少来评判一个 FTL 算法性能的优劣,擦除次数越少其访问性能就会越好,对闪存的磨损度越少,闪存使用寿命就会越长. 另外,我们还通过调整 RAM 大小来观察 RAM 对基于页级地址映射的 FTL 算法的性能影响.

3.2 性能与分析

1) 运行时间比较. 图 3 展示了 FAST, DFTL、理想的页级地址映射算法和本文提出的 OAFTL 方法在闪存上运行 3 种不同数据集的总运行时间. 如前所述,我们为理想的页级地址映射算法提供足够的 RAM 大小以存储所有的映射项信息,因此,理想的页级地址映射算法的响应时间是其余算法的一个比较基准. 如图 3 所示, FAST 算法的响应时间最长,这是因为我们测试的应用中数据随机性较大,该算法对于处理随机访问操作性能表现并不理想. DFTL 算法与 OAFTL 算法的性能要略优于其他算法,而本文提出的 OAFTL 算法的响应时间又远少于 DFTL. 这是因为 OAFTL 算法通过识别访问类型来组织 RAM 中的映射表信息,能够缩短读取地址转换信息的时间,并且 OAFTL 对映射页的缓冲机制也能够提高系统的性能.

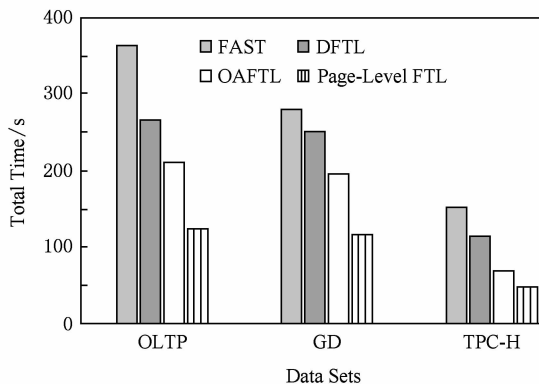


Fig. 3 Total time of different FTL algorithms running on three kinds of data set.

图 3 不同的 FTL 算法运行 3 种数据集的运行时间对比

2) 擦除次数比较. 图 4 展示了 FAST, DFTL、理想的页级地址映射算法和本文提出的 OAFTL 算法在不同数据集下的闪存擦除次数的比较. FAST 算法中, 通过日志块吸收数据的更新操作, 而日志区只占闪存很少的一部分, 当日志空间不够时就会触发擦除操作. 在我们测试的 3 种数据集中, 随机操作所占的比率较大, 由图 4 可见, FAST 算法引发的擦除操作最多, 这与数据集运行时间是相互对应的. 理想的页级地址映射算法是所有基于页级地址映射的算法的基准, 实验为页级地址映射算法提供足够大小的 RAM. 对于两种改进的基于页级地址映射的 FTL 算法: OAFTL 和 DFTL, 从实验结果可以看出, OAFTL 算法的擦除次数比 DFTL 算法要少, 这主要是因为 OAFTL 算法为每个映射页提供一个日志页, 用于缓冲频繁修改的映射项信息, 从而极大地减少了擦除操作的次数.

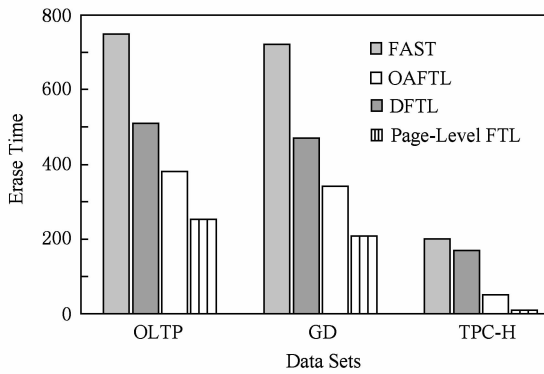


Fig. 4 Erase times of different FTL algorithms running on three kinds of data set.

图 4 不同的 FTL 算法运行 3 种数据集闪存擦除次数对比

3) RAM 大小对基于页级地址映射的 FTL 算法的影响. 基于块级地址映射和混合级地址映射的 FTL 算法都把逻辑到物理地址的映射信息存储在 RAM 中, 实验中我们设定 RAM 大小可容纳 FAST 算法所需的映射项信息, 因此, 扩大 RAM 对于 FAST 算法来说并不会影响它的地址信息转换性能. 此处我们只讨论 RAM 大小对基于页级地址映射的 FTL 算法的性能影响, 在不同 RAM 大小的情况下运行 OLTP 数据集的结果. 如图 5 所示, 随着 RAM 大小的增加, 两种算法的读响应时间都是不断减少, 这与理论推导是一致的. RAM 越大可以缓冲的映射项越多, 上层读操作请求的逻辑地址在 RAM 中的命中率就越大, 因此可以缩短读操作响应时间. 而 OAFTL 算法的读性能又高于 DFTL 算法, 当 RAM 大小等于 512 KB 时, OAFTL 的读性

能是 DFTL 算法的 30% 左右. 随着 RAM 的增大, 两种算法都趋向于理想的页级地址映射算法, 读响应时间不断缩短. 图 6 展示了 OAFTL 与 DFTL 算法写操作响应时间随 RAM 改变的变化图. 随着 RAM 的增大, 两种页级地址映射算法的响应时间越来越短. RAM 为 512 KB 时, OAFTL 算法的读性能要比 DFTL 算法好 15%. 随着 RAM 的增大, 两种算法都趋向于理想的页级地址映射算法, 写操作性能不断提升.

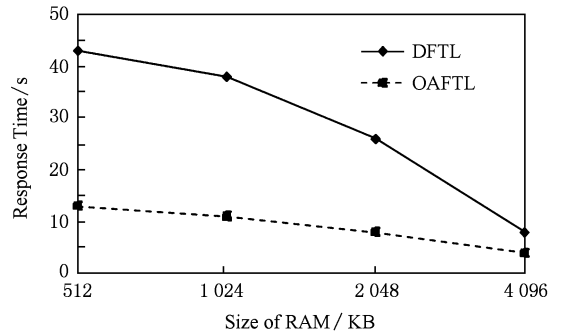


Fig. 5 Response time of read operations for OAFTL and DFTL.

图 5 OAFTL 与 DFTL 算法读操作响应时间对比

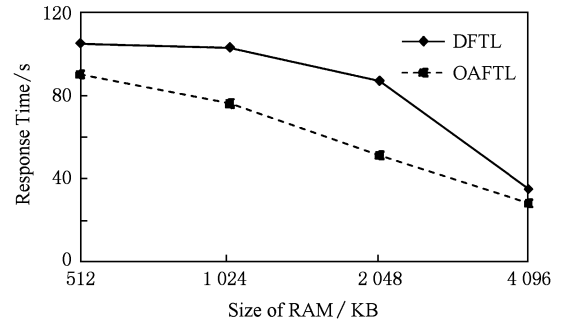


Fig. 6 Response time of write operations for OAFTL and DFTL.

图 6 OAFTL 与 DFTL 算法写操作响应时间对比

4 结 论

本文提出了一种高效的基于页级地址映射的闪存转换层策略. 该算法基于页级地址映射策略, 有效提高了闪存空间的利用率. 把频繁使用的映射信息缓冲在 RAM 中, 提出根据访问操作类型的不同分为读缓冲映射表和写缓冲映射表, 用于分别处理读操作和写操作, 大大减少了读取映射信息的代价, 提高读写操作的响应时间. 提出为存储映射信息的页提供了一个日志页, 这样可以缓解映射信息的频繁更新. 在实验部分, 我们使用 3 种典型的企业级应用

测试集来验证算法的有效性,结果显示,我们的算法性能要大大优于已有算法.例如在 OLTP 数据集下,OAFTL 算法的读写性能较 FAST 算法提升了 48%,比 DFTL 算法提升了 24%.

参 考 文 献

- [1] SamSung. Datacenter SSDs: Solid footing for growth [EB/OL]. [2011-06-21]. http://www.samsung.com/global/business/semiconductor/products/SSD/downloads/datacenter_ssds.pdf
- [2] Zhou Da, Liang Zhichao, Meng Xiaofeng. HF-Tree: An update efficient index for flash memory [J]. Journal of Computer Research and Development, 2010, 47(5): 832-840 (in Chinese)
(周大, 梁智超, 孟小峰. 一种闪存数据库的高更新性能索引结构[J]. 计算机研究与发展, 2010, 47(5): 832-840)
- [3] Jim G. A radical view of flash disks [EB/OL]. [2011-06-21]. http://research.microsoft.com/~Gray/talks/Flash_Is_Good.ppt
- [4] Amir B. Flash file system: United States, US005404485A [P]. 1995-04-04
- [5] Gal E, Toledo S. Algorithms and data structures for flash memories [J]. ACM Computing Survey, 2005, 37(2): 138-163
- [6] Kim J, Kim J M, Noh S H, et al. A space-efficient flash translation layer for compactflash systems [J]. IEEE Transactions on Consumer Electronics, 2002, 48(2): 366-375
- [7] Lee S, Park D, Chung T, et al. A log buffer based flash translation layer using fully associative sector translation [J]. ACM Trans on Embedded Computing Systems, 2007, 6(3): 18-es
- [8] Kang J, Jo H, Kim J, et al. A superblock-based flash translation layer for NAND flash memory [C] //Proc of the 6th ACM & IEEE Int Conf on Embedded Software. New York: ACM, 2006: 161-170
- [9] Lee S, Shin D, Kim Y, et al. LAST: Locality-aware sector translation for NAND flash memory-based storage systems [J]. ACM SIGOPS Operating System Review, 2008, 42(6): 36-42
- [10] Gupta A, Kim Y, Urgaonkar B. DFTL: A flash translation layer employing demand-based selective caching of page-level address mapping [C] //Proc of the 14th Int Conf on Architectural Support for Programming Language and Operating Systems. New York: ACM, 2009
- [11] Lee S, Moon B. Design of flash-based DBMS: An in-page logging approach [C] //Proc of the ACM SIGMOD Int Conf on Management of Data. New York: ACM, 2007: 55-66
- [12] Lim S, Lee S, Moon B. FASTER FTL for enterprise-class flash memory SSDs [C] //Proc of the 2010 Int Workshop on Storage Network Architecture and Parallel I/Os. Los Alamitos, CA: IEEE Computer Society, 2010

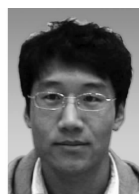
- [13] Samsung Electronics. 1G × 8Bit/2G × 8Bit NAND Flash Memory, Version 1.1 [EB/OL]. (2006-07-18) [2011-06-21]. http://datasheet.eeworld.com.cn/pdf/138612_SAMSUNG_K9WAG08U1A.html
- [14] Intel Corporation. Understanding the flash translation layer (FTL) specification [EB/OL]. (1998-12-01) [2011-06-21]. <http://www.embeddedfreebsd.org/Documents/Intel-FTL.pdf>
- [15] Chung T, Park D, Park S, et al. System software for flash memory: A survey [G] //LNCS 4096; Proc of the Int Conf on Embedded and Ubiquitous Computing. Berlin: Springer, 2006: 394-404
- [16] Chung T, Park D, Park S, et al. A Survey of flash translation layer [J]. Journal of Systems Architecture: the Euromicro Journal, 2009, 55(5/6): 332-343
- [17] Jin Peiquan, Su Xuan, Li Zhi, et al. A flexible simulation environment for flash-aware algorithms [C] //Proc of the 18th ACM Conf on Information and Knowledge Management. New York: ACM, 2009: 2093-2094
- [18] Intel Corporation. OLTP performance comparison: Solid state drives vs. hard disk drives [EB/OL]. [2011-06-21]. http://download.intel.com/design/flash/nand/extreme/OLTP_Performance_Comparison_Solid-State.pdf



Qi Xiaoying, born in 1987. MSc candidate. Student member of China Computer Federation. Her main research interests include storage management of flash-based database systems.



Tang Xian, born in 1978. PhD candidate. Her major research interests is in flash database system(txianz@gmail.com).



Liang Zhichao, born in 1986. MSc candidate. Student member of China Computer Federation. His current interest include query processing of flash-based database systems (frankey0207@gmail.com).



Meng Xiaofeng, born in 1964. Professor and PhD supervisor. Senior member of China Computer Federation. His research interests include Web data management, cloud data management, mobile data management, XML data management, flash-aware DBMS, privacy protection(xfmeng2006@gmail.com).