doi:10.3969/j.issn.1000-436x.2013.06.009

基于页面写相关的闪存转换层策略

陈金忠，姚念民，蔡绍滨，战福瑞，孙美玲
（哈尔滨工程大学 计算机科学与技术学院，黑龙江 哈尔滨 150001）

摘 要：针对固态硬盘(SSD)的闪存转换层(FTL)策略诸如 BAST、FAST 增加了垃圾回收的成本，带来了固态硬盘性能的下降等缺点，提出了一种基于页面“写相关”的 FTL 策略 PWRST。PWRST 的基本思想是分析 I/O 请求的访问历史并找出“写相关”的页面，将“写相关”的页面存储到同一数据块。从而减少垃圾回收开销和 I/O 请求的平均响应时间。实验结果表明 PWRST 在 Postmark 和 IOzone 负载下的响应时间比 BAST 减少了 35%，比 FAST 减少了 26%。在 TPC-C 负载下的响应时间比 BAST 减少了 12%，比 FAST 减少了 10%。

关键词：固态硬盘；闪存转换层；垃圾回收；写相关
中图分类号：TP333.35 **文献标识码**：A **文章编号**：1000-436X(2013)06-0076-09

Page write-related scheme for flash translation layer

CHEN Jin-zhong, YAO Nian-min, CAI Shao-bin, ZHAN Fu-rui, SUN Mei-ling
(College of Computer Science and Technology, Harbin Engineering University, Harbin 150001, China)

Abstract: Most existing flash translation layer (FTL) schemes such as block associative sector translation (BAST) and fully associative sector translation (FAST) increase garbage collection overhead and degrade the performance of solid-state drive (SSD). A novel flash translation layer scheme called page write-related sector translation (PWRST) was proposed. PWRST finds write-related pages from request history set and writes these pages into the same data block, which reduces garbage collection overhead and average request response time. Experiment results indicate that PWRST reduces average response time by 35% compared to BAST and 26% compared to FAST for Postmark and IOzone workload. For TPC-C workload, PWRST reduces response time by 12% compared to BAST and 10% compared to FAST.

Key words: solid state drive; flash translation layer; garbage collection; write-related

1 引言

过去几十年，机械硬盘的机械特性带来了随机读写的低性能和高功耗[1~3]。近年来，闪存的体积小、低功耗和抗震性等特点，使得闪存技术被广泛应用于 MP3、PDA 和移动电话[4]。随着固态硬盘价格逐年降低，不久的将来，固态硬盘将成为企业级存储系统的重要组成部分[5]。与机械硬盘不同，固态硬盘是基于半导体芯片，主要包括 2 种类型的闪存芯片、NOR 和 NAND[6]。NOR 类似于内存，容量小，它允许通过地址访问任何的一个闪存单元，数据的写入和擦除速度慢。NAND 容量大，数据的写入和擦除的速度较快，但需要经过 I/O 地址转换才能访问。因此，NAND 更适用于固态硬盘。大多数的硬盘厂商都支持和传统硬盘相同的接口，例如 SATA 和 SCSI 接口。虽然固态硬盘的读写速度比传统的机械硬盘快，但无论是基于 NOR 或 NAND 类型的固态硬盘都不支持“就地更新”，在更新某一块上的数据页之前，必须先把整个数据块擦除，称为“写前擦除”。相对读写操作，块擦除的代价非常高。为了避免这种昂贵的擦除开销，固态硬盘引入闪存转换层(FTL)。FTL 通过预留一些空闲的日

收稿日期：2012-10-11；**修回日期**：2013-01-20
基金项目：国家自然科学基金资助项目（61073047）；中央高校基本科研业务费专项基金资助项目（HEUCFT1007, HEUCF100607, HEUCFT1202）；哈尔滨市科技创新人才专项基金资助项目（2012RFLXG023）
Foundation Items: The National Natural Science Foundation of China (61073047); Fundamental Research Funds for the Central Universities (HEUCFT1007, HEUCF100607, HEUCFT1202); Harbin Special Funds for Technological Innovation Talents (2012RFLXG023)

志块，把需要更新的数据写入空闲日志块，从而避免了“写前擦除”。

经典的 FTL 策略有 2 种，分别是基于块相联的 BAST[7]策略和基于全相联日志缓存的 FAST[8]策略。BAST 为每个数据块指定唯一的日志块，更新的数据只能写入对应的日志块。BAST 适合顺序存取。对于随机存取，BAST 会造成存储空间的浪费。FAST 采用全相联的映像方式，更新的数据可以写入到任何的日志块，不会造成存储空间的浪费。FAST 还将日志块划分为顺序缓存日志块与随机缓存日志块，分别用于优化顺序存取与随机存取。然而，在 FAST 策略中，一个日志块可能关联多个数据块，增加了垃圾回收的成本。为了克服 BAST、FAST 等策略的缺点，本文提出了一种基于页面“写相关”的 FTL 策略 PWRST。PWRST 统计 I/O 请求的历史信息，将“写相关”的页面存储在同一数据块。PWRST 减少了页面复制开销与块的擦除成本，降低了垃圾回收的成本，提高了固态硬盘的性能。

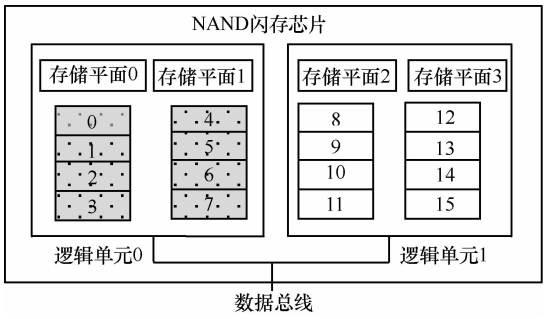
2　相关工作

2.1　NAND 内部结构

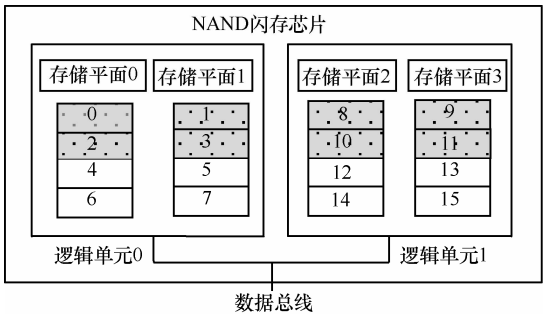
NAND 闪存芯片由若干个逻辑单元组成，每个逻辑单元包括多个存储平面，每个存储平面由多个存储块构成。假设每个闪存芯片由 2 个逻辑单元组成，每个逻辑单元由 2 个存储平面构成，每个存储平面由 4 个存储块构成。如图 1 所示，按照存储块在每个存储平面上的分布方式，将固态硬盘分为 3 种不同的架构，分别是无条带化架构、部分条带化架构和全条带化架构。图 1(a)显示无条带化架构的 NAND 闪存，将存储块按顺序存放在每个存储平面上，这种结构不支持并行读写。图 1(b)显示了部分条带化架构的 NAND 闪存。这种结构的特点是将存储块分散存放在成对存储平面上，同一个逻辑单元内部的 2 个存储平面可以并行读写。全条带化架构的 NAND 闪存如图 1(c)所示。存储块分散存放在每一个存储平面上，2 个逻辑单元可以并行读写。

2.2　NAND 闪存转换层

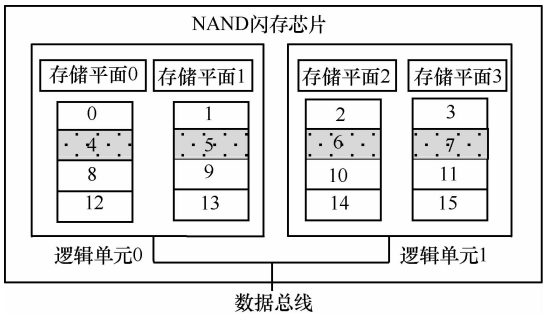
由于 NAND 闪存在更新存储页时必须擦除整个存储块，这大大地降低了 NAND 的性能。FTL 通过预留一些日志块，将更新的数据写入空闲日志块，从而避免擦除整个数据块。闪存转换层主要功能如下。



(a) 无条带化



(b) 部分条带化



(c) 全条带化

图 1　SSD 的 3 种不同的内部架构

1) 映射机制(mapping mechanism)：按照映射的粒度，分为页面级别映射与块级别映射。页面级别映射的优点是容易实现且高效，缺点是需要大量的空间存储页面映射表。块级别映射节省了存储映射表的空间，但在垃圾回收时需要大量的页面复制操作，增加了垃圾回收的开销。许多 FTL 策略采用混合映射机制，大量的数据块采用块级别映射，少量的日志块采用页面级别映射[9,10]。这种混合映射机制提高了固态硬盘的性能。

2) 垃圾回收(garbage collection)：当固态硬盘的空闲块数量少于某个下限值时(通常小于总块数的 15%)，垃圾回收算法扫描整个固态硬盘以回收存储块。文献[11,12]通过减少垃圾回收过程中擦除块数来延长固态硬盘的使用寿命。

3) 损耗均衡(wear-leveling)：由于程序的局部

性原理，固态硬盘的某些存储块会被频繁的更新和擦除，导致这些块的损耗加快。闪层转换层通常采用“冷块”与“热块”的交换的方法来达到所有存储块损耗均衡的目的。文献[13]提出了 FeGC 机制，优先回收失效页数量多和失效时间长的数据块，达到了减少垃圾回收开销和损耗均衡的目的。针对闪存阵列，文献[14]提出了 FAP 策略，通过交换擦除次数多和擦除次数少的块，达到 NAND 闪存芯片之间的损耗均衡的目标。

2.3　块相联(BAST)策略

BAST 的基本思想是每一个数据块关联唯一的日志块，更新时，页面写入相应的日志块。如果日志块没有空闲的页面，即使其他的日志块有空闲页面，也不能写入。假设 LPN 表示 I/O 请求的逻辑页号，data 表示写入的数据，每个块的大小为 S。写操作定义为 write(LPN, data)。在写操作之前，需要将逻辑页号转换成逻辑块号(LBN)，计算公式为 $LBN=LPN \bmod S$。然后，根据 LBN 查找逻辑块的物理块号(PBN)，最后，将数据 data 写入数据块 PBN。假设每个存储块包括 4 个页面，日志块数为 2。I/O 请求的逻辑页面向量为 $\boldsymbol{V}$={0,1,2,3,4,5,6,7, 2,3,2,3,7,1,1,2}。逻辑块 0 对应数据块 0，为数据块 0 分配日志块 0。逻辑块 1 对应数据块 1，为数据块 1 分配日志块 1。BAST 的操作流程如图 2 所示。由于页 0、1、2、3 的逻辑块号为 0，写入对应的数据块 0。页 4、5、6、7 的逻辑块号为 1，写入对应的数据块 1。当更新逻辑页 2、3、2、3 时，写入对应的日志块 0。当更新逻辑页 7、1、1、2，逻辑页 7 写入到日志块 1。此时数据块 0 对应的日志块已经写满，逻辑页 1、1、2 不能写入日志块 0，需要执行擦除操作。从图 2 可以看出，日志块 1 还有空闲的页面，但不能将数据块 0 的页面写入日志块 1。BAST 造成了日志块空间的浪费，其主要原因是在 BAST 策略中，一个数据块只关联唯一的日志块，更新的页面只能写入对应的日志块。

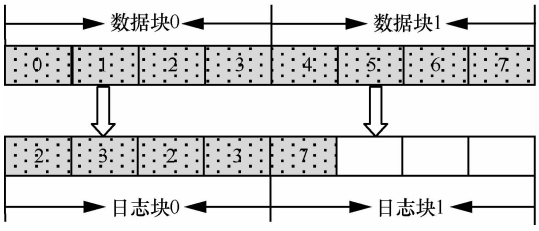


图 2　BAST 策略写操作流程

2.4　全相联(FAST)策略

FAST 策略的基本思想是数据块可以共享日志块。更新的数据可以写入任何日志块的空闲页面。FAST 策略的操作流程如图 3 所示。FAST 策略更新逻辑页 1、1、2 时，由于日志块 1 还有空闲页，把逻辑页 1、1、2 写入日志块 1。FAST 策略充分利用存储空间，减少不必要的擦除开销和提高日志块的空间利用率。如图 3 所示，日志块 1 的逻辑页 7 对应数据块 1，逻辑页 1、1、2 对应数据块 0。如果回收日志 1，需要将日志块 1 与数据块 0，数据块 1 合并，进行多次页面复制和块擦除。这是 FAST 的主要缺点。其原因是在 FAST 策略中，一个日志块关联了多个数据块。因此，FAST 策略虽然克服了 BAST 策略的缺点，但也带来了昂贵的页面复制与块擦除开销，从而降低了固态硬盘的性能和缩短了固态硬盘的使用寿命。在 2.5 节将具体分析 3 种垃圾回收的开销。

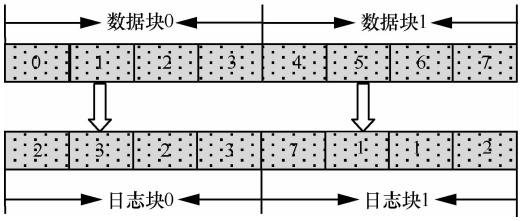


图 3　FAST 策略写操作流程

2.5　垃圾回收开销的分析

垃圾回收分为 3 种形式，全合并(full-merge)回收、部分合并(partial-merge)回收和交换合并(switch-merge)回收。如图 4(a)所示，全合并回收需要复制大量的有效页(valid)和多次块擦除(erase)。如图 4(b)所示，部分合并回收需要复制少量的有效页和 2 次块擦除。如图 4(c)所示，交换合并回收不需要复制页面，仅需 2 次块擦除。因此，应该尽可能地增加部分合并回收与交换回收的次数，减少全合并回收的次数。假设 C_c 表示复制一页的开销，C_e 表示擦除数据块的开销，N 表示数据块包含页的数量，N_v 表示数据块包含有效页的数量，N_a 表示日志块所关联的数据块的数量，W_f、W_p 和 W_s 分别表示全合并回收，部分合并回收和交换合并回收的开销，则有

$$W_f = N_a \times \{(NC_c) + C_e\} + C_e \tag{1}$$

$$W_p = N_v C_c + 2C_e \tag{2}$$

$$W_s = 2C_e \tag{3}$$

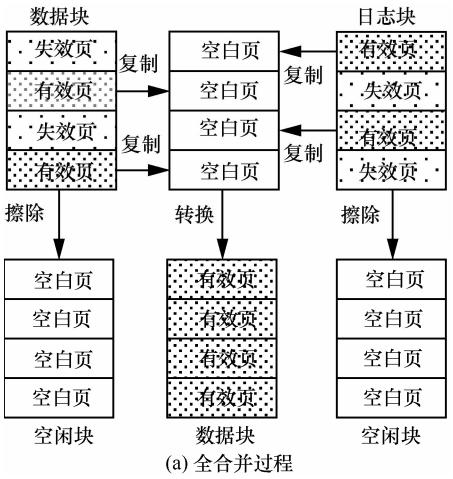


图 4　3 种全合并回收过程

由式(1)~式(3)可知，在全合并回收时，日志块所关联的数据块数量 $N_a>1$，在部分合并和交换合并时，日志块关联的数据块数量 $N_a=1$。因此，减少垃圾回收开销的关键是减小 N_a。由 2.4 节可知，FAST 的一个日志块可能关联多个数据块，即 $N_a>1$。在垃圾回收时，需要复制大量的页面和多次擦除块，使得固态硬盘性能的下降。

3　PWRST 策略

针对 BAST 和 FAST 的缺点，PWRST 根据 I/O 请求的历史特性，将“写相关”的页面存储到同一数据块。所谓“写相关”是指 I/O 请求访问页面 *A* 之后，在最近一段时间内将会访问页面 *B*，称页面 *A* 与页面 *B* 是“写相关”。例如，文件系统在更新文件时，对应的元数据也会被更新，这 2 次操作的页面是“写相关”。如果数据块的页面都是“写相关”，这些页面可能都是失效的，在垃圾回收时，只需要进行交换合并回收。

3.1　系统架构

SSD 的系统架构如图 5 所示。PWRST 分为 3 个模块，分别是 I/O 请求历史收集模块(RHC)、页面写相关分析模块(WRA)和页面聚类模块(PC)。首先，I/O 请求历史收集模块负责记录负载最近一段时间内访问的逻辑页号。其次，页面写相关分析模块根据 I/O 请求历史找出“写相关”的页面。最后，页面聚类模块将具有“写相关”的页面存储到同一数据块。

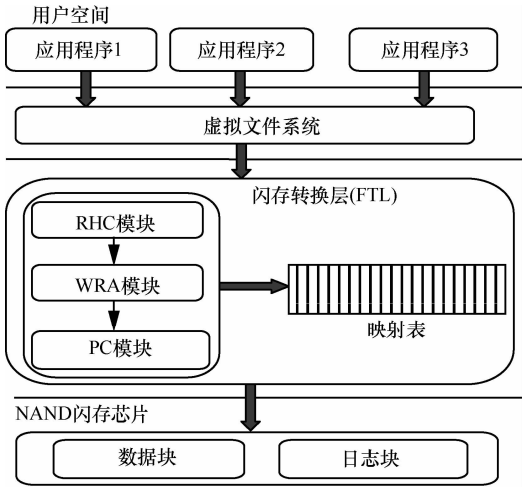


图 5　SSD 的系统架构

3.2　I/O 请求历史收集模块(RHC)

首先，文件系统发出的 I/O 请求将逻辑扇区(LBA)号映射为逻辑页号(LPN)，FTL 根据逻辑页号得到逻辑块号(LBN)和块内偏移(offset)，然后通过查询块映射表得到物理块号(PBN)。RHC 模块使用散列表对 I/O 请求的时刻 *T* 和访问的逻辑页号进行记录。其中，*T* 和 *LPN* 分别表示了 I/O 请求的时间和空间特性。散列表可以实现对 I/O 请求的快速查

找，更新以及删除操作，其维护非常方便。因此，散列表可以有效地维护 I/O 请求的历史信息。

3.3 页面写相关分析模块(WRA)

为了描述页面“写相关”，首先定义以下几个参数。

定义 1 (1 步转移概率) $P_{ij}^{(1)}$。在 t 时刻访问页面 LPN_i，并在 t+1 时刻访问页面 LPN_j 的概率称为 1 步转移概率。

定义 2 (n 步转移概率) P_{ij}^{n}。在 t 时刻访问页面 LPN_i，在 t+ n 时刻访问页面 LPN_j 的概率称为 n 步转移概率。

定义 3 (n 步之内转移概率) $P_{ij}^{<n}$。在 t 时刻访问页面 LPN_i，并在 t+j(j<n)时刻访问页面 LPN_j 的概率称为 n 步之内转移概率。

由上述定义可知

$$P_{ij}^{<n}=\sum_{k=1}^{n-1}P_{ij}^{(k)} \tag{4}$$

根据以上定义，下面给出几个定理。

定理 1 I/O 请求在 t 时刻访问页面 LPN_i，在 t+k(k<j<n)时刻访问页面 LPN_k，并最终在 t+ j(j<n)时刻访问页面 LPN_j 的概率可表示为

$$P_{ikj}^{<n}=\sum_{p=1}^{n-2}\sum_{q=1}^{p+q<n}P_{ik}^{(p)}P_{kj|ik}^{(q)} \tag{5}$$

其中，$P_{ik}^{(p)}$ 表示 p 步转移概率。

证明 I/O 请求在 t 时刻访问页面 LPN_i，在 t+k (k<j<n)时刻访问页面 LPN_k，并最终在 t+j(j<n)时刻访问页面 LPN_j 可分为以下几种情况。

1) I/O 请求在 t 时刻访问页面 LPN_i，并在 t+1 时刻访问页面 LPN_k，最终在 t+j(j<n)访问页面 LPN_j。

2) 以此类推，I/O 请求在 t 时刻访问页面 LPN_i，并在 t+j–1 时刻访问页面 LPN_k，最终在 t+ j(j<n)访问页面 LPN_j。

因此，$P_{ikj}^{<n}$ 等于所有上述情形的概率之和，即

$$\begin{aligned}P_{ikj}^{<n}&=P_{ik}^{(1)}\left(P_{kj|ik}^{(1)}+P_{kj|ik}^{(2)}+\cdots+P_{kj|ik}^{(n-2)}\right)+\\&\quad P_{ik}^{(2)}\left(P_{kj|ik}^{(1)}+P_{kj|ik}^{(2)}+\cdots+P_{kj|ik}^{(n-3)}\right)+\cdots+P_{ik}^{(n-2)}P_{kj|ik}^{(1)}\\&=\sum_{p=1}^{n-2}\sum_{q=1}^{p+q<n}P_{ik}^{(p)}P_{kj|ik}^{(q)}\end{aligned}$$

推论 1 I/O 请求在 t 时刻访问页面 LPN_i，在 $t+k_1(k_1<k_2)$ 时刻访问页面 LPN_{k_1}，在 $t+k_2$ ($k_2<j<n$) 时刻访问页面 LPN_{k_2}，并最终在 t+ j(j<n)时刻访问页面 LPN_j 的概率可表示为

$$P_{ik_1k_2j}^{<n}=P_{ik_1k_2}^{<n_1}P_{k_2j|ik_1k_2}^{<n_2}\ \ (n_1+n_2<n+1) \tag{6}$$

证明 可分为 2 个子步骤。

1) I/O 请求在 t 时刻访问页面 LPN_i，在 $t+k_1(k_1<k_2)$ 时刻访问页面 LPN_{k_1}，并在 $t+k_2$ 时刻访问页面 LPN_{k_2}。

2) I/O 请求在 $t+k_2(k_2<j<n)$ 时刻访问 LPN_{k_2}，最终在 t+ j(j<n) 时刻访问 LPN_j。

由定理 1 并结合式(5)得到

$$P_{ik_1k_2j}^{<n}=P_{ik_1k_2}^{<n_1}P_{k_2j|ik_1k_2}^{<n_2},n_1+n_2<n+1$$

即推论 1 成立。

定理 2 I/O 请求在 t 时刻访问页面 LPN_i，在 $t+k_1(k_1<k_2)$ 时刻访问页面 LPN_{k_1}，在 $t+k_2(k_2<k_3)$ 时刻访问页面 LPN_{k_2}，以此类推，在 $t+k_m(k_m<j<n)$ 时刻访问页面 LPN_{k_m}，并最终在 $t+j(j<n)$ 时刻访问页面 LPN_j 的概率可表示为

$$P_{i,k_1,k_2,\cdots,k_m,j}^{<n}=P_{ik_1k_2,\cdots,k_m}^{<n_1}P_{k_mj|i,k_1,k_2,\cdots,k_m}^{<n_2}\ (n_1+n_2<n+1)$$

证明 由定理 1 和推论 1，容易证明定理 2 成立。

定义 4 页面写相关。n 步之内归一化概率定义为 $p_{i,j}^{<n}=\left(P_{ij}^{<n}+P_{ji}^{<n}\right)/2$。若 $p_{i,j}^{<n}>\theta_{\text{th}}$，称页面 LPN_i 与页面 LPN_j 是写相关。如果多个页面 LPN_{i_1}，LPN_{i_2},…, LPN_{i_m} 的 n 步之内的归一化概率 $p_{i_1,i_2,\cdots,i_m}^{<n}=\sum P_{\vartheta(i_1,i_2,\cdots,i_m)}^{<n}/m!>\theta_{\text{th}}$。就称 LPN_{i_1}，LPN_{i_2},…, LPN_{i_m} 是写相关。其中，m 表示数据块包含的页数。$\vartheta(i_1,i_2,\cdots,i_m)$ 表示页面的全排列。θ_{th} 是门限值。根据不同的负载确定 θ_{th} 的值，将 θ_{th} 设置为所有页面的归一化概率的平均值，在实际的系统中可进行调节。

图 6 给出了一组 I/O 请求的逻辑页号，开始时刻是 T_0。假设 n=3，首先计算 2 个页面的归一化概率。例如计算 $p_{1,4}^{<3}$。由定义 4 和式(4)得

$$p_{1,4}^{<3}=\left(P_{14}^{<3}+P_{41}^{<3}\right)/2=\left(\sum_{k=1}^{2}P_{14}^{(k)}+\sum_{k=1}^{2}P_{41}^{(k)}\right)/2 \tag{7}$$

由图 6 可知，从访问页面 1 到访问页面 4 的情况有：1→3→2→4、1→4 和 1→3→4。所以，$P_{14}^{<3}$= 2/3。从访问页面 4 到访问页面 1 的情况有：4→1、4→5→1 和 4→1，即访问页面 4 之后，3 步之内必然访问页面 1。所以，$P_{41}^{<3}$ =1。最终，$p_{1,4}^{<3}$ =5/6。接

下来，计算多个页面的归一化概率，以 3 个页面为例，计算 $p_{1,3,4}^{<3}$，由定义 4 可得

$$p_{1,3,4}^{<3}=\left(P_{134}^{<3}+P_{143}^{<3}+P_{314}^{<3}+P_{341}^{<3}+P_{413}^{<3}+P_{431}^{<3}\right)/6 \quad (8)$$

由式(2)可计算式(8)的每一项，以 $P_{134}^{<3}$ 为例，即 $P_{134}^{<3}=P_{13}^{1}\times P_{34|13}^{1}=1/2\times1/2=1/4$。

同理，可计算得 $P_{143}^{<3}=0$，$P_{314}^{<3}=0$，$P_{341}^{<3}=1/2$ $P_{413}^{<3}=1/3$，$P_{431}^{<3}=0$。所以，$p_{1,3,4}^{<3}=13/72$。

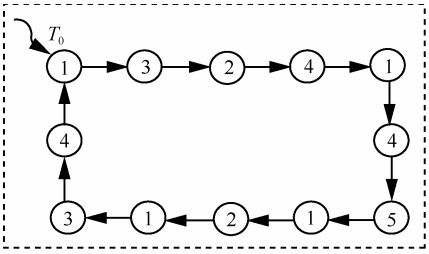


图 6　I/O 请求访问的页面序列

3.4　页面聚类模块

从 3.3 节可知，“写相关”的页面具有访问时间的关联性，如果将这些页面存储到 SSD 的同一数据块，那么在垃圾回收时，数据块包含的页面可能都是失效的页面，只需要一次擦除操作。从而避免了昂贵的页面复制开销，提高了垃圾回收的效率。页面聚类的模块功能是将“写相关”的页面存储到同一个数据块，其基本原理是利用了数据挖掘中的层次聚类的方法。即初始时将每个页面划分为一个单独的组，在接下来的迭代中，把那些“写相关”的页面合并成一个组，直到这个组的大小等于数据块的大小，然后将这些页面写入空闲块。图 7 给出了具体的操作流程。首先，从 I/O 请求的历史集合 RHC 找出访问频率最高的页面 p_i，放入组 A_i。然后，根据页面写相关分析模块，找出与页面 p_i “写相关”的页面 p_j，并将页面 p_j 放入 A_i，组中的页面数 c_i 加 1。如果 A_i 中页面的数量等于块的大小，聚类完成。否则，继续聚类。对于某些系统公共页面来说，可能有许多与其写相关系数较大的页面。这些公共页面可以使用页面映射的方式，那么在垃圾回收时，就不会引起全合并操作。

4　性能分析

实验分别测试了 BAST、FAST 和 PWRST 在 Postmark[15]、IOzone[16] 和 TPC-C[17]负载下的垃圾回收开销和 I/O 请求的响应时间。

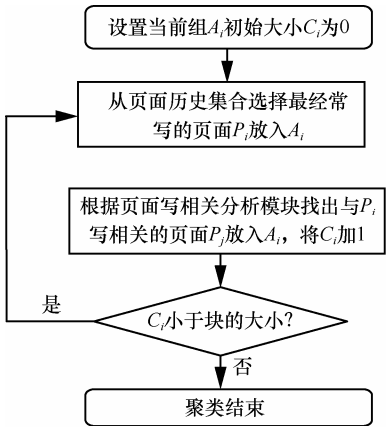


图 7　页面聚类算法

4.1　测试环境

本实验所使用的模拟器是建立 DiskSim 之上的 SSD 模拟器[18]。该模拟器提供了对垃圾回收机制与交叉读写机制的模拟。表 1 给出了 SSD 模拟器的配置参数，包括 SSD 基本组成部分、页面的读写时延、擦除块的时延、擦除模式等。

表 1　　SSD 模拟器配置参数

模拟器相关参数	值
每个固态硬盘的芯片数	8
每个芯片的存储平面数	8
每个存储平面的数据块数	2 048
每个块包含的页面数	64
芯片传输时延	0.000 025 ms
页面的读时延	0.025 ms
页面的写时延	0.2 ms
数据块的擦除时延	1.5 ms
控制命令开销	0 ms
是否使用交叉存取策略	是
是否显示擦除	是
存储平面中的块映射方式	全条带化架构

4.2　测试负载和参数设置

测试的负载包括 Postmark、IOzone 和 TPC-C。其中，Postmark 和 IOzone 是 I/O 密集型的文件系统测试集，包括大量的随机读写操作。IOzone 负载是对大文件的测试。Postmark 负载则是频繁、大量地存取小文件。TPC-C 负载是数据库测试集。模拟实验中的 2 个重要的参数：TH_s 和 θ_{th}。其中，TH_s 表示系统空闲块数占总存储块的比例。θ_{th} 表示“写相关”的门限值，分别设置为 0.5、0.7 和 0.9，对应的 PWRST 策略表示为 PWRST(0.5)、PWRST(0.7)和 PWRST(0.9)。

4.3 垃圾回收开销

SSD 垃圾回收主要包括两方面：页面迁移数和擦除块数；其中页面迁移数是指在全合并回收和部分合并回收过程中复制的页面数量。而擦除块数是指在全合并、部分合并和交换合并回收过程擦除的块数总和。SSD 模拟器分别对 Postmark、IOzone 和 TPC-C 3 种负载，生成 10^6 个读写请求进行测试。

4.3.1 页面迁移数

表 2~表 4 分别给出了 TH_s 的值为 0.15、0.10 和 0.05 时，3 种算法垃圾回收时的页面迁移数。PWRST(0.7)在 Postmark 和 IOzone 负载下的页面迁移数比 BAST 减少了 30%，比 FAST 减少了 20%。在 TPC-C 负载下，PWRST(0.7)页面迁移数比 BAST 减少 15%,比 FAST 减少了 12%。主要原因是 BAST 数据块对应唯一的日志块，当关联的日志块没有空间时，即使其他的日志块有空闲页时，也会触发垃圾回收机制。FAST 的日志块可能关联多个数据块，增加了全合并的次数，从而增加了页面迁移数。另外，3 种策略的页面迁移数都随着 TH_s 值的减小而增加，这是由于随着空闲块数目的减少，执行垃圾回收的概率增大。另外，PWRST(0.7)的页面迁移数量少于 PWRST(0.5)和 PWRST(0.9)。PWRST(0.9)只将归一化概率大于 0.9 的页面写入同一数据块，因而会遗漏一些页面。而 PWRST(0.5)可能将一些不相关的页面写入同一数据块，因此会增加页面迁移数。

4.3.2 块擦除次数

表 5~表 7 给出了 BAST、FAST 和 PWRST 在 3 种负载下的块擦除次数。BAST、FAST 和 PWRST 这 3 种策略块擦除次数随着 TH_s 的减小而增加。在 Postmark 和 IOzone 负载下，PWRST(0.7)的块擦除次数比 BAST 减少 10%,比 FAST 减少 8%。相比于 BAST 和 FAST，PWRST 利用页面“写相关”减少了全合并回收的次数和块的擦除次数。对于固态硬盘，擦除次数不仅与性能相关，而且影响固态硬盘的使用寿命。因此，PWRST 策略能够有效地提高固态硬盘的性能和延长固态硬盘的使用寿命。

表 2 当 TH_s=0.15 时，BAST、FAST 和 PWRST 的页面迁移数

算法	Postmark	IOzone	TPC-C
BAST	15 762	16 417	15 487
FAST	14 695	15 560	15 198
PWRST(0.5)	14 283	15 422	14 021
PWRST(0.7)	12 135	13 182	13 564
PWRST(0.9)	13 632	15 002	14 432

表 3 当 TH_s=0.10 时，BAST、FAST 和 PWRST 的页面迁移数

算法	Postmark	IOzone	TPC-C
BAST	18 503	20 596	18 321
FAST	16 528	19 011	17 843
PWRST(0.5)	15 432	18 342	16 093
PWRST(0.7)	14 053	15 843	15 932
PWRST(0.9)	16 032	18 325	16 912

表 4 当 TH_s=0.05 时，BAST、FAST 和 PWRST 的页面迁移数

算法	Postmark	IOzone	TPC-C
BAST	28 809	30 840	28 401
FAST	25 160	27 097	27 218
PWRST(0.5)	23 356	23 734	25 292
PWRST(0.7)	20 148	22 455	23 357
PWRST(0.9)	23 159	28 475	25 634

表 5 TH_s=0.15 时，3 种策略的块擦除数

算法	Postmark	IOzone	TPC-C
BAST	21 156	21 762	21 015
FAST	20 862	21 369	20 987
PWRST(0.5)	19 924	20 123	20 712
PWRST(0.7)	19 230	19 812	20 523
PWRST(0.9)	20 349	20 175	20 621

表 6 TH_s=0.10 时，3 种策略的块擦除数

算法	Postmark	IOzone	TPC-C
BAST	22 445	23 576	22 269
FAST	21 952	23 084	21 790
PWRST(0.5)	21 102	22 663	21 721
PWRST(0.7)	20 123	21 452	21 401
PWRST(0.9)	21 204	22 112	21 568

表 7 TH_s=0.05 时，3 种策略块的擦除数

算法	Postmark	IOzone	TPC-C
BAST	27 156	28 613	26 741
FAST	25 987	27 801	25 863
PWRST(0.5)	24 931	25 011	25 722
PWRST(0.7)	21 903	22 811	23 902
PWRST(0.9)	23 031	24 912	25 081

4.4 3 种合并回收的开销

图 8~图 10 给出了在 $TH_s = 0.05$ 时，全合并回收，部分合并回收与交换合并回收在 3 种负载下的开销。BAST 在 Postmark 负载下的全合并次数占 80%左右，FAST 的全合并回收次数占 50%左右，而 PWRST 全合并次数只占了 10%左右。FAST 充分利用了存储空间，

避免了不必要的擦除开销。因此，FAST 的全合并开销比 BAST 低。PWRST 将“写相关”的页面写入同一数据块，减少了日志块关联的数据块数量，增加了部分合并回收与交换合并回收的比例。因此，PWRST 垃圾回收的开销比 BAST 和 FAST 都小。

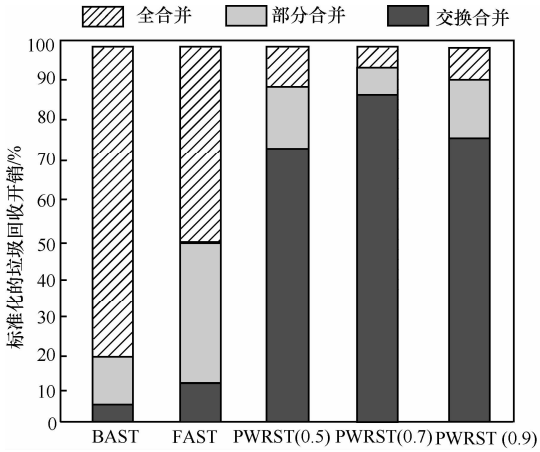


图 8　在 Postmark 负载下合并开销的比例

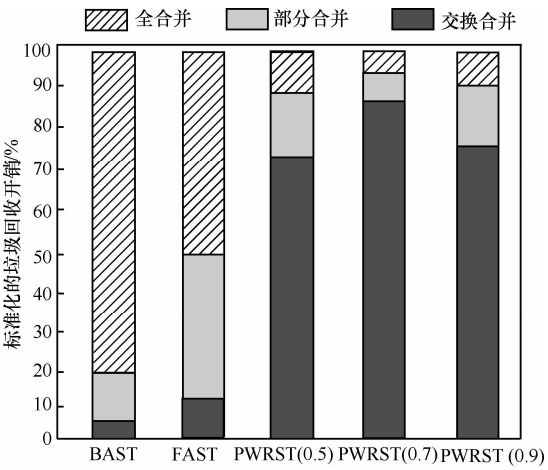


图 9　在 IOzone 负载下合并开销的比例

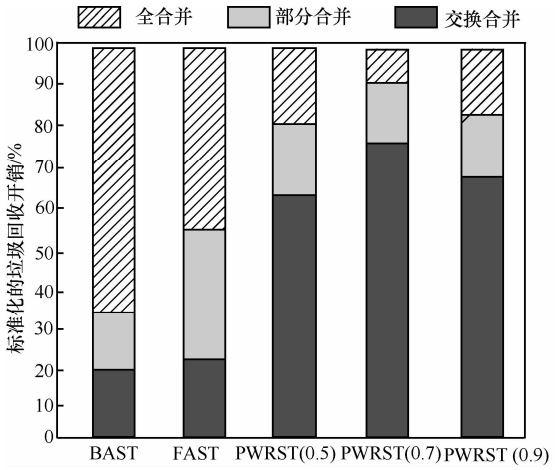


图 10　在 TPC-C 负载下合并开销的比例

4.5　不同 FTL 策略的 I/O 平均响应时间

图 11~图 13 给出了 3 种 FTL 策略在 Postmark、IOzone 和 TPC-C 3 种负载下的平均响应时间。当 $TH_s = 0.15$ 时，PWRST(0.7)在 Postmark 下的平均响应时间比 BAST 减少了 18%，比 FAST 减少了 14%。PWRST(0.7)在 TPC-C 下的平均响应时间比 BAST

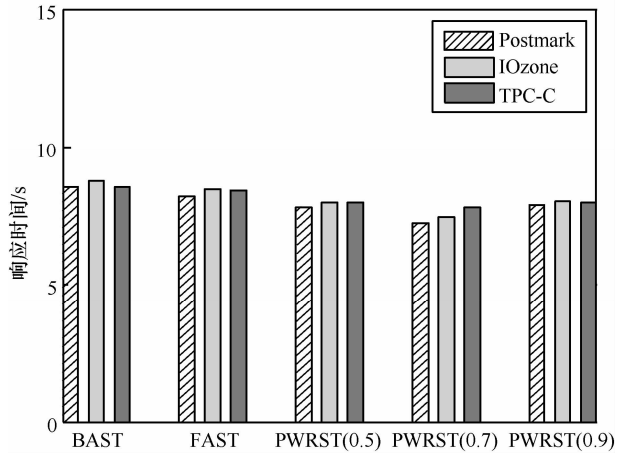


图 11　当 $TH_s = 0.15$ 时，BAST、FAST 和 PWRST 的平均响应时间

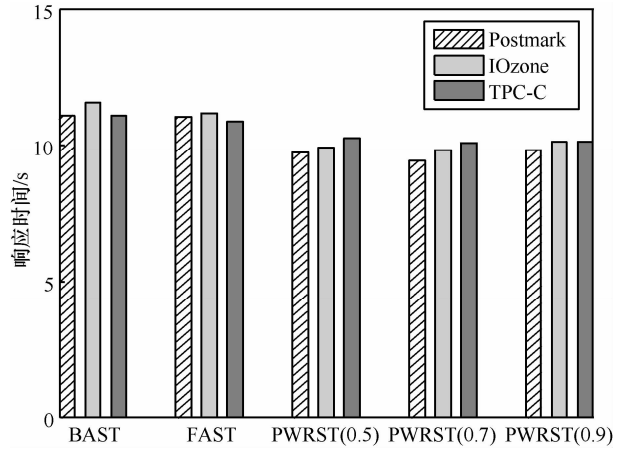


图 12　当 $TH_s = 0.10$ 时，BAST、FAST 和 PWRST 的平均响应时间

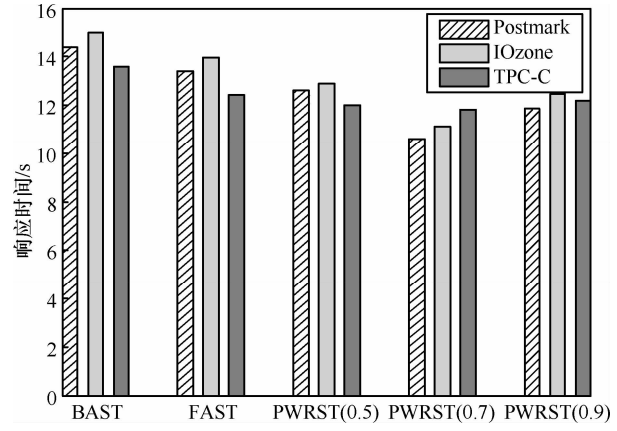


图 13　当 $TH_s = 0.05$ 时，BAST、FAST 和 PWRST 的平均响应时间

策略减少了 10%，比 FAST 减少了 8%。当 $TH_s = 0.05$ 时，PWRST(0.7)在 Postmark 下的平均响应时间比 BAST 减少 35%，比 FAST 减少 26%。在 TPC-C 负载下，PWRST(0.7)的平均响应时间比 BAST 减少了 12%,比 FAST 减少了 10%。因此，本文提出的 PWRST 策略在响应时间等方面优越于 BAST 和 FAST。

5 结束语

经典的闪存转换层策略（如 BAST、FAST）都有其自身的缺点，BAST 会造成存储空间的浪费和不必要的擦除开销。FAST 策略中的日志块可能关联多个数据块，垃圾回收的开销非常大。本文提出了一种基于页面写相关的 FTL 策略 PWRST。PWRST 将“写相关”的页面存储到同一数据块，减少了全合并回收的次数。实验表明，PWRST 减少了块的擦除次数，降低了垃圾回收的开销和 I/O 的平均响应时间。进一步验证了 PWRST 的可行性和优越性。

参考文献：

[1] CHUNG T S, PARK S W, LEE D H, *et al*. Systems software for flash memory: a survey[A]. Proceedings of the 2006 IFIP International Conference on Embedded and Ubiquitous Computing[C]. Korea, 2006. 394-404.

[2] DING X N, JIANG S, CHEN F, *et al*. DULO: an effective buffer cache management scheme to exploit both temporal and spatial localities[J]. ACM Trans Storage, 2007, 3(2):1242522.

[3] LI Z M, CHEN Z F, SUDARSHAN M S, *et al*. C-miner: mining block correlations in storage systems[A]. Proc of FAST'02[C]. San Francisco, USA, 2004. 173-186.

[4] GAL E, TOLEDO S. Algorithms and data structures for flash memories[J]. ACM Computing Surveys, 2005, 37(2):138-163.

[5] LEVENTHAL A. Flash storage memory communications[J]. Communications of the ACM, 2008, 51(7):47-51.

[6] SANTARINI M. NAND versus NOR[J]. EDN, 2005, 50(21):41-48.

[7] KIM J S, 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.

[8] LEE S W, PARK D J, CHUNG T S, *et al*. A log buffer based flash translation layer using fully associative sector translation[J]. IEEE Transactions on Embedded Computing Systems, 2007, 6(3):18-45.

[9] KANG J U, JO H, KIM J S, *et al*. A superblock based flash translation layer for NAND flash memory[A]. Proc of ICES'06[C]. Seoul, Korea, 2006. 161-170.

[10] CHAO H, SHIN D , EOM Y I. Kast: k-associative sector translation for NAND flash memory in real-time systems[A]. Design, Automation Test in Europe Conference Exhibition[C]. Nice, France, 2009. 507-512.

[11] CHEN F, LUO T, ZHANG X D. CAFTL: A content-aware flash translation layer enhancing the lifespan of flash memory based solid state drivers[A]. Proceedings of FAST'2011[C]. San Jose, CA, USA, 2011. 77-90.

[12] WU G Y, HE X B. ΔFTL: improving SSD lifetime via exploiting content locality[A]. 7th ACM European Conference on Computer Systems, EuroSys'12[C]. Bern, Switzerland, 2012. 253-265.

[13] KWON O, KOH K, LEE J, *et al*. FEGC: an efficient garbage collection scheme for flash memory based storage systems[J]. Journal of Systems and Software, 2011, 84(9):1507-1523.

[14] ANAM Z, SUMAYYA S, SABA Z, *et al*. A Flash aging prevention technique (FAP) for flash based solid state disks[A]. International Conference on Computer Science & Education- ICCSE[C]. Singapore, 2011. 531-536.

[15] KATCHER J. Postmark: A New File System Benchmark[R]. Technical Report TR 3022, Network Appliance, 1997.

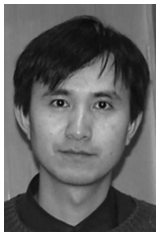
[16] NORCOTT W, CAPPS D. IOzone file system benchmark[EB/OL]. http://www.iozone.org.

[17] Transaction processing performance council TPC-C benchmark[EB/OL]. http://www. tpc.org/tpcc/spec/tpcc_current.pdf, 2004.

[18] KIM Y, TAURAS B, GUPTA A, *et al*. Flashsim: a simulator for NAND flash-based solid-state drives[A]. 1st International Conference on Advances in System Simulation, SIMUL 2009[C]. Porto, Portugal, 2009. 125-131.

作者简介：

陈金忠（1986-），男，安徽巢湖人，博士，哈尔滨工程大学博士生，主要研究方向为 Linux 存储系统、I/O 调度算法和固态硬盘。



姚念民（1974-），男，黑龙江大庆人，博士，哈尔滨工程大学教授、博士生导师，主要研究方向为网络存储、信息安全、无线传感器网络以及高可信计算。

蔡绍滨（1973-），男，黑龙江哈尔滨人，博士，哈尔滨工程大学教授、博士生导师，主要研究方向为无线传感器网络、水声传感器网络。

战福瑞（1986-），男，吉林省吉林市人，哈尔滨工程大学硕士生，主要研究方向为无线传感器网络。

孙美玲（1985-），女，黑龙江五常人，哈尔滨工程大学硕士生，主要研究方向为信号与信息处理、语音识别。