doi:10.11887/j.cn.201501002 http://journal.nudt.edu.cn

闪存阵列的可重构策略*

宋振龙,方 健,魏登萍,张晓明
(国防科技大学 计算机学院,湖南 长沙 410073)

摘 要:利用低延迟、低功耗、高可靠的闪存芯片构建闪存存储阵列是实现高性能存储系统的有效手段。但应用传统磁盘阵列技术构建闪存存储阵列,会引入磨损均衡、校验数据更新频繁导致阵列生命周期降低等问题。针对闪存固有的读写特性,设计实现了一种基于NAND Flash的高性能磁盘阵列机制——基于缓存的可重构磁盘阵列策略。该机制采用可重构条带的思想,利用存储等级的内存作为缓存,对数据顺序重组。实验结果表明:该策略能够有效降低垃圾回收开销,提高闪存阵列的性能和使用寿命。

关键词:闪存;固态存储;阵列;磨损均衡;擦除
中图分类号:TP391.9 **文献标志码**:A **文章编号**:1001-2486(2015)01-008-06

A reconfigurable redundant arrays of inexpensive disks mechanism of NAND Flash

SONG Zhenlong, FANG Jian, WEI Dengping, ZHANG Xiaoming
(College of Cumputer, National University of Defense Technology, Changsha 410073, China)

Abstract: Building storage array based on the low latency, low power consumption, and high reliability NAND Flash is an efficient way to implement high-performance storage system. However, adopting traditional redundant arrays of inexpensive disks (RAID) techniques to build storage array based on NAND Flash bring several problems such as wear-leveling and decreasing lifecycle of storage array caused by updating parity code frequently. A cache-based reconfigurable RAID mechanism that dynamically constructs a new data stripe based on the non-volatile SCM is proposed. Experimental results show that this mechanism can reduce the garbage collection overhead, improve the performance and lifecycle of storage array based on NAND Flash efficiently.

Key words: Flash; solid state storage; redundant arrays of inexpensive disks; wear-leveling; erase

最近几年固态盘或存储阵列由于其功耗小、带宽高等优点在计算机领域得到广泛应用。与传统的磁盘不同,固态盘或固态存储阵列具有自身特点:第一,异地更新的问题。存在页地址中的数据,当需要被更新时只能在非当前页中更新,或者将块擦除后更新。第二,与非型闪速存储器(nor-and flash,NAND Flash)其本身的擦写(program/erase)限制,当擦写次数达到上限以后Flash相应块就会损坏,在构建盘阵后这种问题更加突出。第三,擦除不均衡导致整个存储生命周期受到影响。

当前针对NAND介质的磁盘阵列(Redundant Arrays of Inexpensive Disks, RAID)系统的优化有许多研究。文献[1]已经认识到由于小写给基于Flash介质的RAID系统带来的性能下降问题。文献[2]认识到Flash介质的擦写限制,提出了寿命感知RAID方法。文献[3]提出基于SSD间的磨损均衡(Cross-SSD Wear-Leveling, CSWL)的RAID方法。

为了进一步提高RAID5的写性能,数据可重构策略被广泛的采用和研究。但这些研究主要集中在RAID存储系统的动态扩展上[4-5]。文献[6]提出一种条带动态、可变(Dynamic and Variable Size Striping-RAID,DVS-RAID)机制以提高RAID5机制的读写性能。DVS-RAID根据写请求的到达顺序重新组织数据,避免了数据更新带来的校验码频繁更改问题。但DVS-RAID方法需要为子条带保存部分校验码,这在随机写请求非常多的应用中会造成很大的空间浪费。

为了便于说明问题,假设现在有一存储为基于Flash的RAID,指令队列如图1所示。

如图2所示,初始状态包含4个条带,分别是

* **收稿日期**:2014-06-13
基金项目:国家973计划资助项目(2012AA01A301);国家自然科学基金资助项目(61202118)
作者简介:宋振龙(1978—),男,山东胶州人,助理研究员,博士研究生,E-mail:songzhl@sina.com

strip0,strip1,strip2 和 strip3,每一个条带均包含 5 个数据和 1 个校验,当数据 D00 更新 1 时,不能像传统的基于磁盘的 RAID 在原位置更新(in-place update),只能在异地找一个数据块进行更新。对应指令队列,任何一次写操作都可能引起校验数据的异地更新,对应引发一次校验数据的更新。更新校验数据需要读取原先的数据,因此还会降低 RAID 的性能。同时将传统的 RAID 机制引入 Flash 中可能会引发大量的垃圾回收操作,从而降低存储系统的生命周期。

当上述指令队列完成时,Flash RAID 中的数据如图 2 所示。指令队列合计有 6 次写指令,但是 Flash RAID 中进行了 11 次写。显然直接将 RAID 机制引入 Flash 中,还需要做很多的工作。

针对这个问题,提出一种基于缓存的可重构 RAID 策略(Cache-Based Reconfigurable RAID, CBR-RAID),以解决基于闪存的 RAID 存储系统的小写问题,有效提升系统的生命周期和性能。

Access sequence	1	2	3	4	5	6	7	8	9
Command	Wr	Wr	Wr	Wr	Wr	Wr	Wr	Wr	Wr
Logic Address	S(0,0)	S(1,1)	S(0,3)	S(2,0)	S(3,0)	S(3,5)			

图 1　指令队列

Fig. 1　Command sequence

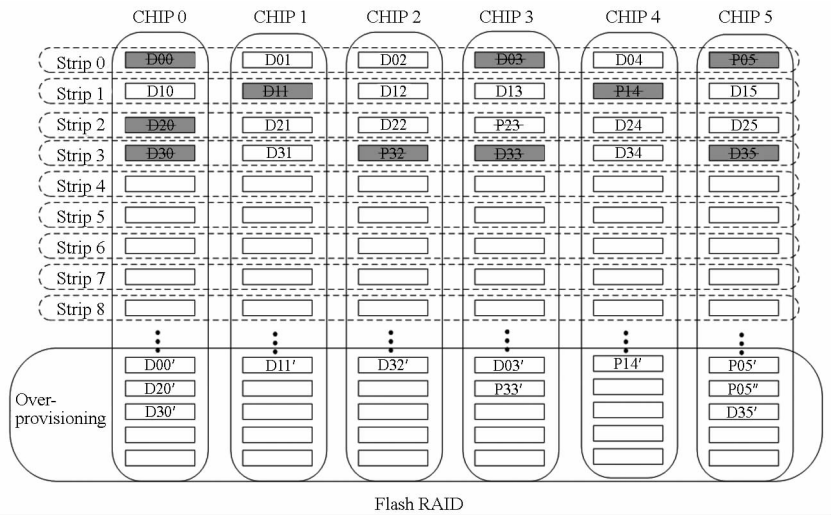

图 2　指令数据状态

Fig. 2　Command data state

1　基于缓存的可重构 RAID 策略

1.1　可重构 RAID 机制

如图 3 所示,假设 6 块 Flash 芯片组成 RAID 存储系统,不同于传统 RAID 机制中条带以数据为单位组织的构建形式,该可重构 RAID 机制以物理页为单位组织 RAID 条带。不同的 Flash 芯片中,具有相同物理块号(Physical Block Number, PBN)和物理页号(Physical Page Number, PPN)的页构成一个 RAID 条带。假设当前系统中保存了 S0 和 S1 的数据。其中,数据 D0 ~ D4 和校验码 P0 组成条带 S0,数据 D5 ~ D9 和校验码 P1 组成条带 S1。当需要更新数据 D1 ~ D5 时,只需要将新数据 D1′ ~ D5′组织成新的条带 S2,计算出该条带的校验码 P2,并令原数据无效即可。

由上述的更新过程可以看出,这种可重构 RAID 机制有以下 4 个优点:1)更新的数据直接组成新的条带,不需要考虑原数据在哪个芯片中,也不需要写更新校验数据,因此可以有效提高系统性能;2)写数据直接构成新的条带,校验码的计算不需要读出原数据,减少了读开销,从而会提高系统的写性能;3)每次写操作都会均匀地写每一个页面,既保证芯片间的磨损均衡,又提高了芯片间的并行性;4)条带由物理块号和物理页号索引,避免了维护条带表的时间和空间开销。

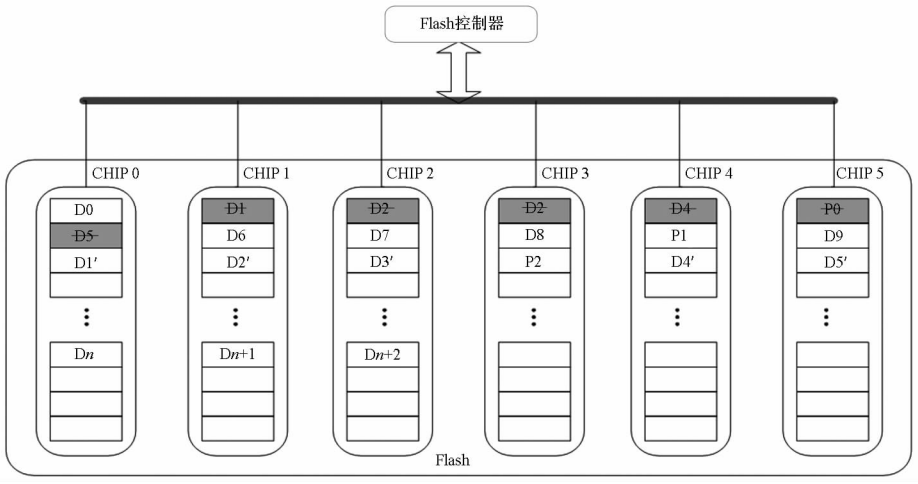

图 3　可重构 RAID 机制数据更新过程

Fig. 3　Update process of constructing RAID

1.2　基于缓存的可重构 RAID 策略

1.2.1　总体结构

CBR – RAID 采用储存级内存(Storage Class Memory,SCM)作为闪存 RAID 的缓存,根据缓存的输出重组 RAID 条带,该系统总体结构如图 4 所示。CBR – RAID 由控制器、SCM 以及 Flash 芯片三部分组成。其中控制器负责接收上层文件系统的请求,管理 SCM 的输入输出,管理 Flash 的地址映射、垃圾回收、磨损均衡等;SCM 和 Flash 芯片负责完成数据的存储。系统的有效存储空间由若干块 Flash 芯片和少量 SCM 共同构成,其中 SCM 负责缓存来自上层文件系统的数据,Flash 负责以 RAID 形式保存数据(为了说明算法的方便,算法中的 RAID 为 RAID5)。以图 4 为例,CBR – RAID 系统包括 6 块 Flash 芯片和 1 块 SCM 芯片。为了能在 Flash 芯片和 SCM 芯片中找到想要的数据,系统需要维护两个地址映射表:Flash 地址映射表(Flash Address Mapping Table, FAMT)和 SCM 地址映射表(SCM Address Mapping Table, SAMT)。

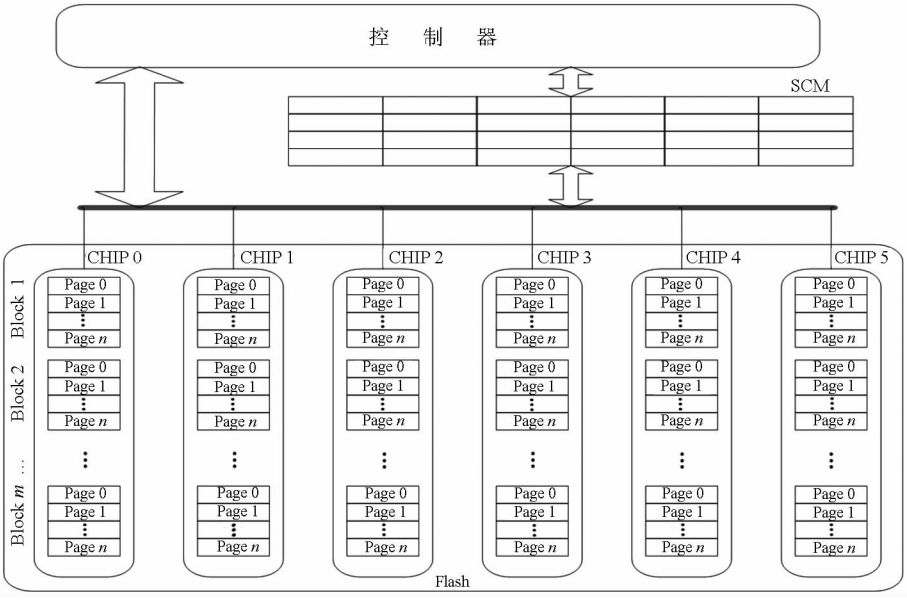

图 4　基于缓存的可重构 RAID 机制结构图

Fig. 4　Architecture of CBR – RAID

数据在 Flash 芯片和 SCM 中有不同的组织形式,但整个系统只维护数据的一个版本,也就是说数据要么在 Flash 芯片中,要么在 SCM 中。数据在 Flash 芯片中以 RAID5 的形式存储,假设系统配置 N 块 Flash 芯片,则每个条带由 $N-1$ 个数据页和 1 个校验码页组成,N 个页分布在 N 块 Flash 芯片中。RAID5 条带以物理单元形式组织,即不同芯片中具有相同物理块号和相同物理页号的页

组成条带,而不依赖于数据,属于同一条带的不同芯片中的页用芯片号(Chip Number,CN)区分。数据在 SCM 以页形式存储,页大小与 Flash 芯片的页大小一致。

CBR – RAID 的基本工作原理如下:

(1)当读请求到达时,系统首先查找 SCM,若 SCM 中有需要的数据则读取该数据,把数据返回给请求方,读请求处理结束;若 SCM 中没有相应的数据,则查找 Flash,并直接把从 Flash 读取的数据返回给请求方,而不需要缓存到 SCM 中。

(2)当写请求到达时,系统只将数据写到 SCM 中。若 SCM 中未缓存该写数据的旧版本,数据直接写入 SCM 中。若 SCM 中缓存有该写数据的全部或部分,先修改已缓存的部分,即将新数据覆盖其旧版本,其余未缓存部分直接写入 SCM 中。

(3)当 SCM 空间不足时,SCM 需要替换出部分空间,将数据写回 Flash 中,以确保 SCM 有足够的空间响应写请求。SCM 写回时,选择与 Flash 中 RAID 条带数据项同等数量的页,重组 RAID 条带,并为该条带计算新的校验码,并行将该条带写入。数据写完,清空 SCM 中的数据。

从上述工作原理可以看出,该方法具有以下 4 个特性:1)该方法利用 SCM 缓存写入的数据,隐性地利用了读写的局部性原理,提高读写性能;2)该方法对缓存于 SCM 中的写请求重新组织 RAID 条带,减少数据和校验码频繁更改给 Flash 芯片带来的磨损,从而增加了 Flash 的使用寿命;3)该方法在 SCM 替换数据时,以全并行的方式写 Flash 芯片,充分利用了 RAID 的并行性,提高了性能;4)该方法延迟条带检验码的计算和写入,等到系统空闲时或需要替换时再计算条带的校验码并写入 Flash 中,提高了系统写性能。

SCM 由若干个页组成,页的大小与 Flash 芯片中页的大小一致。当上层文件系统写请求到达时,要写入的数据可写到 SCM 中任意一页。当 SCM 空间不足需要写回 Flash 芯片中时,SCM 选择当中的 $N-1$ 个页(N 为 Flash 中的条带大小,含校验页)组成新条带,并计算该条带校验码,将此 N 个页并行写入 Flash 芯片中,实现 N 个通道同时利用,加快 Flash 的写入速度。对于 SCM,为其维护一个采用页映射的地址映射表 SAMT,该映射表与 Flash 的映射表 FAMT 最大区别在于数据修改的操作上。Flash 数据改写时,由于异地更新,会为原数据页分配新的存储单元,并修改映射表。而 SCM 数据改写时,由于进行本地更新,新数据可直接写在原来的存储单元中,映射表不需要修改。

1.2.2　工作过程

(1)读请求的处理

当读请求到达时,系统根据读请求的逻辑地址(Logical Address,LA),找到数据并返回。

(2)写请求的处理

写请求与读请求不同,上层文件系统的所有写操作都由 SCM 响应。当写请求到达时,系统首先查找 SAMT,判断 SCM 是否缓存有该写请求的所有数据或部分数据。若 SCM 中没有缓存要写入的数据,则将要写入的数据直接写到 SCM 的空闲空间中,并在 SAMT 中加入新写入的项。在数据写入 SCM 中时,需要无效 Flash 中的相应数据。当 SCM 中缓存有该写请求的所有数据或部分数据时,缓存数据直接覆盖修改,SAMT 不变。

(3)SCM 写回过程

当 SCM 的空闲空间不足时,需要替换出部分数据并将其写回 Flash 中。SCM 每次替换多个页,替换页的数量与 Flash 中一个条带包含的数据页数相等。当需要写回时,SCM 将这些数据组织成 RAID 条带,计算该条带的校验码后,将数据和校验码并行写入 Flash 中。写回的触发条件有两种。第一种是主动写回,当系统空闲时,根据替换策略选择将要被写回的多个页替换出 SCM。第二种是被动写回,当 SCM 中空闲空间不足时必须选择一定数量的页写回 Flash 中,以保证 SCM 有足够的空闲空间缓存将要到达的写数据。写回操作分两步:第一步根据替换策略选择将要被替换的候选数据页,并组织成条带形式;第二步计算该条带的校验码,再将数据和校验码并行写入到 Flash 中。当一个页被写回 Flash 时,需要删除该页在 SAMT 中的项,并在 FAMT 中为其建立新的地址映射。这种直接以条带为单位的写回方式使得芯片间的磨损比较均衡,从而延长系统中 Flash 的使用寿命。

(4)数据恢复过程

当 CBR – RAID 系统中 Flash 的某些页出现读错误,可以通过 RAID 算法修复。通过该页所在的条带的其余数据(包括有效数据和无效数据)以及校验码,恢复该页数据。由于 Flash 中,属于同一条带的不同页具有相同的物理块号和物理页号,所以,在读取修复需要的数据时可以通过该页的物理块号和物理页号获得其余数据。

(5)垃圾回收(Garbage Collection ,GC)

任何属于同一个条带的数据必须在同一次

GC 操作中被迁移。定义不同芯片中有相同物理块号的块组成一个全局块(Global Block, GB),垃圾回收时以 GB 为单位。垃圾回收操作的开销主要来源于被回收块中有效数据的迁移,被回收块中有效数据越多,垃圾回收的开销越大。因此,为了降低每次垃圾回收的开销,系统在做垃圾回收操作时,优先选择所有 GB 中有效页最少的 GB。

1.2.3 SCM 替换策略

当 SCM 空间不足时,需要替换出一定数量的页,以保证 SCM 的正常工作。由于 SCM 中数据是以条带形式写回 Flash,因此每次需要替换出大小等于整条带的数据。为了使所选将要被替换的条带给系统带来的损失最小,需要设计替换策略。

最简单有效的替换策略是最近最少使用法(Least Recently Used, LRU),该算法能充分利用数据访问的时间和空间局部性,使缓存获得较高的命中率。但该算法没有考虑闪存的特性,不利于闪存性能的发挥。由于写操作和擦除操作是访问闪存的主要开销,目前许多研究主要集中在通过减少写操作来提高闪存系统的读写特性。干净优先区(Clean First Least Recently Used, CFLRU)策略[7]将缓存分为工作区和干净优先区,当需要替换时,优先选择干净优先区中的一个干净数据进行替换,若此时无干净数据则选择最近最少使用的脏数据进行替换。基于 LRU 的写重定序(LRU and Writes Sequence Reordering, LRU - WSR)策略[8]在 CFLRU 基础上增加了数据的冷热属性。当需要替换时,LRU - WSR 首先选择最近最少使用的页,若此页为干净数据或冷脏数据,则直接替换;若此页为热脏数据,则将其标记为冷脏数据。上述两种方法提高了缓存的命中率,减少了 Flash 的写操作,但这两种方法并未考虑替换时写 Flash 引发的合并、垃圾回收等额外开销。韩国三星电子软件实验室的 Kim 等[9]提出了基于重构块(Block Padding LRU, BPLRU)策略,该策略以块为聚簇,以上层的随机写请求重构为顺序写请求,并以块为单位写入 Flash。BPLRU 策略减少了 Flash 的写操作和擦除操作,提高存储系统的读写性能,但该策略基于块映射,并不适用于页映射的可重构 RAID 机制。

针对可重构 RAID 机制,现提出最近最少使用组替换策略(Least Recently Used Group, LRUG)。LRUG 策略为每个页保存一个替换度,替换度初始值为 0,也就是说当一个页被写入或修改时,替换度为 0。替换度在每次写请求到达时,以常数大小自动增加,替换度高的页优先替换。考虑到数据页之间的关联,本策略将来自同一个写请求的所有页归为一组,属于同一组的数据有相同的替换度。当写某组的某个页时,置该组所有页的替换度为 0,并将该页从本组删除,删除页与新写请求的其他页重新分为一组。如此对组进行重构是为了避免同一组中只有少部分页频繁更新,而其余冷数据页占据大量空间的情况。

以图 5 为例,此时 SCM 中保存了四组数据,其中 D0 ~ D3 为第一组,D4、D5 为第二组,D6 ~ D8 为第三组,D9 ~ D13 为第四组,括号内为每个页的替换度,并假设替换度每次自增 10。当写请求 D13 ~ D17 到达时,首先修改 D13 为 D13′并将与 D13 同为一组的 D9 ~ D12 替换度改为 0。然后 D14 ~ D17 写入 SCM 中,与 D13′归为同一组,该组替换度为 0。更新后结果如图 6 所示。

D0(60)	D1(60)	D2(60)	D3(60)	D4(80)	D5(80)
D6(50)	D7(50)	D8(50)	D9(80)	D10(80)	D11(80)
D12(80)	D13(80)				

图 5 初始状态

Fig. 5 Initial state

D0(70)	D1(70)	D2(70)	D3(70)	D4(90)	D5(90)
D6(60)	D7(60)	D8(60)	D9(0)	D10(0)	D11(0)
D12(0)	D13′(0)	D14(0)	D15(0)	D16(0)	D17(0)

图 6 更新后状态

Fig. 6 State after updating

在选择替换候选页时,LRUG 总是选择替换度最大的页替换,目的是将最不常用的数据替换出 SCM。优点:一组内数据再次被同时访问的概率较大,以条带形式写回 Flash 中,当下次需要读时,可以并行读取而不至于读请求都集中在同一块 Flash 芯片中。

2 性能测试

2.1 实验环境

为了验证 CBR - RAID 的性能,在 Disksim SSD extension[10-11]上实现了 RAID5 和 CBR - RAID(基于 RAID5)。本实验中所模拟固态盘的参数设置如表 1 所示,实验中 RAID 条带大小为 8 个页。

表 1　固态盘模拟参数设置

Tab. 1　Parameter of SSD simulator

参数	值	参数	值
容量	32GB	读延迟	25μs
页大小	4KB	写延迟	200μs
块大小	256KB	擦除延迟	1.5ms
最小空闲块	5%	最大擦除次数	100K

本次实验所用的 Trace 共有 4 个,Trace1 来自运行在金融服务器上的在线事务处理应用程序(On-Line Transaction Processing,OLTP)[12],Trace2 来自邮件服务器的工作负载[13],Trace3、Trace4 是由 diskmon 收集的,其中 Trace3 包含大量随机读请求和随机写请求,Trace4 包含大量顺序读请求和顺序写请求。其特性如表 2 所示。

表 2　相关 Trace 特性

Tab. 2　Characteristics of I/O Trace

Workloads	总请求数	平均写请求大小	读写比例
Trace1	5 334 987	11.9KB	0.301:1
Trace2	2 909 798	222KB	4.881:1
Trace3	3 210 349	12.6KB	0.282:1
Trace4	3 244 863	18.9KB	0.384:1

2.2　测试结果

2.2.1　平均执行时间

图 7 显示了 RAID0、CBR – RAID、RAID5 在 4 种不同的负载下的平均执行时间的比较结果,其中 y 轴表示平均运行时间。对于 Trace1 和 Trace3,CBR – RAID 相对 RAID5 分别有 23% 和 38% 的性能提升。

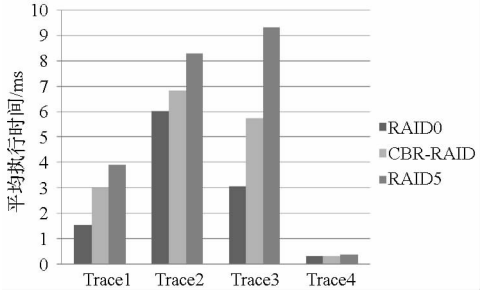

图 7　平均运行时间

Fig. 7　Average run time

2.2.2　擦除次数

图 8 显示了 RAID0、CBR – RAID、RAID5 在 4 种不同的负载下的擦除次数的比较结果,其中 y 轴表示擦除时间。从图 8 中可以看出,与 RAID5 比较对于包含较多随机写的 Trace1 和 Trace3,CBR – RAID 分别能减少 22% 和 46% 的擦除操作。

2.2.3　垃圾回收次数

图 9 显示了 RAID0、CBR – RAID、RAID5 在 4 种不同的负载下的垃圾回收时间的比较结果,其中 y 轴表示回收比例。为了方便比较,以 RAID5 的回收次数为基准。从图 9 中可以看出,RAID0 具有最小的回收开销,CBR – RAID 次之,而 RAID5 进行了最多的回收操作。在随机访问的负载下,RAID5 机制由于小写请求引发了大量的校验码写入,垃圾回收的频率相对 CBR – RAID 更多。顺序访问负载下 CBR – RAID 和 RAID5 差距较小,因为这种负载下两者的校验码更新频率是相当的,在随机写较多的情况下,CBR – RAID 较 RAID 5 机制能有效降低垃圾回收开销。

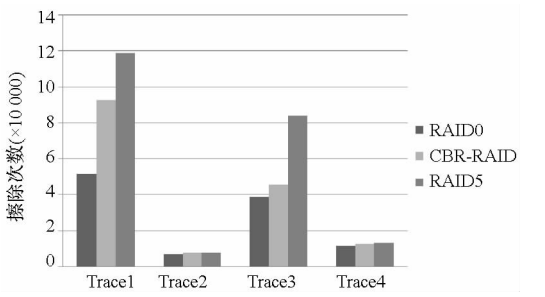

图 8　擦除次数

Fig. 8　Count of erase

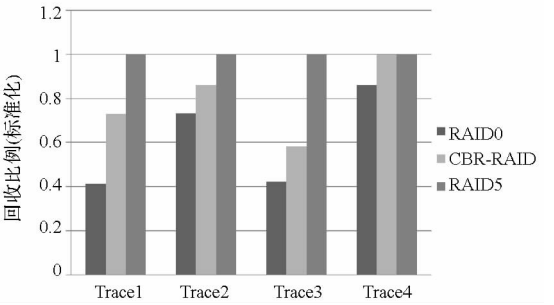

图 9　垃圾回收

Fig. 9　Garbage collection

3　结论

利用 NAND Flash 构建 RAID 已经在许多系统中得到实际的应用[5],但是存在各种问题,也是当前的研究热点,基于 Flash 特性的可重构 RAID 策略利用 SCM 特性,将数据重新组织并写入到 RAID 中,有效降低了 Flash 中异地更新带来的写放大问题。实验证明采用 CBR – RAID 的 Flash 阵列能够显著提升系统的性能,降低擦除次数,减少垃圾回收的次数。当然这种方法带来地址转换表的复杂设计,需要结合 CBR – RAID 的特点对闪存地址映射(Flash Translation Layer,FTL)进行优化,期望减少 FTL 的空间。通过引入 CBR – RAID 机制,Flash 磁盘阵列高效地实现冷热数据分离,可以有效降低垃圾回收开销,减少对 Flash 的磨损。对后续在这两方面将继续关注并进行研究。

(下转第 20 页)

speed up cache-to-cache transfers in CC-NUMA multiprocessors[C]// Proceedings of the 14th International Parallel and Distributed Processing Symposium (IPDPS'00), Cancun, Mexico, 2000: 721 – 728.

[4] Zhang Z. Architectural sensitive application characterization: the approach of high-performance index-set (HP-Set)[R]. Technical Report HPL – 2001 – 75, HP Laboratories Palo Alto, USA, 2001.

[5] Abdel-Shafi H, Hall J, Adve S V, et al. An evaluation of the fine-grain producer-initiated communication in cache-coherent multiprocessors[C]//Proceedings of the 3rd IEEE Symposium On High-Performance Computer Architecture, San Antonio, USA, 1997:204 – 215.

[6] Trancoso P, Torrellas J. The impact of speeding up critical sections with data prefetching and forwarding[C]//Proceedings of the International Conference on Parallel Processing, Minneapolis, USA, 1996:79 – 86.

[7] Martin M M K, Harper P J, Sorin D J, et al. Using destination-set prediction to improve the latency/bandwidth tradeoff in shared-memory multiprocessors[C]//Proceedings of the International Symposium on Computer Architecture, San Diego, USA, 2003:206 – 217.

[8] Demetriades S, Cho S. Predicting coherence communication by tracking synchronization points at run time[C]//Proceedings of the International Symposium on Microarchitecture, Vancouver, Canada, 2012:351 – 362.

[9] Lebeck A R, Wood D A. Dynamic self-invalidation: reducing coherence overhead in shared-memory multiprocessors[C]// Proceedings of the International Symposium on Computer Architecture, Santa Margherita Ligure, Italy, 1995:48 – 59.

[10] Lai A, Falsafi B. Selective, accurate, and timely self-invalidation using last-touch prediction[C]//Proceedings of the International Symposium on Computer Architecture, Vancouver, Canada, 2000: 139 – 148.

[11] Somogyi S, Wenisch T F, Hardavellas N, et al. Memory coherence activity prediction in commercial workloads[C]// Proceedings of the 2004 Workshop on Memory Performance Issues, Munich, Germany, 2004:37 – 45.

[12] Cheng L, Carter J B, Dai D. An adaptive cache coherence protocol optimized for producer-consumer sharing[C]// Proceedings of the International Symposium on High Performance Computer Architecture, Scottsdale, USA, 2007: 328 – 339.

[13] Magnusson P S, Christensson M, Eskilson J, et al. Simics: A full system simulation platform[J]. IEEE Transactions on Computer, 2002, 35(2):50 – 58.

[14] Martin M M K, Sorin D J, Beckmann B M, et al. Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset[J]. ACM SIGARCH Computer Architecture News, 2005 33(4):92 – 99.

(上接第 13 页)

参考文献(References)

[1] Baek S H, Park K H. Matrix-stripe-cache-based contiguity transform for fragmented writes in RAID-5 [J]. IEEE Transaction Computer, 2007,56(8): 1040 – 1054.

[2] Du Y M, Xiao N, Liu F, et al. CSWL: cross-SSD wear-leveling method in SSD-based RAID systems for system endurance and performance[J]. Journal of Computer Science and Technology, 2013, 28(1): 28 – 41.

[3] Chung T S, Park D J, Park S, et al. System software for flash memory: a survey[C]//Proceedings of International Conference Embedded and Ubiquitous Computing, 2006: 394 – 404.

[4] Baek S H, Park H H. Prefecting with adaptive cache culling for striped disk array[C]//Proceedings of USENIX, Boston, 2008: 363 – 376.

[5] 刘秀菊. RAID5 小写更新的冗余管理机制[J]. 微电子学与计算机,2012, 29(7): 112 – 115.
LIU Xiuju. A Redundancy management policy for RAID5[J]. Microelectronics & Computer, 2012, 29(7): 112 – 115. (in Chinese)

[6] Li Z, Jin P Q, Su X, et al. CCF-LRU: a new buffer replacement algorithm for flash memory[J]. IEEE Transaction on Consumer Electronics, 2009, 55(3): 1351 – 1359.

[7] Woodhouse D. JFFs: the journaling flash file system[EB/OL]. 2001[2008 – 12 – 05]. http://sources. redhat. com/jffs2/jffs2. pdf.

[8] Wu C T, He X B. GSR: a global stripe-based redistribution approach to accelerate RAID5 scaling[C]//Proceedings of 41st ICPP, 2012: 460 – 469.

[9] Zhang G Y, Zheng W M, Shu J W. ALV: a new data redistribution approach to RAID5 scaling [J]. IEEE Transactions on Computer, 2010, 59(3): 345 – 357.

[10] Kim H, Ahn S. BPLRU: a buffer management scheme for improving random writes in flash storage[C]//Proceedings of 6th USENIX Conference on File and Storage Technologies, 2008.

[11] Bucy J S, Schindler J, Schlosser S W, et al. The disksim simulation environment version 4.0 reference manual[R]. Carnegie Mellon University Parallel Data Lab Technical Report CMU – PDL – 08 – 101, 2008.

[12] Prabhakaran V, Wobber T. SSD extension for DiskSim simulation environment[CP/OL]. 2009. http://research. microsoft. com/en-us/downloads/b41019e2-1d2b-44d8-b512-ba35ab814cd4/

[13] 方健,宋振龙,李琼,等. 基于闪存的存储阵列研究[C]// 第十七届计算机工程与工艺年会暨第三届微处理器技术论坛, 2013:170 – 176.
FANG Jian, SONG Zhenlong, LI Qiong, et al. The research of raid based on NAND flash[C]//Proceedings of NCCET, 2013:170 – 176. (in Chinese)