软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2014,25(11):2575−2586 [doi: 10.13328/j.cnki.jos.004573]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

一种闪存敏感的多级缓存管理方法*

王江涛[1,2], 赖文豫[1], 孟小峰[1]

[1](中国人民大学 信息学院,北京 100872)

[2](淮阴师范学院 计算机科学与技术学院,江苏 淮安 223300)

通讯作者: 王江涛, E-mail: jiangtaow@ruc.edu.cn, http://www.ruc.edu.cn

摘 要: 基于闪存的固态硬盘(solid state driver,简称 SSD)已经广泛应用于各种移动设备、PC 机和服务器.与磁盘相比,尽管 SSD 具有数据存取速度高、抗震、低功耗等优良特性,但 SSD 自身也存在读写不对称、价格昂贵等不利因素,这使得 SSD 短期内不会完全取代磁盘.将 SSD 和磁盘组合构建混合系统,可以发挥不同的硬件特性,提升系统性能.基于 MLC 型 SSD 和 SLC 型 SSD 之间的特性差异,提出了一种闪存敏感的多级缓存管理策略——FAMC.FAMC 将 SSD 用在内存和磁盘之间作扩展缓存,针对数据库系统、文件管理中数据访问的特点,有选择地将内存牺牲页缓存到不同类型的 SSD.FAMC 同时考虑写请求模式和负载类型对系统性能的影响,设计实现对 SSD 友好的数据管理策略.此外,FAMC 基于不同的数据置换代价提出了适用于 SSD 的缓冲区管理算法.基于多级缓存存储系统对 FAMC 的性能进行了评测,实验结果表明,FAMC 可以大幅度降低系统响应时间,减少磁盘 I/O.

关键词: 固态硬盘;数据库;多级缓存;置换代价;缓冲区

中图法分类号: TP311

中文引用格式: 王江涛,赖文豫,孟小峰.一种闪存敏感的多级缓存管理方法.软件学报,2014,25(11):2575−2586. http://www.jos.org.cn/1000-9825/4573.htm

英文引用格式: Wang JT, Lai WY, Meng XF. Flash-Aware multi-level cache management strategy. Ruan Jian Xue Bao/Journal of Software, 2014,25(11):2575−2586 (in Chinese). http://www.jos.org.cn/1000-9825/4573.htm

Flash-Aware Multi-Level Cache Management Strategy

WANG Jiang-Tao[1,2], LAI Wen-Yu[1], MENG Xiao-Feng[1]

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

[2](School of Computer Science and Technology, Huaiyin Normal University, Huaian 223300, China)

Corresponding author: WANG Jiang-Tao, E-mail: jiangtaow@ruc.edu.cn, http://www.ruc.edu.cn

Abstract: Solid state driver (SSD) based on flash memory has been widely used in various types of applications such as mobile devices, PC machines, and servers. Compared with conventional disk, SSD enjoys faster access speed, better shock resistance, and lower power. However, it will not completely replace the disk as the secondary storage in the short run due to its inherent properties such as asymmetric read/write and high price of per gigabyte. Integrating SSD and magnetic disk together can benefit from different performance advantage to obtain good high performance and low cost. This paper proposes a flash-aware multi-level cache scheme (FAMC) which considers the significant discrepancy between MLC type and SLC type SSDs. FAMC uses two types of SSD as a cache layer between main memory and magnetic disk. Depending on the characteristics of data access in database application and file management, FAMC conditionally caches the page evicted by buffer manager to different types of SSD. FAMC considers the impact of the write pattern and type of workloads on the system performance, which adopts flash-aware algorithms and data structures to manage the data stored in SSD. Furthermore, in view of the asymmetry of the replacement cost, the study proposes a flash-friendly buffer replacement policy. The strategy is implemented on a

* 基金项目: 国家自然科学基金(61379050, 91224008); 国家高技术研究发展计划(863)(2013AA013204); 教育部高等学校博士学科点专项科研基金(20130004130001)

收稿时间: 2013-09-10; 修改时间: 2013-12-18; 定稿时间: 2014-01-27

simulation storage system based on SLC type SSDs and MLC type SSDs, and its performance is evaluated. The experimental results show that FAMC can significantly reduce system response time and disk I/O cost.

Key words: solid state driver; database; multi-level cache; replacement cost; buffer

随着社交网络、移动互联网等一系列新技术和应用的飞速发展,数据产生的规模和速度呈现指数级增长,基于磁盘的数据管理系统难以满足海量数据应用对存储带宽的需求,数据处理亟需新型高效的存储设备来提高系统性能.闪存的出现,为解决这一问题提供了有效途径.闪存是一种全电设备,具有非易失、高速读写、抗震、低功耗等特性.最近几年,随着闪存芯片容量的增加和价格的下降,基于闪存的固态硬盘(solid state drive,简称SSD)开始应用于高吞吐、数据访问密集等企业级应用环境[1−3].SSD 由闪存芯片封装而成,对外提供和磁盘相同的 I/O 接口,上层应用不需做任何修改即可直接在 SSD 上运行.虽然 SSD 具有磁盘无法比拟的诸多优良特性,但 SSD 也并不是完美无缺的,闪存自身固有的一些物理特性使得 SSD 在短期内不会完全取代磁盘成为主流的存储介质:

- 首先,SSD的读写代价是不对称的.一般来说,SSD的读性能非常好,但随机写性能较差,频繁的随机写操作会使 SSD 的读写速度和带宽大幅下降.
- 其次,闪存具有写前擦除特性.对某一数据页重写前需要对整个数据块执行擦除操作:一方面,擦除操作的代价非常昂贵;另一方面,因为每个闪存存储块可供擦除的次数是有限的,频繁的随机写操作不仅会降低读写速度,更会导致闪存可用寿命的缩短.
- 最后,尽管 SSD 的价格在不断走低,但即便是低端产品,其单位容量的价格仍然是磁盘的 10 倍以上.

在现有的存储体系中引入 SSD,根据 SSD 和磁盘的物理特性,有选择地把数据分配到不同的存储介质来处理,这样不仅可以满足大数据、高带宽等数据应用的需求,也可以有效降低企业成本.将 SSD 和磁盘共存形成混合存储系统,是未来一段时期数据管理的研究热点.

在混合存储系统中,在内存和磁盘之间添加 SSD 作扩展缓存层,可以大幅提升缓冲区访问性能,缓解磁盘 I/O 压力,缩短系统恢复时间.目前,已有一些学者和机构就利用 SSD 缓存系统来解决磁盘 I/O 瓶颈问题进行了研究[4−6].如何将经常访问的数据透明地缓存在高速固态存储系统,是混合存储数据管理研究的一个重要方向.

本文基于 SLC 型 SSD 和 MLC 型 SSD 在产品价格、读写性能和可用寿命等方面的巨大差异,提出一种闪存敏感的多级缓存管理策略——FAMC.FAMC 采用两种不同类型的固态硬盘作内存的扩展缓存,根据内存牺牲页的状态和置换代价,在保持较高访问命中率的同时,设计实现了对不同类型 SSD 友好的替换算法.总体来说,本文的主要贡献如下:

1) 依据闪存读写的不对称特性,设计实现了基于置换代价的缓冲区替换策略.算法根据负载访问特性和页面状态调整内存数据页的置换优先级,有效降低了置换代价.
2) 基于 SLC 和 MLC 型 SSD 的性能差异,有选择性地将内存中的干净页和脏页分别缓存在两种 SSD 中,对于缓存的数据页,FAMC 设计实现了对不同类型 SSD 敏感的数据写入和替换策略.
3) 采用不同粒度的存储策略组织缓存在 SSD 中的数据,内存缺页时能够快速识别并读取相关数据,SSD 空间碎片的回收效率高.

本文第 1 节介绍不同类型固态盘的存储特性、问题描述和相关工作.第 2 节详细介绍本文设计的 FAMC 多级缓存架构.第 3 节通过实验结果对比分析 FAMC 的性能优势.最后总结全文.

1 问题定义及相关工作

1.1 闪存存储器

闪存芯片可以分为 NOR 型和 NAND 型两种,其中,NOR 型闪存可以按位访问,容量小,主要用于存储可执行的程序代码;NAND 型闪存的容量大,适合进行数据存储.现有的 SSD 基本上都是基于 NAND 型闪存设计.根据芯片上每个晶胞所能存储的比特位数,NAND 闪存又可以分为单级晶胞(SLC)和多级晶胞(MLC)两类,两者之间

的区别在于每单元存储的数据量不同,SLC 每单元存储 1 个比特位,MLC 每单元存储多个比特位.SLC 和 MLC 型 SSD 在价格、读写性能和寿命等方面均有较大的差异.本文对多款 SSD 产品进行了性能测试,表 1 给出其中 3 款 SSD 产品的性能对比.

Table 1 Comparison of price and performance for MLC-based SSDs and SLC-based SSDs

表 1 MLC 型 SSD 和 SLC 型 SSD 性能价格比较

SSD 类型	4KB 随机访问吞吐(IOPS)		顺序访问带宽(MB/s)		价格($/GB)
	读操作	写操作	读操作	写操作	
MLC1_SSD	10 989	4 407	156.2	54.7	1.37
MLC2_SSD	36 278	13 177	254.39	83.17	2.12
SLC_SSD	38 018	23 223	261.2	189.23	13.81

SLC 型 SSD 与 MLC 型 SSD 相比有更好的写性能、更长的写入寿命,但在容量和价格方面却又存在很大的劣势,因此,根据数据应用的特点和不同类型 SSD 的产品优势,将两种类型 SSD 与磁盘组合构建混合存储系统,会获得更高的性价比.

1.2 问题定义

在基于固态盘的多级缓存架构中,SSD 的作用是暂存从内存置换出的数据,在内存缺页时,系统首先访问 SSD 中的数据.若命中,则将数据读入内存;否则,从磁盘读取数据.SSD 的存储容量远超过内存,通过在 SSD 暂存内存换出的数据,可以大量减少系统缺页引发的磁盘 I/O 请求,有效弥补内存和磁盘之间的性能差异.然而,SSD 有与内存和磁盘截然不同的存储特性.例如,SSD 随机写操作会触发许多闪存块的擦除,擦除操作不仅代价昂贵而且还会在闪存块内产生很多存储碎片,这些存储碎片使得逻辑地址与物理地址不再连续,进而影响闪存的预读性能.文献[7]指出,存有大量存储碎片的闪存其读写性能只有原来的 30%.另一方面,频繁的擦除操作会缩短闪存的使用寿命,影响系统的可靠性和可用性[8].因此,在内存-SSD-磁盘多级存储体系中,缓存管理算法不仅要考虑数据页的读写状态、数据访问形式,而且要结合闪存的读写特性来设计对闪存敏感的结构和应用.SSD 作扩展缓存层最需要解决以下几个核心问题:

- 对于内存中的数据,缓冲区替换算法选择哪些数据,并以什么样的方式缓存到 SSD?
- 如何组织管理暂存在 SSD 中的数据?
- SSD 缓存管理策略在何时将数据换出或写回磁盘?

1.3 相关工作

在基于闪存的多级缓存系统领域,研究者主要致力于设计高效的缓存管理策略来提高 SSD 访问命中率,利用 SSD 高速的随机读性能,缓解磁盘 I/O 瓶颈,提升系统吞吐和响应性能.文献[9]提出了一种基于 SSD 的多级缓存系统——TAC 管理策略.TAC 以数据片段(连续存储的 32 个数据页为一个片段)为热度计量单位来记录数据当前被访问的冷热程度,当数据从磁盘读入内存时,TAC 判断该数据所在的片段是否满足当前的热度阈值,若满足,则数据被同步写入 SSD.当内存缺页时,系统会首先访问缓存在 SSD 的数据,若访问命中,则从 SSD 读取相应数据.当 SSD 可用空间不足时,TAC 将当前热度最低的数据置换出 SSD.TAC 利用 SSD 做扩展缓存,在一定程度上降低了磁盘读操作的次数,但不能减少对磁盘的写操作,对于更新操作密集的负载,频繁的磁盘 I/O 会制约缓存性能的提升.文献[10]依据 SSD 缓存数据页的类型和脏页回写方式,提出 CW,DW 和 LC 这 3 种缓存管理策略,其中,CW 只将内存换出的干净页缓存至 SSD;DW 缓存内存换出的脏页和干净页,脏页将被同步写入 SSD 和磁盘;LC 与 DW 的区别在于处理脏页的方式不同,LC 首先将内存换出的脏页暂存于 SSD,当聚集的脏页满足一定条件时,再批量写入磁盘.对于更新操作密集的负载,LC 策略引发的磁盘 I/O 数量明显降低,这是因为 LC 将内存换出的脏页批量缓存于 SSD,系统对同一数据页的多次修改可能经过 SSD 暂存而变成最终对磁盘的一次写操作;而对于读操作密集的负载,CW 和 DW 省去了 LC 维护脏数据页队列复杂的管理代价,因此系统性能相对较好.文献[11]提出的 FaCE 系统采用顺序追加的方式将数据写入 SSD,如果待写入的数据在 SSD 存有旧版本,FaCE 将其标记为无效数据.FaCE 实现了对数据的顺序写操作,充分利用 SSD 的高速随机读性能和非易失性

提高数据访问的吞吐量,减少系统恢复时间.然而,尽管 FaCE 避免了对 SSD 的随机写操作,但无效数据回收、SSD 有效存储空间降低和复杂的元数据管理等问题制约了系统性能的进一步提升.

2 FAMC 缓存策略

2.1 系统架构

FAMC 缓存策略的系统架构如图 1 所示,图中箭头代表数据页的流向,阴影正方形代表脏页,MLC 固态硬盘中的长方形为页块(segment).内存缓冲区逻辑上分为干净页和脏页两个 LRU 队列.

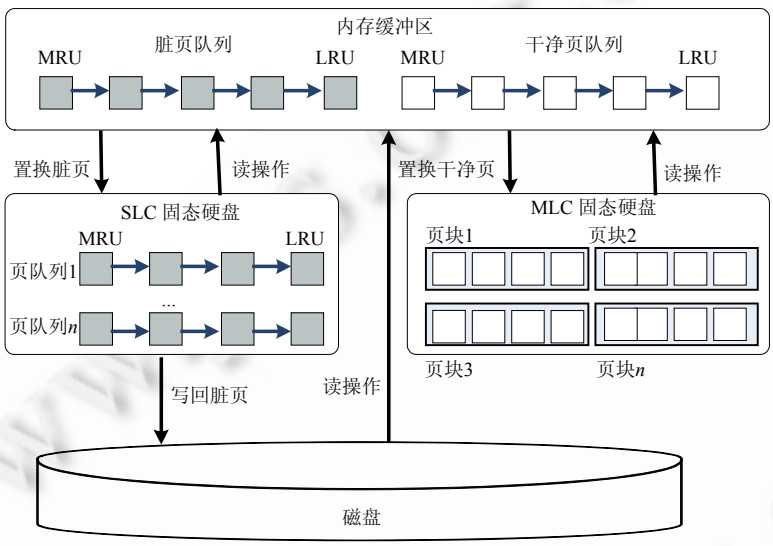


Fig.1 Structure of FAMC

图 1 FAMC 的结构

系统启动后,数据从磁盘读入内存,当内存可用缓冲区不足,需要置换数据页时,FAMC 依据替换算法选择牺牲页置换到 SSD 缓存层,干净页写入 MLC 型 SSD,脏页写入 SLC 型 SSD.如果内存发生缺页,FAMC 首先访问 SSD 缓存层,若命中,则读数据进入内存;若不命中,则从磁盘读入数据.当 SSD 的可用空间达到设定的阈值时,则需要将 SSD 中的部分数据替换出去,SLC 中的脏页需要写回磁盘.

FAMC 缓存策略关键模块包括:

(1) 针对 SSD 的读写不对称特性,设计合理的内存缓冲区替换算法(将在第 2.2 节详细介绍);

(2) 内存数据如何被缓存或替换出 MLC 型和 SLC 型 SSD (将在第 2.3 节详细介绍).

2.2 内存缓冲区管理

经典的内存缓冲区替换算法的目标是提高访问命中率,减少磁盘 I/O,然而对于基于闪存的缓冲区管理来说,维持较高的访问命中率并不一定能降低外存访问开销[12,13].这是因为闪存的写性能低于读性能,置换脏页引发的读写代价要远大于替换干净页的代价.在 FAMC 中,内存置换出的数据首先会被写到基于闪存的固态盘,因此,缓冲区替换算法必须在访问命中率和昂贵的闪存写代价之间做一个权衡.FAMC 将内存中的数据页按页面状态分为 Q_c 和 Q_d 两个 LRU 队列,Q_c 和 Q_d 分别负责维护干净页和脏页,队列长度满足公式(1):

$$\frac{length(Q_c)}{length(Q_d)}=\frac{C_r}{C_r+C_w}\times\theta=\omega \tag{1}$$

其中,C_r 和 C_w 代表对 SSD 执行一次读和写操作的时间,θ的大小取决于 SLC 和 MLC 固态盘的存储容量以及负载类型.FAMC 内存缓冲区数据替换过程见算法 1.

算法 1. *Selectvictim*(buffer B,sizeratio ω).

输入:B 是当前缓冲区;ω是队列长度比阈值.

1. L_c=*Length*(Q_c) //计算队列 Q_c 的长度
2. L_d=*Length*(Q_d) //计算队列 Q_d 的长度
3. if $L_c/L_d<\omega$ then
4. q=select the tail of Q_d queue //从 Q_d 选择牺牲页 q
5. else
6. q=select the tail of Q_c queue //从 Q_c 选择牺牲页 q
7. end if
8. return q

算法 1 中,参数ω用于控制 Q_c 和 Q_d 的队列长度,ω取值与 SSD 不同的读写速度、可用容量以及负载访问模式相关.为了实时监测负载访问情况,FAMC 在内存维护了一个记录数据页读写比例的变量,根据获取的读写信息,周期性地调整 Q_c 和 Q_d 的长度比.当内存空间不足时,FAMC 计算当前 Q_c 和 Q_d 队列中数据页的数量,若二者的比例小于ω,则表明缓存的脏页数量较多,为防内存污染,需要将脏队列中处于 LRU 端的数据页替换出内存;反之,则从干净页队列选择牺牲页.

2.3 SSD数据管理

SSD 具有读写不对称特性,对于不同访问模式的负载,呈现出较大的性能差异.本文对多款不同类型 SSD 的读写性能进行实验测试,测试结果比较相近.图 2 和图 3 给出一款 SSD 在响应时间和访问吞吐方面的性能表现.

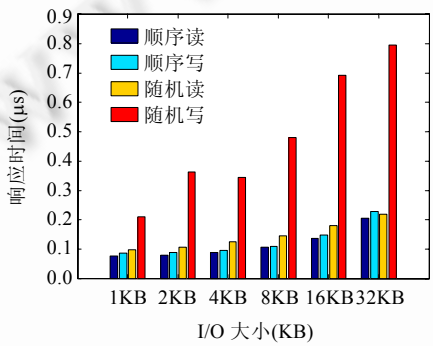


Fig.2 Average response time on different I/O sizes

图 2 不同 I/O 请求大小下响应时间对比

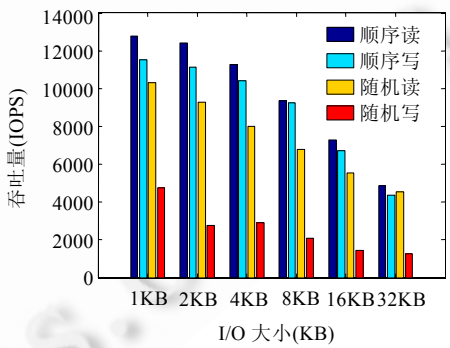


Fig.3 Throughput on different I/O sizes

图 3 不同 I/O 请求大小下访问吞吐量对比

从实验结果来看,SSD 随机写操作在响应时间和 I/O 吞吐等方面的性能远不及顺序访问,这是因为对 SSD 随机写操作会触发许多块擦除操作.此外,SSD 通过内部的闪存转换层对块内数据进行异位更新,无效旧数据会导致块内产生大量存储碎片,而频繁的垃圾回收操作也会影响 SSD 的读写性能.另一方面,与磁盘的读写特点不同,SSD 的随机读性能与顺序读性能的差别不是很大.针对 SSD 这种特有的 I/O 特性,FAMC 设计了对 SSD 友好的缓存策略,采用闪存敏感的数据存储和替换算法来减少对 SSD 的随机写操作,充分发挥其良好的随机读性能.第 2.3.1 节和第 2.3.2 节将分别介绍两种不同类型 SSD 的数据管理策略.从第 2.3.1 节开始,如无说明,文中出现的 M_SSD 和 S_SSD 分别指的是 MLC 型和 SLC 型 SSD.

2.3.1 M_SSD 数据管理

M_SSD 的存储容量比较大,价格相对较低,其读操作的性能与 S_SSD 相当,但 M_SSD 写操作,尤其是随机写性能较差.文献[14]对 SSD 写性能的测试结果表明:大范围地址空间的随机写操作会使系统性能急剧下降;而在小范围地址空间(4MB 存储范围)内,随机写性能与顺序写相当.因此,M_SSD 作扩展缓存应该尽量减少或避免

对 SSD 的随机写操作.一些工作基于文献[15]提出的 Log-Structure 管理策略,将内存置换出的页以顺序追加的方式写入 SSD.这种方法虽然可以避免随机写操作,但对于多级缓存架构来说,其缺点非常明显:首先,用 SSD 做内存扩展的主要目标是在内存缺页时能够利用其良好的读性能获取缓存在 SSD 的数据,而快速、有效地定位 SSD 中的数据是影响性能的重要因素.M_SSD 缓存的数据量远超过内存,在顺序追加模式下,查找目标数据需要消耗很多内存和 CPU 资源.文献[16]指出:在闪存存储系统中,CPU 代价与系统性能紧密相关.此外,顺序追加会导致很多无效数据大范围分布在 SSD,空间回收和数据合并需要占用大量的系统资源.在文献[7]中,作者通过性能测试发现:对 SSD 采用大块的 I/O 访问策略,有利于发挥 SSD 内部的并行读写机制,系统性能提升显著.基于上述特性,FAMC 将 M_SSD 中的数据以页块为单位组织成一个队列,每个页块的大小是 512KB.当内存缓冲区需要置换数据页时,队列中处于 LRU 端的页将被批量顺序写入 M_SSD 存储块.M_SSD 数据写入的过程见算法 2.基于块粒度的数据管理主要是考虑到闪存写前擦除的物理特性,因为闪存小范围的随机写操作可能会导致整个数据块在写入数据前执行擦除操作,擦除操作需要将块内原有的数据进行迁移,因此,在整个写操作过程中会引发较多的写操作.基于块的写操作模式可以在一定程度上降低数据块被擦除的机率,进而获得较好的 I/O 性能.本文通过实验对不同策略下的 SSD 物理写次数进行了统计(本文第 3.2.2 节),结果表明,基于块粒度的数据管理模式引发的物理写次数明显降低.

算法 2. write *MLC*(·).

```
1. if SSD is full then                          /*如果 SSD 没有可用存储空间*/
2.     evictSegment(·)
3. end if
4. do while curent segment is not full          /*将数据写入当前 segment*/
5.     get p from the LRU of Qc                 //从 Qc 选择处于 LRU 端的数据换出内存
6.     if p is not exist in SSD then
7.         write p to current segment
8.     end if
9. loop
```

算法的第 4 行~第 9 行将数据写入 SSD,当 SSD 存储空间不足时,算法第 2 行选择牺牲页替换出 SSD,以产生新的可用空间.已有的替换策略通常基于访问最近性,将最近很少被访问的数据页替换出去.事实上,在多级缓存体系中,因为存放在底层缓冲区的数据是由上一级缓存过滤下来的,数据访问的局部性特征不明显.文献[17]表明:访问频度较多的数据对系统缓存性能的贡献比访问最近性更大,缓存命中率更高.FAMC 综合考虑页块的访问频度和数据的有效性,设计实现了基于页块的替换策略.数据替换过程见算法 3.

算法 3. *evictSegment*(·).

```
1. Snum=total number of segment                 //Snum 是当前 SSD 中的 segment 数量
2. for   i=1 to Snum
3.     weight[i]=frenquency(i)/invalid_count(i)  //计算第 i 个 segment 的权值
4.     if periodfre(i)<T then                    /*对访问较少的 segment 加入衰减因子 f */
5.         weight[i]=weight[i]×f
6.     end if
7. next i
8. victim_segment=min(weight)
9. swap out victim_segment
```

算法 3 的第 2 行~第 7 行计算每个页块的置换权值.影响权值的因素包括访问频度、无效数据页的数量和时间衰减因子.其中,*frenquency*(*i*)用于计算页块 *i* 中数据页被访问的频度,高频访问的页块应留在 SSD 中;*invalid_count*(*i*)统计页块 *i* 内无效页的数量.这里,无效数据指的是从 M_SSD 读入内存的页被修改,变为脏页,

M_SSD中相应的原数据就变得不再有效.为了降低访问频度的累积效应,算法的第4行~第6行对页块的访问频度进行衰减,在一定时间段里,访问次数低于 T 的页块,其访问频度将被衰减降低.算法的第8行、第9行选择权值最小的 *segment* 替换出 SSD.因为 M_SSD 存放的是干净页,替换过程只需将数据擦除即可,碎片回收的效率较高.

2.3.2 S_SSD 数据管理

S_SSD 具有很好的随机读写性能,FAMC 用 S_SSD 缓存从内存置换出的脏页.S_SSD 中的数据以页为单位被组织成多个 LRU 队列:$Q_1,Q_2,\ldots,Q_n$,每个队列由置换优先级相同的数据页组成,队列 Q_j 中的数据比队列 Q_i 中的数据有更高的优先级驻留在 S_SSD($i<j$).数据替换时,算法首先定位当前优先级最低的队列,然后选择处于LRU端的页替换出S_SSD.S_SSD置换出的有效数据需要写回磁盘.如果将S_SSD中反复更新多次的数据页保留在S_SSD,则可以减少磁盘回写I/O的次数,进而提高系统的性能.为此,FAMC区别对待S_SSD的读和写操作,赋予S_SSD的写操作比读操作更大的访问频度.数据页 p 的优先级满足公式(2):

$$Q\_pri(p)=\log_2\left(p_{old}+p \text{ is dirty}?1:\log_2\frac{C_w}{C_r}\right) \tag{2}$$

其中,C_r 和 C_w 代表 S_SSD 的读写代价,p_{old} 表示页 p 之前被访问的次数.在 p 被访问命中时,由公式(2)计算该页的置换优先级,调整 p 进入相应的优先级队列.FAMC 同时考虑了访问频度的累积效应,在一段时间内,访问次数较少的数据页将会被调整到更低一级的队列.S_SSD 数据访问过程见算法4.

算法 4. *Access_S_SSD*(page *p*,operation *type*).

输入:*p* 是当前被请求写入的数据页;*type* 是操作类型.

```
1.  if type is read then                          /*处理来自内存的读操作*/
2.     p.frenquency++
3.     updatepriority(p)
4.     adjustqueue(·)
5.  else
6.     if SSD is not full then                    /*更新数据页的访问频度和置换优先级*/
7.        p.frenquency=p.frenquency+int(Cw/Cr)
8.        updatepriority(p)
9.        adjustqueue(·)
10.       Q=locatequeue(p)
11.       insert p to the tail of Q
12.    else                                       /*从优先级最低的队列选出牺牲页*/
13.       Q=findminpriority(·)
14.       remove q from Q
15.       insert p to the tail of Q
16.       write q to disk
17.    end if
18. end if
19. adjustqueue(·)
20.    for each p in a queue                      /*调整队列的置换优先级*/
21.       if periodfre(p)<θ then
22.          k=p.priority
23.          remove p from its Q[k−1]
24.          insert p into the head of Q[k−2]
```

25. end if

26. next p

算法4的第1行~第4行处理对S_SSD的读操作;第2行、第3行更新p的置换优先级;第4行的*adjustqueue*(·)对近期访问频度较低的页进行降级处理,调整过程如算法的第19行~第26行所示.参数θ用来表示访问频度阈值,如果在一定时间段内,数据页p的访问次数低于这一阈值,则对该数据页进行优先级的降级处理,调整该页进入比它低一级的数据队列.这种策略可以在一定程度上降低短时间内频度累积效应,减少缓存污染.对S_SSD的写请求处理过程由算法的第5行~第17行完成,如果S_SSD存储空间未满,第7行对C_w/C_r取整作为p增加的访问频度.算法的第8行~第11行更新调整p的置换优先级,将p写入相应队列的MRU端;如果S_SSD存储空间已满,算法的第13行~第16行定位优先级最低的队列,并将处于LRU位置的页置换出S_SSD.S_SSD数据管理采用多LRU队列结构,最近访问最多的数据会被优先保留在SSD.另外,算法同时区分内存对SSD的读和写请求,写操作频繁的数据拥有更高的置换优先级,这在很大程度上可以减少脏页回写引发的磁盘I/O数量.

3 实验结果及分析

本文对FAMC多级缓存策略的有效性进行了验证,与文献[10]提出的lazy-cleaning(LC)策略和文献[11]提出的FaCE策略进行了性能对比.实验所用的两个数据集基于TPC-C标准,采用BenchmarkSQL运行PostgreSQL数据库获得.数据集的统计信息见表2.

Table 2 Statistics of the datasets used in our experiment

表2 实验所用数据集的统计信息

数据集	访问次数	读次数	写次数	读写比
T1	2 193 476	1 805 230	388 246	82.3%/17.7%
T2	1 988 563	1 262 737	725 826	63.5%/36.5%

3.1 实验环境

本文基于DiskSim 4.0[18]和微软发布的针对DiskSim 4.0的SSD Model[19]设计实现了一个多级缓存实验仿真平台.DiskSim4.0是一个准确度高、配置灵活的磁盘存储系统仿真工具.微软开发的SSD Model具有仿真准确、可配置性高等特点.实验代码使用C实现,并运行在Ubuntu操作系统上,页面大小为4KB.缓冲区大小设定为4MB,M_SSD和S_SSD的容量比为5:1.在Disksim的Cache Devices模块分别加入SLC SSD和MLC SSD模拟设备,其存储性能参数基于Intel X25-E SSD和Intel X25-M SSD.在Cachemem和Cachedev模块中,分别实现了FAMC和LC的各级缓存策略.

3.2 性能对比

为了对比FAMC的系统性能,首先将两个trace文件分别在单一存储设备上运行,实验结果如图4所示.

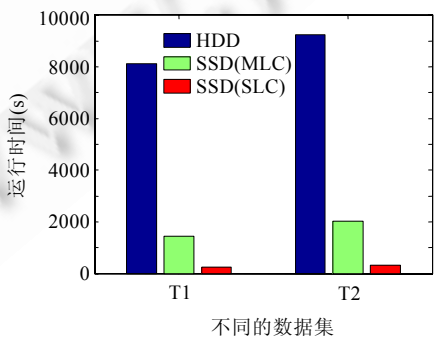


Fig.4 Comparison of running time for different storage devices when executing T1~T2

图4 不同存储设备下T1~T2的运行时间比较

从图 4 中可以看出:SSD 在两种数据集上运行的时间都远低于磁盘,T1 和 T2 在 SLC 型 SSD 上的运行时间分别只有 MLC 型 SSD 的 17.1%和 15.4%.

3.2.1 运行时间

最近性优先(LRU)[20]和先进先出(FIFO)[21]是缓冲区管理最为经典的算法,目前已被许多应用系统采用.为了验证 FAMC 策略的有效性,本文实现了 FaCE 和 LC 策略,并将它们与 FAMC 进行性能对比.其中,LC 缓存策略采用的是 LRU-2 算法,而 FaCE 则采用的是基于 FIFO 的多版本共存策略.实验对不同 SSD 和内存容量比下数据集的运行时间进行了对比.实验结果如图 5 和图 6 所示,其中,LC-S 表示用 S_SSD 做 LC 的扩展缓存,LC-M 表示用 M_SSD 做 LC 的扩展缓存,而 FaCE-S 和 FaCE-M 分别表示以 S_SSD 和 M_SSD 为扩展缓存的运行时间.

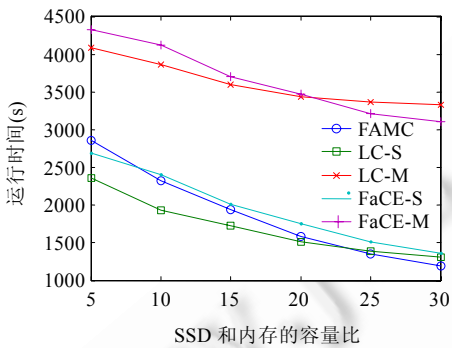


Fig.5 Comparison of time when executing T1

图 5 运行 T1 的时间比较

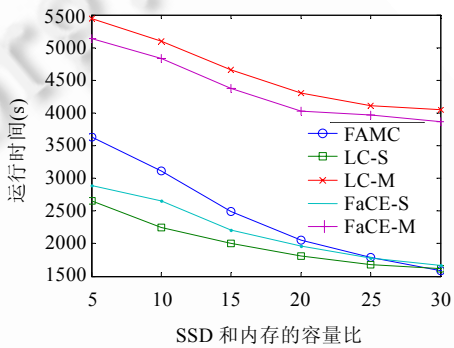


Fig.6 Comparison of time when executing T2

图 6 运行 T2 的时间比较

图 5 描述了数据集 T1 在 3 种缓存策略下的运行时间.从实验结果来看:在不同的 SSD 容量下,数据集在 FAMC 上的运行时间比 LC-M 和 FaCE-M 均有大幅降低.当充当内存扩展的固态盘为 S_SSD 时,在较小 SSD 容量下,基于 M_SSD 和 S_SSD 混合使用的 FAMC 架构的执行时间会略高于 LC-S 和 FaCE-S.随着 SSD 容量的增大,FAMC 的执行时间在不断减少,当 SSD 与内存容量比超过 25 时,FAMC 策略的执行性能优于基于 S_SSD 的 LC-S 和 FaCE-S 算法.另外,根据表 1 所显示的不同类型 SSD 价格和实验中不同类型 SSD 的容量比可以得出,FAMC 使用的 SSD 的价格只占全 S_SSD 模式下的 29%.因此,FAMC 在获得系统性能提升的同时,也有效降低了存储成本,这有助于降低大规模数据中心的硬件购置代价.对于实验结果做如下分析:LC 采用 LRU-2 替换算法管理 SSD 中的数据,维护 LRU 列表会导致较多的随机写操作,频繁的随机写操作占据了大量系统资源,进而影响缓存系统的性能.特别是 LC-M 策略,从图 5 可以看出:随着 M_SSD 容量的增大,数据集运行时间性能提升并不明显.FaCE-M 与 LC-M 相比,因为采用顺序追加的方式,写操作代价相对于 LC-M 的随机操作模式会明显降低,当 SSD 容量增大时,FaCE-M 执行时间会低于 LC-M 策略.需要注意的是:对比单一设备的运行时间(由图 4 示),当 SSD 和内存容量比达到 25 时,FAMC 运行时间低于 S_SSD 模式的执行时间.对于数据集 T2,尽管其写操作比重比数据集 T1 更大,FAMC 系统仍然取得了较好的性能.由图 6 可以看出:当 SSD 和内存的容量比达到 30 时,FAMC 和 LC-S 和 FaCE-S 的执行时间基本相当,但 FAMC 花费的存储代价远低于 LC-S 和 FaCE-S.这主要是因为 FAMC 采用不同的缓存策略处理干净页和脏页,SSD 数据替换算法充分考虑了不同 SSD 产品的读写特性和页面状态.相比之下,对于写操作较多的访问负载,LC 策略引发的随机写操作严重制约了系统性能的提升.在数据集 T2 下,FaCE-M 的性能始终优于 LC-M,其主要原因在于:FaCE 采用多版本共存策略,当内存中的数据页置换到 SSD 时,FaCE 将该数据页顺序地写入,旧版本数据则被置为无效.这种策略避免了对 SSD 的随机写操作,对于更新操作密集的负载,对 SSD 的写操作代价较小,因此执行时间要低于 LC-M 算法.综合两种负载,FAMC 在系统性能提升和硬件代价两方面获得了很好的权衡,缓存策略的性价比更高.

3.2.2 SSD 写操作次数

写操作性能差,一直是制约 SSD 数据访问性能提升的重要因素.减少对 SSD 的写操作,有利于提高系统性

能,延长 SSD 寿命.本文对 FAMC,LC-S 和 FaCE-S 这 3 种架构在实验过程中产生的 SSD 物理写次数进行了统计分析,这里的写操作指的是在 SSD 设备上运行数据集产生的总的物理写,其中包括数据替换、块擦除等操作.图 7 和图 8 给出了两种数据集在 FAMC,LC-S 和 FaCE-S 架构下总写操作次数的对比情况.

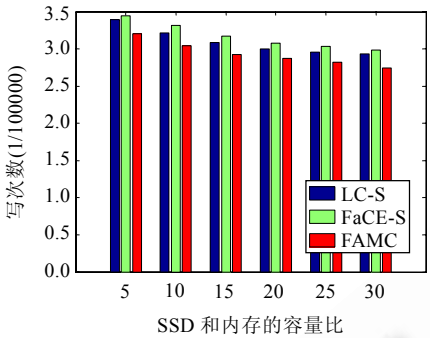


Fig.7 Comparison of SSD write when executing T1

图 7 运行 T1 的 SSD 写次数比较

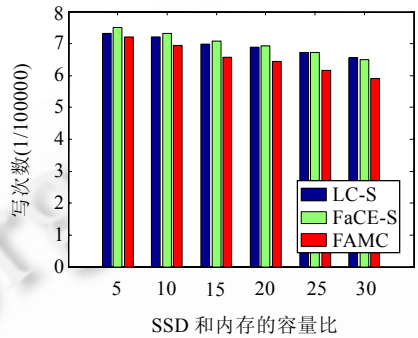


Fig.8 Comparison of SSD write when executing T2

图 8 运行 T2 的 SSD 写次数比较

与 LC-S 和 FaCE-S 相比,FAMC 采用适合 SSD 读写特性的数据替换算法,系统运行引发的 SSD 物理写操作写低于 LC-S 和 FaCE-S.造成这一现象的主要原因在于:M_SSD 模块充分考虑了 SSD 的读写不对称特性,通过小范围随机写,减少了数据迁移、数据合并等引发的写操作;而 S_SSD 采取的基于频度的替换算法保证了较高的访问命中率,减少了数据写入或换出 SSD 的次数.对于数据集 T2,FaCE-S 采用的顺序追加写模式减少了数据块擦除时有效数据的迁移代价,当 SSD 容量增大时,其物理写次数略低于 LC-S 策略.

3.2.3 SSD 访问命中率

SSD 访问命中率是衡量多级缓存系统的一个重要指标,图 9 和图 10 分别描述在不同的 SSD 与内存容量比下,FAMC,LC-S 和 FaCE-S 这 3 种缓存策略的 SSD 访问命中率.

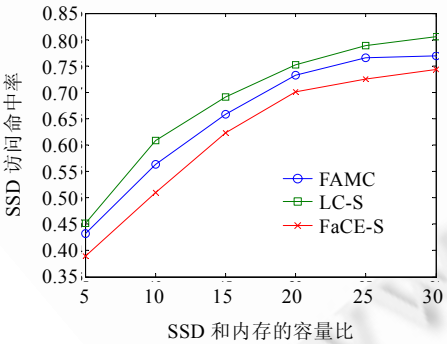


Fig.9 Comparison of hit ratio when executing T1

图 9 运行 T1 的命中率比较

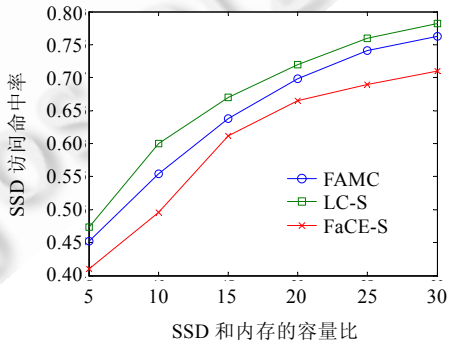


Fig.10 Comparison of hit ratio when executing T2

图 10 运行 T2 的命中率比较

从实验结果来看,FAMC 的 SSD 命中率在缓存容量较小时低于 LC-S.这是因为 FAMC 中 M_SSD 模块采用较粗粒度的替换策略,在一定程度上影响了对干净页的缓存命中率.当 SSD 与内存容量比达到 20 以后,两种策略的 SSD 命中率比较接近.对于二级缓存策略,FAMC 基于频度的替换算法比 LC-S 侧重于最近性原则更加有效,这保证了 FAMC 具有较高的缓存命中率.需要说明的是:虽然 FAMC 中 SSD 访问命中率略低于 LC-S,但

FAMC 采用对 SSD 友好的写操作模式,数据写入或替换出 SSD 引发的物理写操作次数明显降低(由图 7 和图 8 可见),SSD 替换数据的代价小;另一方面,FAMC 在置换 SSD 中的牺牲页时会选择将更多的脏页保留在 SSD,这在很大程度上减少了脏页写回磁盘的数量,缓解了磁盘 I/O 对系统性能的影响.FaCE-S 策略表现出较低的访问命中率,这主要是因为 FaCE-S 采用顺序追加的方式,随着数据被频繁的修改,SSD 中会因为有较多的无效数据而使得可用存储空间降低,进而影响了数据访问的命中率.

4 结 论

本文提出了一个高性价比的多级缓存管理策略——FAMC,通过在内存和磁盘之间加入不同类型的 SSD 作扩展缓存,可以有效地降低系统响应时间.FAMC 针对不同 SSD 产品的硬件特性,设计不同的策略来缓存从内存替换出的数据页,采用 SSD 敏感的算法和数据结构组织和替换缓存的数据,SSD 写操作次数明显降低,系统的可靠性和可用性得到了保障.将 SSD 引入现有的存储体系,是当前数据管理研究领域的一个热点,发挥 SSD 的非易失性,合理利用暂存的数据,有利于提高系统恢复和并发执行的能力.

致谢 在此,我们向对本文的工作给予支持和建议的老师和同学表示感谢.

References:

[1] Canim M, Bhattacharjee B, Mihaila GA, Lang CA, Ross KA. An object placement advisor for DB2 using solid state storage. Proc. of the VLDB Endowment, 2009,2(2):1318−1329. [doi: 10.14778/1687553.1687557]

[2] MySpace-Uses-Fusion. 2006. http://www.fusionio.com/case-studies/myspace-case-study.pdf

[3] IT@Intel white paper. 2012. http://www.intel.com/content/dam/www/public/us/en/documents/best-practices/solid-state-drive-caching-differentiated-storage-paper.pdf

[4] IBM solidDB. 2010. ftp://public.dhe.ibm.com/software/data/soliddb/info/7.0/man/zh.CN/GettingStartedGuide_cn7005.pdf

[5] Oracle timesten. 2011. http://www.oracle.com/technetwork/database/database-technologies/timesten/overview/timesten-exalogic-wp-july2011-487843.pdf?ssSourceSiteId=ocomen

[6] Mesnier MP, Chen F, Akers JB, Luo T. Differentiated storage services. In: Proc. of the ACM 23rd Int'l Symp. on Operating Systems Principles (SOSP 2011). Cascais: ACM Press, 2011. 57−70. [doi: 10.1145/2043556.2043563]

[7] Chen F, Koufaty AD, Zhang XD. Understanding intrinsic characteristics and system implications of flash memory based solid state drives. In: Proc. of the ACM SIGMETRICS 11th Int'l Conf. on Measurement and Modeling of Computer Systems (SIGMETRICS 2009). ACM Press, 2009. 181−192. [doi: 10.1145/1555349.1555371]

[8] Micron: C200 1.8-inch SATA NAND flash SSD .2007. http://download.micron.com/pdf/datasheets/realssd/realssd_c200_1_8.pdf

[9] Canim M, Mihaila GA, Bhattacharjee B, Ross KA, Lang CA. SSD bufferpool extensions for database systems. Proc. of the VLDB Endowment, 2010,3(2):1435−1446.

[10] Do J, Zhang DH, Patel JM, De Witt DJ, Naughton JF, Halverson A. Turbocharging DBMS buffer pool using SSDs. In: Proc. of the ACM SIGMOD Int'l Conf. on Management of Data (SIGMOD 2011 and PODS 2011). ACM Press, 2011. 1113−1124. [doi: 10.1145/1989323.1989442]

[11] Kang WH, Lee SW, Moon B. Flash-Based extended cache for higher throughput and faster recovery. Proc. of the VLDB Endowment, 2012,5(11):1615−1626. [doi: 10.14778/2350229.2350274]

[12] Park SY, Jung D, Kang JU, Kim J, Lee JW. CFLRU: A replacement algorithm for flash memory. In: Proc. of the IEEE 9th Int'l Conf on Compilers, Architecture, and Synthesis for Embedded Systems (CASES 2006). IEEE Press, 2006. 234−241. [doi: 10.1145/1176760.1176789]

[13] Li Z, Jin PQ, Su X, Cui K, Yue LH. CCF-LRU: A new buffer replacement algorithm for flash memory. IEEE Trans. on Consumer Electronics, 2009,55(3):1351−1359. [doi: 10.1109/TCE.2009.5277999]

[14] Bouganim L, Jónsson BT, Bonnet P. uFLIP: Understanding flash I/O patterns. In: Proc. of the ACM Int'l Conf. on Innovative Data Systems Research (CIDR 2009). ACM Press, 2009. 1−12.

[15] Rosenblum M, Ousterhout JK. The design and implementation of a log-structured file system. ACM Trans on Computer Systems, 1992,10(1):26−52. [doi: 10.1145/146941.146943]

[16] Lee SW, Moon B, Park C, .Kim JM. Kim SW. A case for flash memory SSD in enterprise database applications. In: Proc. of the ACM SIGMOD Int'l Conf. on Management of Data (SIGMOD 2008 and PODS 2008). ACM Press, 2008. 1075−1086. [doi: 10.1145/1376616.1376723]

[17] Zhou YY, Philbin J, Li K. The multi-queue replacement algorithm for second level buffer caches. In: Proc. of the 26th Annual USENIX Technical Conf. (USENIX 2001). ACM Press, 2001. 91−104.

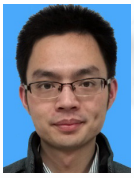
[18] Bucy J, Schindler J, Schlosser S, Ganger G. DiskSim 4.0. 2010. http://www.pdl.cmu.edu/DiskSim

[19] Agrawal N, Prabhakaran V, Wobbe T, Davis JD, Manasse MS, Panigrahy R. Design tradeoffs for SSD performance. In: Proc. of the 33th Annual USENIX Technical Conf. (USENIX 2008). ACM Press, 2008. 57−70.

[20] O'Neil EJ, O'Neil PE, Weikum G. The LRU-*K* page replacement algorithm for database disk buffering. In: Proc. of the ACM SIGMOD Int'l Conf. on Management of Data (SIGMOD'93). ACM Press, 1993. 297−306. [doi: 10.1145/170035.170081]

[21] Johnson T, Shasha D. 2Q: A low overhead high performance buffer management replacement algorithm. In: Proc. of the ACM Int'l Conf. on Very Large Data Bases (VLDB'94). ACM Press, 1994. 439−450.

王江涛(1978－),男,山东莱阳人,博士生,CCF 学生会员,主要研究领域为闪存数据库系统,混合存储数据管理系统.
E-mail: jiangtaow@ruc.edu.cn



赖文豫(1989－),男,硕士,主要研究领域为数据库存储管理,查询优化.
E-mail: xiaolai913@gmail.com

孟小峰(1964－),男,博士,教授,博士生导师,CCF 会士,主要研究领域为网络数据管理,云数据管理,移动数据管理,社会计算,闪存数据库,隐私保护.
E-mail: xfmeng@ruc.edu.cn