闪存数据库:现状、技术与展望

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)

摘 要 随着闪存存储技术的发展,闪存已经广泛应用于各种移动设备、PC 机和服务器中.作为一种完全不同于磁盘的新型存储介质,闪存具有非易失、高速读写、抗震、低功耗、高存储密度等物理特性,这使得基于闪存的数据管理问题成为新的挑战.数据库系统是数据管理的重要技术,将现有的数据库系统直接移植到闪存上并不能充分发挥其硬件特性,设计实现基于闪存的数据库系统是当前的一个研究热点.文中介绍了闪存的特性和闪存转换层;总结了缓冲区、索引、查询和事务等数据库关键技术;讨论了基于闪存的混合存储数据管理.最后,基于该领域亟待解决的诸多问题,指出了未来的研究方向.

关键词 闪存;缓冲区;索引;查询;混合存储

中图法分类号 TP311 **DOI 号** 10.3724/SP.J.1016.2013.01549

Flash-Based Database: Studies, Techniques and Forecasts

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

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

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

Abstract With the growing popularity of flash memory, flash has been widely used in various types of applications, such as mobile devices, PC machines, and servers. Flash, as a new type of storage media unlike disk, has its inherent properties which include non-volatility, high access speed, shock resistance, lower power, and high storage density etc. These properties, however, will bring new challenges for data management. Although disk-based database management system has the powerful ability for managing data, its current techniques cannot make full use of high I/O performance of flash memory if we transfer it to flash without modification. Therefore, it is a hot topic to design and implement flash-based database systems. In this survey, the new properties and translation layer of flash is introduced firstly. Secondly, several database critical techniques on buffer, index, query optimization, and transaction processing are surveyed and classified. Thirdly, we discuss some research issues on flash-based hybrid storage systems. At last, based on the analyzed problems, suggestions for future research works are put forward.

Keywords flash memory; buffer; index; query; hybrid storage

1 引 言

随着社交网络、物联网等新技术和应用的快速发展,数据产生的规模和速度呈现爆炸式增长,海量数据处理给计算机系统性能带来巨大挑战.作为数据主要存储介质的磁盘已经越来越不能满足实际应用对存储带宽的需求.在过去的几十年,CPU 的处

收稿日期:2012-12-24;最终修改稿收到日期:2013-04-17.本课题得到国家自然科学基金(61070055,91024032,91124001)、国家"八六三"高技术研究发展计划项目基金(2012AA010701,2013AA013204)、中国人民大学科学研究基金(11XNL010)资助.**王江涛**,男,1978 年生,博士研究生,中国计算机学会(CCF)会员,主要研究方向为闪存数据库系统、混合存储系统.E-mail:jiangtaow@ruc.edu.cn.**赖文豫**,男,1989 年生,硕士研究生,主要研究方向为闪存数据库查询优化.**孟小峰**,男,1964 年生,博士,教授,博士生导师,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、闪存数据库系统、隐私保护.

理速度提高近 600 倍，而磁盘的读写速度只提升了不足 10 倍，低速的磁盘已经成为制约系统性能提升的瓶颈. 随着多核、GPU 等高性能处理器的出现，这一现象必然会更加突出. 磁盘自身机械寻道特性使其性能难有大幅提升，数据处理急切需要新型高效的存储设备来提高存储系统的性能. 诞生于 20 世纪 80 年代末的闪存存储器(Flash Memory)为解决这一问题提供了有效途径. 闪存是一种全电设备，通过电子电路来读取数据，具有非易失、极高的读写速度、抗震、低功耗、体积小等特性，目前已经广泛应用于嵌入式系统、航空航天、消费电子等领域[1-2]. 闪存的读写速度超过磁盘百倍以上，随着制作工艺的发展，闪存的容量不断增大，应用领域开始逐步扩展到高吞吐、数据访问密集的企业级应用环境. 图灵奖获得者 Gray 先生就曾预测："就像磁盘取代磁带一样，闪存将会取代磁盘"[3]. 数据库系统是数据管理的重要技术，现有数据库系统大都基于磁盘存储进行设计和优化，闪存有与磁盘不同的物理特性，将数据库系统直接运行在闪存上不能充分发挥其优良性能. 探讨适合闪存特性的数据库技术是当前数据管理领域的研究热点.

本文第 2 节介绍闪存存储特性、固态硬盘和闪存转换层；第 3 节对闪存数据库关键技术进行分析归类；第 4 节介绍基于闪存的混合系统数据管理相关研究；第 5 节展望未来研究工作；第 6 节对全文做出总结.

2 闪存存储

2.1 闪存类型

根据制作工艺，闪存存储器可以分为 NOR 型和 NAND 型两种. NOR 型闪存可以按位进行访问，具有可靠性高、随机读取速度快的优势，但擦除和写操作速度较慢、容量小，主要用于存储可执行的程序代码. NAND 闪存的容量大，适合进行数据存储. 根据芯片晶胞所能存储的比特位数，NAND 闪存又可分为单级晶胞(SLC)和多级晶胞(MLC)两类，SLC 每单元存储 1 个比特位，MLC 每单元存储多个比特位. 文中提及的闪存设备若不是特别注明都是指 NAND 型闪存.

2.2 闪存特性

一个 NAND 型闪存芯片通常由若干个块组成，每个块又由若干页组成. 例如，容量为 1 GB 的三星 K9WAG08U1A① 闪存芯片包含 8192 个块，每个块由 64 个页组成，每一个页由数据区和备用区组成，2 KB 的数据区用于存储用户数据，64 B 的备用区用来存储校验、逻辑页地址等信息. 闪存具备读、写和擦除 3 种操作，页是闪存的基本读写单位，重写数据前必须进行擦除，擦除操作以块为单位，执行时间和能耗远高于读写操作. 在页被擦除前，SLC 型闪存可以对同一个数据页进行多次写操作，最小写单元为 512 个字节. 闪存芯片的读写方式与磁盘截然不同，二者的 I/O 操作性能对比见表 1.

表 1 闪存与磁盘 I/O 性能对比[4]

类型	时间/s		
	读(2 KB)	写(2 KB)	擦除(128 KB)
磁盘	12.7 ms	13.7 ms	N/A
NAND 型闪存	80 μs	200 μs	1.5 ms

总的来说，闪存不同于磁盘的特性包括以下几点：

(1) 无机械延迟. 作为一种纯电子设备，闪存没有机器延迟，随机访问和顺序访问的开销相当，具有很高的随机读性能；

(2) 读写不对称. 闪存对不同类型访问操作表现的性能差别很大. 一般来说，读速度很快，写入数据时，因为需要通过加压的方式对存储单元进行电子填充，所以速度较慢；

(3) 异位更新. 与磁盘的原位更新不同，对闪存数据进行重写需要先执行擦除操作，即便是只更新数据块中的一条数据也需要将整个块擦除. 频繁的擦除操作会使系统性能显著降低. 因此，闪存设备在更新数据时并不会直接在原位进行更新，而是先将原数据置为无效，然后把修改后的数据写到一个新的空闲页；

(4) 擦除次数有限. 闪存芯片的块擦除次数是有限制的. 通常 SLC 闪存支持 10 万次擦除操作，MLC 闪存数据存储密度高，可擦除次数在 1 万次左右. 超过一定擦除次数的闪存单元将不再可用；

(5) 低能耗. 与磁盘相比，闪存的能耗更低，每 GB 读数据能耗只有磁盘的 2%，写操作能耗不足磁盘的 30%. 闪存的出现为建设绿色数据中心以及低能耗数据管理系统提供了有力支持.

闪存有比磁盘更加复杂的硬件特性，传统的磁盘数据库技术在闪存上并不适用，设计适用于闪存的结构、算法和应用是闪存数据管理必须要解决的

① http://www.alldatasheet.com/pdf/177488/SAMSUNG

一个重要问题.

2.3　固态硬盘及闪存转换层

基于闪存的固态硬盘(Solid State Drive，SSD)由闪存芯片、控制器和闪存转换层(Flash Translation Layer，FTL)组成.它对外提供和磁盘相同的I/O接口，存储性能远超过磁盘.近十年来随着闪存芯片容量的增加和价格的下降，集成了多块闪存芯片的固态硬盘已经被认为是最有可能替代磁盘的新一代数据存储载体.个人计算机、服务器和企业级数据管理中心大量使用固态硬盘来提高数据存储性能[5]①②.为了屏蔽闪存与磁盘不同的操作特性，固态硬盘通过闪存转换层为上层文件系统提供通用的读写接口，系统应用不需要任何修改便可直接在固态硬盘上运行.闪存转换层是封装在闪存芯片和文件系统之间的一个软件层，其结构如图1所示.闪存转换层包括3个基本功能：

(1)地址映射.闪存转换层的Allocator模块负责为I/O请求分配可用的空闲空间，解决闪存异位更新造成的地址变换问题；

(2)磨损平衡.闪存单元的擦除次数是有限制的，超过限定擦除次数的单元将成为磨损块而无法使用.数据访问的不均衡性可能导致闪存局部存储区因更新频繁而变成磨损块.闪存转换层中的Wear Leveler模块采用数据迁移等方法平衡存储单元的擦除次数实现闪存整体的损耗均衡；

(3)垃圾回收.闪存转换层的Cleaner模块用于回收无效的旧数据页.垃圾回收首先选择待回收的块，把块中有效的数据写入新的块，然后擦除无效块实现空间的循环再利用.

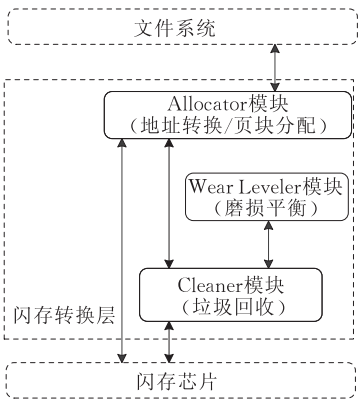


图1　闪存转换层结构

3　闪存数据库系统关键技术

闪存有与磁盘完全不同的物理特性，用闪存直接替换磁盘进行数据管理不能最大化发挥闪存的优良性能，因此有必要设计基于闪存的数据库系统.本节首先阐述研究闪存数据库系统的必要性，然后分别讨论闪存数据库系统研究的几个关键技术.

3.1　闪存数据库系统研究的必要性

为了对比闪存设备对数据库系统的影响，我们将PostgreSQL 8.4.9数据库系统迁移至固态硬盘并进行TPC-C测试.测试环境：CPU为2 Duo CPU E8300 2.8GHz，内存2GB，缓冲区为400MB，操作系统为Ubuntu，操作系统和数据集分别安装在两块80GB Intel SSD上，测试结果见表2.

表2　磁盘和固态硬盘事务处理吞吐性能对比

并发用户	磁盘	SSD	性能提升/%
4	435	5827	13.4
8	608	5436	8.9
16	506	5202	10.2
32	414	4127	9.9

实验表明PostgreSQL在固态硬盘上的事务处理性能提高10倍左右，相比固态硬盘对磁盘百倍以上的读写速度，数据库系统显然没能充分发挥闪存优良的存储性能.文献[4，6]基于闪存对数据库查询处理性能进行了测试，结果只提高了几倍.造成这一现象的根本原因在于现有的数据库系统都是基于磁盘设计，直接运行在闪存上不能充分发挥闪存的物理特性.以查询优化为例，磁盘随机读操作代价是顺序读的130倍，查询优化算法优先选择顺序读写来代替随机读操作，闪存写操作比读操作的代价高，而其随机读代价只是顺序读的2.5倍，现有查询优化算法无法发挥其极佳的随机读性能.设计并实现适合闪存读写特性的数据库系统十分必要.目前IBM、Oracle和微软等许多知名数据库厂商都在它们的数据库产品中添加了针对闪存的应用③④.闪存数据库系统研究必须分析现有技术在闪存数据管理上的缺陷，从系统性和通用性等角度设计适合闪存特性的算法、结构和应用，以充分发挥闪存的性能优势.基于此，我们提出闪存数据库系统框架，如图2所示.

查询处理应该设计基于闪存的代价模型并对查询编译、查询计划生成、路径选择等环节进行优化；存储和索引管理应针对写前擦除特性对数据进行有

① http：//www.fusionio.com/case studies/myspace
② http：//www.facebook.com
③ http：//www.01.ibm.com/software/data/soliddb
④ http：//www.oracle.com/us/products/database/timesten/index.html

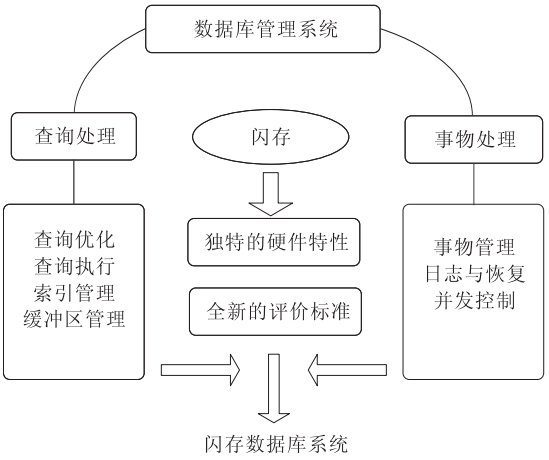


图 2 闪存数据库系统框架

效组织,提高数据检索的效率;缓冲区管理需要考虑闪存读写不对称等特性,在保证较高访问命中率的同时尽量减少对闪存的写操作;事务处理应该发挥闪存高速随机读性能,在日志、恢复和并发控制处理过程中既能保证事务的 ACID 特性,又能细化并发粒度提高事务处理的吞吐量. 缓冲区管理、索引技术、查询处理、事务处理等是构建闪存数据库系统的关键技术.

3.2 缓冲区管理

缓冲区是数据库系统的核心组件,利用数据访问存在局部性特征将最近和经常访问的数据存储在内存上,以快速响应 CPU 的读写请求. CPU 发出对数据页 p 的访问请求,缓冲区管理做如下处理:

(1)查阅页表判断 p 是否在缓冲区,若存在则访问命中,读取 p 并执行对该页的操作;否则访问脱靶(缺页)执行(2);

(2)系统检查缓冲区当前可用空间,若有空间则为请求页分配一个空闲页,然后从磁盘读入页 p,执行对 p 的操作,否则执行(3);

(3)由替换算法选择一牺牲页 q 置换出内存,若 q 是修改页(脏页),则需写回外存,读 p 进内存.

在可用缓冲区大小确定的情况下,置换代价是衡量缓冲区算法性能的首要因素. 页 p 的置换代价可由式(1)来表示:

$$Cost(p)=P_{\text{miss}}\times C_{\text{read}}+P_{\text{miss}}\times P_{\text{dirty}}\times C_{\text{write}} \quad (1)$$

其中,P_{miss} 代表 p 脱靶的概率;C_{read} 和 C_{write} 分别表示在外存上读取和写入 p 的时间;P_{dirty} 表示 p 为脏页的概率. 式(1)可转换为式(2)

$$\begin{aligned} Cost(p)&=P_{\text{miss}}\times\left(1+P_{\text{dirty}}\times\frac{C_{\text{write}}}{C_{\text{read}}}\right)\\ &=P_{\text{miss}}\times(1+P_{\text{dirty}}\times\omega) \end{aligned} \quad (2)$$

其中 ω 表示对外存执行写和读操作的代价比. P_{miss}、P_{dirty} 和 ω 是影响置换代价的要素,对磁盘而言,ω 取值近似等于 1,因此基于磁盘的缓冲区管理算法主要关注 P_{miss} 对置换代价的影响,保持高访问命中率是缓冲区管理策略的主要目标. 经典缓冲区管理算法[7-8]根据数据访问频度和最近性设计页面替换策略. 闪存的读写代价是不对称的,其 ω 的取值受产品类型和访问形式的影响比较大,大小通常超过 5. 闪存数据库缓冲区管理必须同时考虑 P_{miss} 和 P_{dirty} 对置换代价的影响,单纯追求高命中率的设计原则对闪存未必有效,高命中率不一定带来高 I/O 性能. 设计高效的闪存缓冲区管理算法应该满足以下 3 个原则:(1)适当减少对脏页的置换,降低 I/O 代价;(2)保持相对较高的命中率,减少缺页;(3)优化写操作,避免大范围的随机写操作,提高闪存的使用寿命.

根据置换代价的粒度,现有闪存数据库缓冲区替换策略可分为基于页级和基于块级两类.

3.2.1 页级代价

基于页级代价的替换策略在选择牺牲页时考虑置换一页的开销,代价低的数据页优先被置换.

(1)CFLRU

CFLRU[9]是最早提出的闪存数据库缓冲区管理算法. CFLRU 在缓冲区发生缺页中断时优先选择干净页(Clean Page)进行替换,脏页(Dirty Page)被留在缓冲区以减少置换代价. 算法思想见图 3.

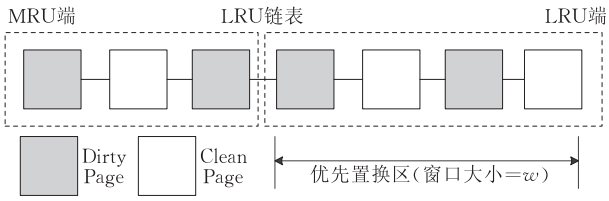


图 3 CFLRU 置换策略示例

CFLRU 将缓冲区中的数据页以 LRU 方式组织成队列,在队列的 LRU 端选择部分数据页作为优先置换区,替换算法从优先置换区按 LRU 顺序选择一干净页作牺牲页,如果没有干净页则选择位于 LRU 端的脏页,优先置换区的大小由参数 ω 来控制. CFLRU 方法可以有效减少脏页写回闪存的次数,但缓冲区中冷的脏数据页可能会因为长时间占据缓存而造成内存污染. 这里的冷热指的是数据在最近时间内被访问的情况. 针对 CFLRU 的问题,文献[10-13]通过设置冷热标识位等方法优化了替换策略对脏页的选择,命中率有所提高.

(2) FOR

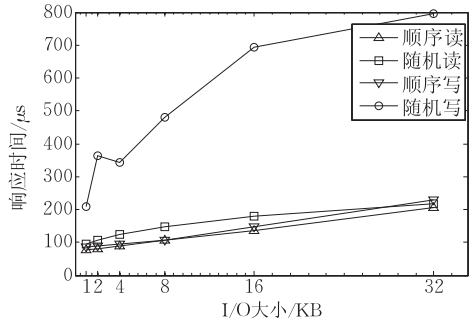
仅从访问的最近性或访问频度来区分数据页的冷热程度不能很好地反映实际应用中数据访问模式的变化. FOR[14] 基于最近性和访问频度提出了一个数据冷热模型：

$$Hotness(p)=\alpha\times IOD_p+(1-\alpha)\times OR_p \quad (3)$$

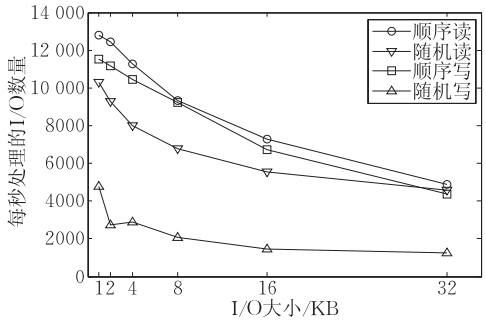
其中，IOD_p 和 OR_p 分别表示数据页 p 的访问频度和最近性；α 是一个可控参数，用于动态调节频度和最近性在热度模型中所占的比重；IOD_p 是指对 p 最近的两次访问之间不同数据页操作的次数；OR_p 是指对 p 的最近一次访问距离当前访问之间不同数据页的操作次数. 数据访问列表记录数据访问和替换的历史信息，由列表可以计算出 IOD_p 和 OR_p. FOR 根据 $Hotness(p)$ 将缓冲区分为冷热两个区，二者长度比满足 C_{read}/C_{write}. 置换算法先从冷数据区选择牺牲页，为了避免数据在变热前频繁被换出内存，FOR 通过冷数据阈值 R_{cold} 来保证最少冷数据页的数量，当冷数据页的数量低于 R_{cold}，热区中较冷的数据页将补充到冷数据区.

3.2.2　块级代价

闪存对读写操作是非常敏感的，对于不同负载访问模式表现出极大的性能差异. 为此，我们分别测试了 5 款不同品牌或型号固态硬盘的读写性能. 图 4 给出了其中一款产品的性能对比.



(a) 响应时间



(b) 吞吐量

图 4　SSD 读写性能对比

从实验结果可以看出，闪存随机写操作的性能远不及顺序写. 造成这一问题的主要原因包括：(1) 随机写操作会触发更多的块擦除操作，擦除操作的代价昂贵；(2) 闪存是通过内部 FTL 软件层对块内原数据进行异位更新，无效旧数据导致块内产生大量存储碎片，垃圾回收的代价比较大，而存储碎片也会使逻辑地址与物理地址不再连续进而影响数据预读性能. 文献[15]曾指出存有大量存储碎片的闪存其读写性能只有原来的 30%. 页级替换策略可能导致闪存产生较多存储碎片而引发频繁的随机写操作. 块级替换策略考虑到闪存良好的顺序写性能，以聚簇的方式将缓冲区中地址相近的数据页分组，然后以组为单位批量将数据换出内存.

(1) FAB

FAB[16] 基于 LRU 算法以数据块的形式组织缓冲区数据，每一个块聚集属于同一闪存块的数据页，当缓冲区可用空间低于设定的阈值，聚集最多数据页的块将被换出内存. FAB 适用于装有闪存芯片的 PDA、媒体播放器、移动电话和数码相机等设备，这类设备的数据管理具有典型的顺序访问比较密集特性，FAB 难以适用随机访问频繁的负载. 针对 FAB 存在的问题，BPLRU[17] 和 REF[18] 在选择牺牲块时通过数据页填充和标记脏页置换历史等方法减少随机写操作的代价.

(2) CFDC

CFDC[19] 将数据页以簇的形式组织到一起，每一个簇由逻辑地址相近的 n 个数据页构成并赋予置换权值，权值低的簇放置在优先置换区. 替换算法选择优先置换区中权值最小的簇替换出内存. 置换权值公式为

$$P(c)=\frac{\sum_{i=1}^{n-1}|p_i-p_{i-1}|}{n^2\times Age(c)} \quad (4)$$

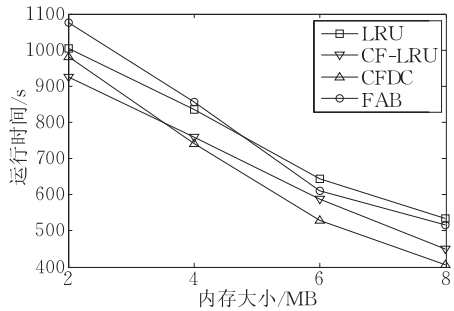
其中，n 表示一个簇所包含的数据页的数量；p_i-p_{i-1} 的绝对值表示簇内相邻两个页的逻辑页号差，通常逻辑上越接近的数据页物理上也相近，$Age(c)$ 反映簇的新旧程度. CFDC 充分考虑簇的访问最近性和访问频度，聚集数据页时表现出更强的灵活性，避免了对闪存进行大范围的随机写操作. 表 3 列出了目前闪存数据库主要的缓冲区管理算法.

表 3 闪存数据库缓冲区算法性能对比

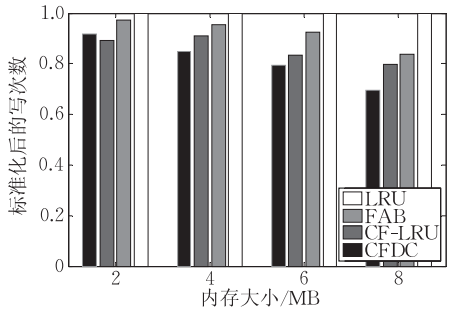
算法	粒度	内存管理	主要特性
CF-LRU[9]	页级	基于 LRU 算法,管理简单	优点:有效减少对闪存的写操作 缺点:存在脏页污染问题、命中率有时较低
FOR[14]	页级	动态调节冷热数据的长度比例,复杂度高	优点:有效区分冷热数据页、命中率高、写代价低 缺点:通过保存置换历史来区分冷热数据,结构复杂,维护代价高
FAB[16]	块级	以块为单位组织脏数据,管理相对简单	优点:以聚簇的形式将脏页写回闪存,闪存写代价低 缺点:对随机访问较频繁的负载性能差,存在脏页污染
CFDC[19]	块级	选择逻辑地址相邻的脏页聚簇,复杂度高	优点:根据权值设定优先置换区,减少脏页污染 缺点:对随机访问较频繁的负载,聚簇的有效性降低
AD-LRU[13]	页级	分为冷热两个队列,管理相对复杂	优点:根据访问负载动态调整冷热数据队列,访问命中率高 缺点:算法设计复杂,冷热数据之间迁移代价高

3.2.3 实验对比

为了对比已有闪存缓冲区管理策略的性能,本文基于 Flash-DBsim[20] 模拟器实现了 LRU、CF-LRU[9]、FAB[16]、CFDC[19] 等替换算法.实验环境配置:数据页大小为 2 KB,每个闪存块包含 64 个页.闪存芯片的读、写和擦除延迟分别为 20 μs,200 μs 和 1.5 ms.实验所用数据集基于 TPC-C 标准,采用 BenchmarkSQL 运行 PostgreSQL 数据库获得.数据请求次数为 2 087 142,读和写操作所占比例分别为 71.3%和 28.7%.CF-LRU 算法的优先置换窗口大小设置为可用缓冲区的 60%,CFDC 算法优先置换区的大小设为 40%,每一个簇的大小与闪存物理块大小相同.实验主要分析数据集的运行时间和闪存物理写次数,结果如图 5 所示.在运行时间方面,从图 5(a)可以看出基于闪存的缓冲区替换策略的总体性能胜过 LRU 算法,其中 FAB 算法采用聚簇的方式替换数据页,对于顺序访问特征不明显的负载,FAB 可能会导致一些频繁访问的数据过早地被替换出内存,在小容量内存环境下其性能不如 LRU 算法.在闪存物理写操作方面,从图 5(b)可以看出,基于闪存的缓冲区替换策略因为考虑了读写不对称特性,替换策略为脏页赋予更高的置换权限,数据集运行过程中产生的闪存物理写次数明显降低,CFDC 算法采用聚簇的方式将脏页批量写回闪存,置换过程避免了大范围的随机写操作,在所有对比算法中引发的物理写次数最少.



(a) 运行时间



(b) 物理写操作的次数

图 5 不同闪存缓冲区替换策略性能对比

3.3 索引管理

索引记录了数据和其存储地址的映射关系,利用索引可以快速定位相关数据,降低 I/O 操作代价、提高查询效率.索引必须能够反映数据存储位置发生的变化,当数据发生更新时索引需要做及时更新.对于由 n 条记录组成的数据集 T,为 T 创建的索引为 T_{index},通过索引对 T 进行查询节省的I/O代价 $gain(T)$可简单的表示为

$$\begin{aligned} gain(T) &= scan(T) - scan(T_{\text{index}}) - update(T_{\text{index}}) \\ &= \alpha \times C_{\text{read}} - \beta \times C_{\text{write}} \end{aligned} \tag{5}$$

式中,$\alpha \times C_{\text{read}}$ 表示扫描 T 和 T_{index} 的代价,α 是查询 T 所需总 I/O 数,C_{read} 是一次读 I/O 代价;索引维护代价 $update(T_{\text{index}})$ 大小为 $\beta \times C_{\text{write}}$,其中 β 表示更新索引产生的 I/O 总数,C_{write} 是一次写 I/O 代价.与磁盘相比,闪存的 C_{write} 远高于 C_{read},因此闪存索引更新需要付出相对更大的代价.以 B^+-Tree 索引为例,B^+-Tree 结点更新通常只需修改小部分数据,由于闪存最小以页为单位进行读写,少量数据更新也需要重写整个页面,索引更新引发的大量擦除操作将极大降低索引性能和闪存寿命,减少索引更新代价是闪存数据库索引管理需要解决的主要问题.批量延时更新和索引结构优化是目前解决这一问题使用较为广泛的技术.

3.3.1　延时更新

对于数据更新引发的索引变化，延时更新不是将更新传播至已有索引，而是把更新操作缓存在内存，当缓存的数据满足设定条件再执行批量更新操作. 延迟更新通过消除冗余操作和批量提交的方法减少了写传播的代价.

(1) LA-Tree

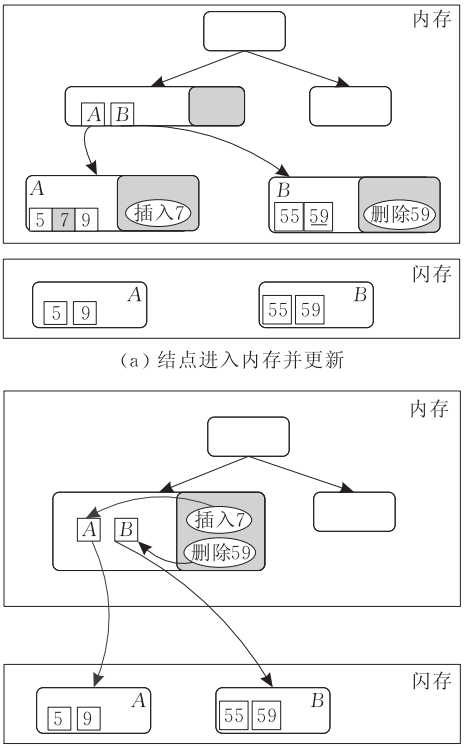
Agrawal 等人[21]提出了一种基于树的索引结构——LA-Tree. LA-Tree 以树根为起点，逻辑上将树划分为一系列高度相同的子树，每一棵子树在内存都有专用的缓冲区用于记录对该子树所有结点的更新. 执行查询操作时，LA-Tree 检查结点所在子树的专用缓冲区是否满足设定的更新条件，若满足则清空缓冲区并对该子树以及与它关联的后代子树进行批量更新. LA-Tree 优化了对闪存的写操作，缺点是对每一子树都需要维护特定的内存空间，这增加了缓冲区管理的复杂性.

(2) Lazy-Update B^+-Tree

文献[22]提出的 Lazy-Update B^+-Tree 将缓冲区划分为两部分，一部分用于缓存 B^+-Tree 结点，另一部分缓存对 B^+-Tree 相应结点的更新请求(更新缓冲池). 更新缓冲池用十字链表结构组织数据，结点结构为$\{key, recptr, type\}$，其中 key 存储元组的主键，$recptr$ 指针指向被更新的元组的地址，$recptr$为 null 代表该元组被删除，$type$ 表示更新操作的类型(删除、插入和修改). Lazy-Update B^+-Tree把对同一结点的更新请求聚集成一个序列，聚集可以消除对 B^+-Tree 结点冗余的更新操作，进而减少索引更新的写代价. 更新缓冲池被填满后，代价最小的序列被更新，更新代价由 $gain(g)=cost(R)+cost(R')-cost(R\cup R')$ 来衡量，R 表示序列 g 当前更新请求，R'表示延时 g 到 t 时刻时的更新请求，$cost(R)$和 $cost(R')$代表在当前时刻和 t 时刻两次单独提交 g 的写代价. $cost(R\cup R')$则是聚集到 t 时刻批量提交的代价. 简单地说，代价最小指的就是某一索引结点更新引发 B^+-Tree 结点分裂或合并的数量最少.

(3) UM-B^+-tree

文献[23]提出了一种基于 B^+-tree 的索引结构——UM-B^+-tree. UM-B^+-tree 把索引结点分成数据区和更新区，数据区存储记录的键值和地址，更新区存储对该结点的更新记录. 如果索引结点需要被换出内存，对该结点的全部更新记录都将迁移到父结点暂存，当结点再次读入内存，暂存在父结点的更新记录将与原数据合并产生版本数据. 当更新记录达到设定的阈值，新数据将被写入闪存. UM-B^+-tree 结点更新过程见图 6.



(a) 结点进入内存并更新

(b) 结点换出内存，更新记录迁至父结点

图 6　UM-B^+-tree 结点更新示例

举例说明，图 6 中结点 A 和 B 读入内存并分别执行一次插入和删除操作，更新记录保存在各自的更新区，换出内存时 A 和 B 将更新记录上移至各自的父结点. UM-B^+-tree 通过暂存更新记录的方式延迟对闪存的更新，一定程度上减少了对闪存的写次数，但每个结点都需要预留保存更新记录的区域，空间利用率相对较低. 延迟更新是牺牲读代价来换取写代价的，这对数据查询、事务恢复及并发控制等会有一定程度的影响.

3.3.2　索引结构优化

优化索引结构是提高索引性能的另一种有效途径. 结构优化不仅可以降低索引更新代价，还可以提高数据查询效率.

(1) μ-Tree

闪存异位更新会引发 B^+-Tree 索引在闪存上代价昂贵的级联更新. 所谓级联更新指的是任何一索引结点的更新可能都会导致根结点到该结点路径上所有结点的更新. 级联更新会产生大量的随机写操作，严重影响索引性能. μ-Tree 索引[24]将根到目标结点路径上的所有结点存储在同一闪存块，数据更新产生的新路径将被存储在新的闪存块，旧块中

原路径结点无效. μ-Tree 结点大小由闪存页面大小和结点所在层次决定. μ-Tree 结点更新过程见图 7. 举例说明,结点 F 发生更新操作,新路径 A 到 F 的所有结点存储在新数据块,原来的路径结点(A,C,F)无效.

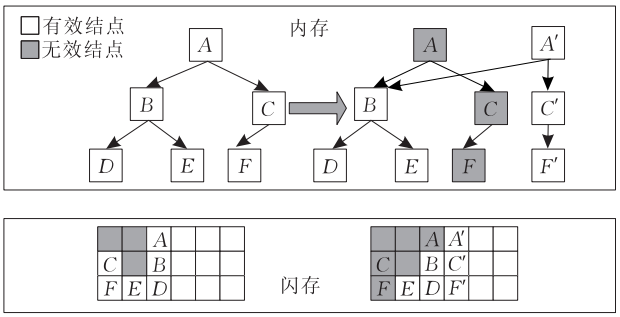


图 7 μ-Tree 索引更新示例

μ-Tree 每次数据更新都发生在同一闪存块,这有利于降低随机写操作代价,但这种方式也会导致闪存内出现许多无效结点,空间利用率低,需要不断地进行垃圾回收,μ-Tree 结构无法支持范围查询.

(2) MicroHash

MicroHash[25]是一种基于 Hash 的闪存索引结构. MicroHash 在内存维护一个能有效存取闪存数据的数据结构,闪存中的记录以循环数组方式被组织成堆结构,内容包括目录和索引页. 每个目录包含一系列的桶,每个桶存放映射到该桶的最新索引页地址,每个索引页存放数据记录的地址,这些页存放在闪存上,仅当被请求时才调入内存. MicroHash 适合小容量的闪存设备,在较小内存开销的情况下实现了对闪存友好的写操作. 文献[26]也提出了一种基于闪存的新索引结构——HashTree. HashTree 混合了散列和树两种索引结构. 它选择合适的 Hash 函数将数据均匀的散列到不同的桶中,其中 Hash 表驻留内存,每一个 Hash 桶存储在闪存,每个桶中的数据采用类似于 FD-Tree[27] 树形结构进行组织. HashTree 将树以 Hash 桶为单位划分成有序的子树,有效减少了树结点级联更新的代价.

(3) PBFilter

PBFilter[28]是基于医疗领域数据特点设计的一种高效索引结构. 医疗 USB 卡记录一般都是插入和查询操作,很少存在删除和更新. PBFilter 按照时间顺序将记录写入闪存,并为这些记录建立(Key,$Value$)索引项. 因为索引项是无序存放的,为了提高查询效率,PBFilter 对 key 值建立 Bloom Filter 索引. 查找记录时,通过查找 Bloom Filter 获取所需记录. Bloom Filter 所需空间比索引项要小得多,相比于其它索引结构,PBFilter 具有更加优秀的随机查询能力. 表 4 列出了几种典型的闪存数据库索引.

表 4 闪存数据库索引性能对比分析

方法	类型	应用场景	主要特性
LA-Tree[21]	Tree	通用数据库系统	优点:优化闪存的写操作,索引更新代价低 缺点:查询性能低、传播代价高
Lazy-Update B$^+$-Tree[22]	B$^+$-Tree	通用数据库系统	优点:冗余更新少,优化闪存的写操作 缺点:结构复杂,维护代价高,不利于数据恢复
μ-Tree[24]	B$^+$-Tree	通用数据库系统	优点:级联更新代价低 缺点:存储太多冗余数据,空间利用率低,无法进行范围查询
MicroHash[25]	Hash+Queue	传感器数据处理	优点:结构相对简单、维护代价少 缺点:只能适用于小容量闪存(传感器)
PBFilter[28]	Bloom Filter	嵌入式数据库	优点:优化更新,无随机写操作、查询性能好 缺点:不支持范围查询、只能应用于特定领域

3.4 查询处理

查询处理是数据库系统最重要的任务之一. 为了提高系统性能,对查询处理进行优化非常必要. 在基于磁盘的数据库系统中,已有的查询优化技术主要考虑磁盘 I/O 代价对查询处理的影响. 比如在处理多表连接时,许多连接算法采用块级 I/O 访问来减少磁盘寻道时间,降低磁盘 I/O 数量. 避免或降低对磁盘的随机访问是众多经典查询优化算法考虑的一个通用原则. 闪存与磁盘的 I/O 特性不同,传统的查询优化技术是否能够发挥闪存的存储特性便成为闪存数据库系统需要考虑的一个重要问题. 文献[29]用固态硬盘替换磁盘对循环嵌套、排序-合并、Grace Hash 和混合 Hash 等连接算法进行了性能测试. 实验结果表明这 4 种连接操作的执行时间均低于在磁盘上执行的时间,但性能提升的幅度却不足 2 倍. 分析算法特征和实验结果,我们认为现有连接算法在查询处理过程中引发的 I/O 操作不适合闪存的存储特性,闪存优良的随机读性能没有得到充分的发挥. 具体表现在两方面:(1) 传统连接算法需要保存大量中间结果而产生较多的写操作,频

繁的写操作会使查询性能急剧恶化；(2) 传统算法着重考虑磁盘 I/O 代价对查询性能的影响，很少考虑 CPU 代价因素，对于固态硬盘来说，读写数据需要通过闪存转换层完成地址转换，所以耗费 CPU 资源比磁盘相对要高，CPU 代价在某种程度上制约了连接性能的进一步提升. 文献[29]还测试了闪存 I/O 大小对连接性能的影响，实验表明块级 I/O 访问有利于提高查询性能，大块的数据请求可以充分利用固态硬盘实时的空闲通道，通过发挥固态硬盘内部并行读写机制来提高性能. 文献[30]基于 PostgreSQL 数据库系统对查询性能进行了对比评测. 与磁盘相比，固态硬盘存储环境下单点查询性能最高有近 50 倍的提升，但不同扫描操作符性能提升的幅度差距比较大. 在查询选择率较低的情况下，索引扫描因为充分发挥了固态硬盘良好的随机读性能，性能表现最为突出. 多表连接操作在固态硬盘上的性能差异也比较大，以循环嵌套连接为例，如果对大表进行索引扫描，查询性能提升的幅度较明显，这是因为大表元组多，内存命中率低，需要更多地随机读操作. Hash 连接算法因为需要更多的随机访问，所以在固态硬盘上性能提升较大，但与其它连接算法相比，需要的查询执行时间最多，这主要是因为对元组进行 Hash 分桶会触发许多随机写操作，而随机写操作是闪存最不擅长的一种访问模式. 从已有的测试结果不难看出，适用于磁盘的经典连接算法在固态硬盘上的性能表现并不突出. 因此，现有查询优化算法必须针对闪存独有的 I/O 特性重新设计. 目前研究者主要从代价模型修改和优化查询连接算法两个角度展开研究.

3.4.1　查询代价模型

数据库系统结构设计和算法优化与存储介质的 I/O 特性紧密相关，在查询规划过程中，不同的查询生成树通过代价模型对相应的查询路径进行估算，代价最小的执行方案将被转换成最终查询计划并由执行器执行. 查询路径的代价估计主要考虑 CPU 代价和磁盘 I/O 代价. 比如，PostgreSQL 数据库系统的代价模型将顺序存取一个磁盘页的代价作为基本单位(取值为 1)，随机存取操作的代价取值为 4. 查询优化算法在代价评估和生成查询计划树等环节只区分磁盘顺序和随机操作代价，多采用排序、延迟物化等手段减少对磁盘的随机 I/O 访问. 闪存的随机读操作与顺序读操作性能相差不大，将现有的代价估计模型运用于闪存数据库系统可能会影响最优查询计划的生成. 因此，针对闪存特有的 I/O 特性对现有代价模型进行优化可以提高查询代价估计的准确性，进而提升数据库的查询性能. 文献[31]基于 PostgreSQL 提出了一个适用于闪存的代价评估模型. 新的代价模型区分读写操作，I/O 代价参数设置为顺序读操作、顺序写操作、随机读操作和随机写操作等 4 种. 扫描操作代价估计只区分顺序读和随机读，物化操作则只对顺序写和随机写进行分别处理. 通过分析排序连接和 Hash 连接等操作产生的 I/O 访问序列，新模型将原有模型的读写代价比进行了调整. 针对连接过程中读写操作混合的情况，新模型通过一个启发式迭代校正算法适时地对代价参数进行调整. 该文献将新的代价模型应用于 PostgreSQL 并进行了 TPC-H 测试，TPC-H 22 个查询的执行时间大部分都有所减少，平均性能提高 48%，部分查询性能有 5 倍的提高，个别查询的执行时间有略微增加. 结合闪存的读写特性，设计适合闪存的代价评估模型是提高查询性能的一个有效途径.

3.4.2　优化查询连接算法

(1) RARE-join

连接查询通常只需要输出连接表的部分属性列，Ailamaki 等人[32]为了避免读取与查询不相关的属性列，提出了一种新的存储模型 PAX. 每一个 PAX 数据页存储一定数量的元组，数据页按元组属性列紧凑堆放，这种存储模型兼有按行存储和按列存储的优点. 文献[33-34]基于 PAX 提出 RARE-join 连接算法，RARE-join 首先根据连接条件创建一个连接结果表，元组结构为$\{id_1, id_2, \cdots, id_i, \cdots\}$，其中 id_i 表示第 i 张表某一元组的存储地址；然后为存储在结果表中的元组地址构建索引，根据索引从原表中读取数据并生成查询结果. RARE-join 只选择与查询结果相关的属性列，这减少了数据的读取量，避免了大量中间结果的生成，物化过程放在查询的最后阶段，闪存的写操作比较少. RARE-join 存在的主要问题是维护底层 PAX 存储和索引的代价较大.

(2) DigestJoin

为了减少排序-合并等连接算法产生的中间结果，Li 等人[35]提出针对闪存特性的 DigestJoin 查询优化算法. DigestJoin 算法将查询过程分为两个阶段：第 1 阶段抽取待连接表元组的键值和连接属性构建 Digest 表，由 Digest 表根据连接属性通过排序或者 Hash 的方式连接产生结果表，结果表存储与查询结果相关联的元组在原始表中的地址；第 2 阶段是实体化查询结果，根据结果表记录的元组地址从闪存读取原始表的相关属性列生成最终的查询结果. 实体化过程需要大量的随机读操作，这是影响查

询性能的关键因素.为了尽量避免反复读取某一数据页,DigestJoin 通过构建数据页之间的连接关系图对实体化的过程进行了优化,有效解决了反复读取同一数据页产生的内存抖动问题.

(3) Sub-Join

文献[36]提出的 Sub-Join 算法首先将执行连接操作的相关数据表在主键和连接列上进行投影,根据连接列进行排序形成连接子表;然后在子表上进行连接操作,连接过程需要的其它结果数据由原始表回取获得.Sub-Join 对子表采用基于列的存储形式,闪存随机读操作相对较少.

Myers[37]提出基于闪存的排序-合并连接算法.排序-合并算法通常先将连接表读入内存并分段排序,然后将排序后的元组写回磁盘形成有序片段.Myers 针对闪存写性能相对较差和寿命有限的特点对中间结果的存取过程进行了优化,只将连接列写入闪存,在排序结束后,再从原始表获取其它数据,完成最初的排序过程.

闪存不对称的读写代价是影响闪存数据库查询性能的关键因素.设计基于闪存的查询优化策略时必须尽量减少写操作尤其是随机写操作,同时应充分利用闪存很好的随机读性能,牺牲读代价以降低写代价是提高查询性能的有效手段.表 5 列出了几种典型的闪存数据库连接算法.

表 5 闪存数据库连接算法性能对比分析

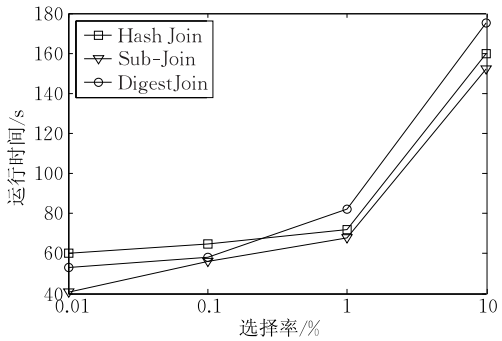
连接算法	连接类型	性能优势	存在的问题
RARE-Join[33]	索引连接	对连接列构建索引,充分发挥 PAX 列存储和闪存随机读性能	记录频繁更新时,PAX 页和索引维护代价高
DigestJoin[35]	排序-合并	避免生成大量的中间结果,连接过程中闪存写操作比较少	数据回取可能造成数据页被频繁地换进换出内存,内存开销大
Sub-Join[36]	循环嵌套	按列存储连接子表,数据的读取快	执行多表查询时数据回取的代价高,中间结果产生比较多的写操作
Subset[37]	排序-合并	大幅降低了对闪存的写操作,充分利用闪存的高速读性能	排序片段连接代价高,难以执行多表连接

3.4.3 实验对比

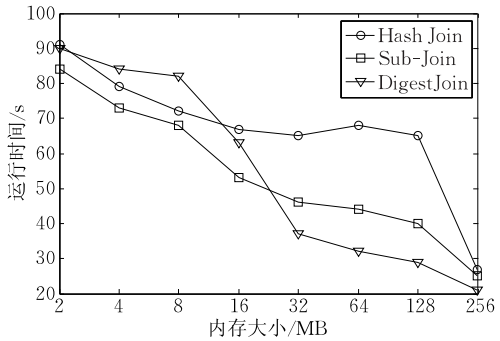
为了对比固态硬盘环境下查询处理的性能,我们分别对传统 Hash 连接、DigestJoin[35]和 Sub-Join[36] 3 种连接算法进行了实验比较.实验平台配置:Intel Core i5-2400@3.10 GHz 处理器,内存 8 GB,操作系统是 Ubuntu12.10,数据集采用 TPC-H 的 CUSTOMER 表和 ORDERS 表并做等值连接,其中 CUSTOMER 由 150 万条记录组成,大小为 256 MB,ORDERS 表由 1.5 亿条记录组成,大小为 2 GB.实验测试了选择率和缓冲区大小对连接算法的影响.

(1)选择率.选择率反映了结果数据的规模,选择率越大生成的结果数据越多,许多查询连接算法都通过变化选择率来评估查询性能.对比实验中选择率的取值为 0.01%~10%.可用内存是 8 MB,图 8(a)描述了不同选择率下的查询执行时间.从实验结果可以看出,当选择率低于 0.1%时,DigestJoin 和 Sub-Join 的性能优于 Hash Join.随着选择率的增大,DigestJoin 和 Sub-Join 因为需要保存更多的中间结果,查询执行时间增长幅度比较大.Sub-Join 因为对子表采用基于列的存储,在一定程度上减少了对 SSD 的写操作,其查询性能略优于 DigestJoin;

(2)缓冲区.缓冲区大小是影响查询连接性能的重要因素,图 8(b)描述了不同缓冲区大小下 3 种连接算法的性能.DigestJoin 受缓冲区大小的影响比较明显,在缓冲区大小超过 16 MB 后,查询连接性能提升幅度较大.Sub-Join 基于循环嵌套连接算法,在生成连接子表时对内存大小要求较低,在缓存



(a) 不同选择率查询执行时间对比



(b) 不同缓冲区大小查询执行时间对比

图 8 不同查询连接算法性能对比

区较小的情况下，查询性能胜过 DigestJoin.

3.5 事务恢复

事务恢复机制用于保障系统发生异常时将数据库系统恢复到之前的稳定状态. 现有的恢复方法主要包括预写日志和影子页两类. 日志记录了事务对数据的更新并被顺序追加到日志文件尾部，恢复操作根据日志完成对某一事务的撤销或重做. 恢复过程通常会产生大量小的随机写操作，闪存处理小范围随机写的代价比较高，然而现有的日志处理技术难以充分利用闪存的存储特性，优化日志结构和恢复策略是提高闪存数据库事务处理性能的关键. 基于影子页的事务恢复机制采用异位更新方式，更新操作会将数据存储在新的物理页，系统维护两张页地址映射表来保存逻辑页号和物理页号的对应关系. 影子页技术不会产生日志来记录对数据库的操作，没有显式的回滚机制对数据库的操作进行撤销. 如果当前事务正确提交，则可以将当前页表写入外存. 如果事务未能提交，则丢弃当前页表并恢复到该事务执行前的状态. 影子页在事务提交或撤销时要执行修改映射信息、回收无效数据等操作，频繁的随机读操作使得影子页技术并不适合磁盘数据库系统. 闪存的物理特性与影子页数据管理十分吻合，在闪存数据库系统中使用影子页有助于提升事务处理能力.

3.5.1 基于日志的方法

HV-recovery[38] 基于闪存异位更新和高速的随机读特性设计实现了适用于闪存的数据库恢复方法. 闪存在执行更新时会自动保存同一数据页的多个不同版本，HV-recovery 根据这一特性提出了一种新型的基于地址的日志处理策略——HV-Logging. HV-Logging 机制的日志结构为 $\{Tid, element, PreAddress, PreValue\}$，其中 Tid 唯一标识某一事务，$element$ 是事务操作的对象元素，$PreAddress$ 指向旧版本历史数据，$PreValue$ 记录被更改元素的旧值. 事务回滚管理模块通过读取日志获得各元素旧版本地址，根据地址取出数据并判断是否与日志保存的旧值相同，若相同，将新的数据页置为无效，恢复原数据页为有效状态；若不同，则重新写入新数据. HV-Logging 优化了日志结构，减少了冗余日志和垃圾数据的存在，提高了空间利用率，利用闪存快速随机读的特性大幅减少恢复时间. LB-Logging[39] 采用链表结构来记录日志. 同一事务更新的各个元素和每个元素的不同版本都分别建立指针链，这保证了事务和被修改元素之间的关联性，恢复操作根据指针链可以准确、快速地找出该事务所有操作日志. FlashLogging[40] 从系统代价的角度提出了一种性价比较高的日志处理方案. FlashLogging 用价格低廉的 USB 闪存存储器专门存储记录日志. 日志通常以顺序追加的方式写入外存，这符合闪存顺序写带宽高的存储特性. FlashLogging 将许多 USB 闪存设备以阵列的形式来组织，解决了单块设备容量有限的问题.

3.5.2 基于影子页的方法

SLC 型闪存允许对同一页在擦除之前多次写入数据. FlagCommit[41] 针对 SLC 闪存这一特性提出一种基于影子页的事务处理机制. FlagCommit 在闪存页的备用区存储和事务相关的元数据，这些元数据包括事务 ID、事务状态、事务更新的前一个数据页地址、提交标记等. 链表指针把同一事务关联的数据页逻辑上连在一起，事务提交或回滚时通过这些元数据进行数据更新或恢复. FlagCommit 中被更新的数据页可被看作影子页，备用区中的元数据构成了众多分散的地址映射表. 事务提交协议根据存储在备用区的提交标记执行提交或回滚操作. 当数据更新或垃圾回收导致链表断裂时，修改提交标记可以保证事务和该事务相关的多个分裂链表逻辑上关联在一起.

影子页在 1977 年被提出. 事务处理因为不需要预写日志而变得更加有效，但影子页在处理过程中会产生很多随机读操作，因此在磁盘存储时代影子页技术未能得到广泛应用. 闪存异位更新的特性天然地保存了事务更新的历史版本数据，这为影子页恢复机制提供了很好的支持. 表 6 列出了几种主要的闪存数据库恢复方法.

表 6　闪存数据库恢复算法性能对比分析

算法	恢复技术	性能优势	存在问题
HV-recovery[38]	日志	数据恢复时避免了大量数据重写操作；有利于延长闪存的使用寿命；恢复操作简单	存储日志文件需要较多的空间；检查点机制较复杂；无效数据的回收代价大
FlagCommit[41]	影子页	避免写入大量日志；有利于实现更细粒度的并发操作；数据恢复更快	元数据管理复杂；垃圾回收机制复杂、开销大；仅适用于昂贵的 SLC 闪存
LB-Logging[39]	日志	优化了事务恢复的日志读取过程；充分发挥闪存快速随机读性能，恢复所需时间大大减少	日志中记录的旧版本数据链表管理复杂；维护日志的开销大
FlashLogging[40]	日志	USB 接口闪存设备单独存储日志，价格低廉、性价比高，日志处理性能高	每个 USB 设备需要单独工作线程服务，维护难度高；擦除等操作比较容易引发异常点，影响稳定性

4 基于闪存的混合存储数据管理

尽管基于闪存的固态硬盘(SSD)比磁盘具有更大的性能优势,但是短期内 SSD 不会完全取代磁盘成为主流存储介质.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.混合存储主要考虑到以下几个因素:

(1)SSD 使用寿命.闪存容量的不断增大是通过牺牲可用寿命的代价来获得的,频繁的数据访问会极大缩短 SSD 的寿命,SSD 完全取代磁盘对系统运行的可靠性和稳定性是一大考验;

(2)价格成本.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十倍,基于闪存的混合存储系统性价比高;

(3)数据访问的局部性.数据访问通常呈现不均衡特性,一段时间内只有部分数据是频繁被访问的,约有 40%的数据访问频度较低.根据 SSD 和磁盘的物理特性将二者组合,有选择地把数据分配到不同存储介质(比如将最热数据或读操作密集的数据交由 SSD 来处理),不仅能满足大数据、高吞吐等应用需求,也可以有效降低企业成本.

考虑 SSD 和磁盘混合存储的结构特性,混合存储数据管理目前研究点包括:(1)SSD 和磁盘同作二级存储;(2)SSD 作为内存的扩展缓存.

4.1 SSD 和磁盘同作二级存储

结合硬件特性和数据访问特征,SSD 和磁盘在混合存储数据管理中分别承担不同的存储任务:(1)磁盘缓存上层应用对 SSD 的写操作;(2)SSD 存储特定类型数据.

4.1.1 磁盘用作 SSD 的写缓存

为了降低随机写操作对 SSD 寿命和系统性能的影响,Griffin[42]选择用磁盘缓存上层应用对 SSD 的写操作,Griffin 以日志的方式将内存换出的脏页顺序写入磁盘,后台进程有条件地将存储在 SSD 上的原始数据和磁盘上的日志数据进行合并生成新数据.文献[43]基于页面分类思想将系统对 SSD 的随机写操作暂存于磁盘,通过磁盘将这些随机写序列转换为顺序操作序列并最终写回 SSD.

将磁盘用作 SSD 写缓存的存储策略发挥了磁盘或磁盘阵列顺序访问性能较好的特点,利用 SSD 高速随机读性能对数据进行合并后顺序写入 SSD.需要注意的是,磁盘 I/O 瓶颈和频繁的数据合并使得这种混合系统难以应对更新操作密集和访问模式多变的负载.

4.1.2 SSD 存储特定类型数据

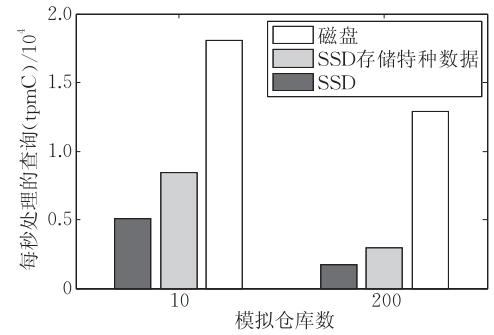
基于 SSD 优良的随机读性能,通过分析负载访问特征和数据类型,将读操作密集的数据、索引文件、临时表等存储在 SSD 有利于提高系统的数据处理能力.

(1)特定访问模式的负载.文献[44]提出了基于页面 I/O 统计信息的页面迁移模型.该模型根据 I/O 统计信息计算数据存放在 SSD 和磁盘的代价,根据代价对页面进行分类并动态地对 SSD 和磁盘数据进行迁移.迁移的目的是将读操作密集的负载放在 SSD 上存储,而写操作密集的数据则保留在磁盘上.I-CASH[45]提出将 SSD 和磁盘进行成对组合,采用智能算法将较长时间不被修改的数据和读操作密集的数据存放在 SSD,以日志的方式记录对 SSD 数据的修改并增量存储于磁盘,I-CASH 尽量避免对 SSD 产生随机写操作,读数据时通过相似度探测获得数据的最新版本;

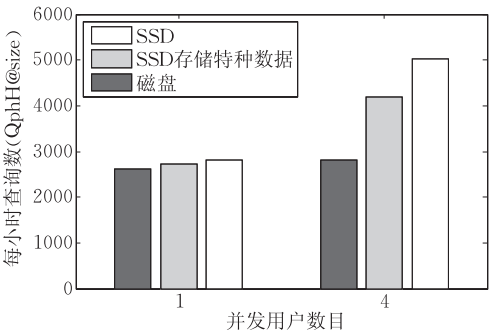
(2)特定数据.数据库系统中通常包含许多访问特征明显的特定类型数据.比如,为了保证数据库的一致性,日志需要在事务提交之前先行写入外存,预写日志引发频繁的磁盘访问,将严重影响数据库系统整体性能的提升.用 SSD 存储这些特定类型数据可以减轻磁盘的 I/O 压力.Hystor 系统[46]通过识别算法发现影响系统性能的关键数据块并将它们备份到 SSD,待 SSD 完成对这些数据块的修改再将新数据重新写回磁盘.用 SSD 存储查询连接操作过程中生成的临时表、中间结果等数据也可以有效提升循环嵌套、Hash、排序-合并等连接算法的性能[6,47];

(3)性能对比.为了对比混合存储的系统性能,我们搭建了基于三星 MZ—7PD 型 SSD 和磁盘的混合存储平台,并在磁盘、混合存储和 SSD 3 种不同的存储架构下对 PostgreSQL 数据库系统的性能进行了测试.SSD 在混合存储环境中的作用是存储日志文件和系统表数据.TPC-C 测试采用 BenchmarkSQL 分别对 10 个和 200 个数据仓库进行事务处理性能测试,实验结果如图 9(a)所示,在混合存储环境下,PostgreSQL 数据库每秒处理的事务数(tpmC)比磁盘存储提高近 1 倍.数据库系统的日志操作多以顺序读写为主,而对系统表的访问更多的是读操作,因此,将日志和系统表分离存储在 SSD 可以发挥 SSD 良好的顺序读写和快速随机读性能.3 种存储架构下的 TPC-H 测试结果如图 9(b)所

示，查询数据流由 22 个复杂查询和 2 个更新操作随机组成，数据库包含 8 张表，数据量为 1 GB. 实验结果显示，将特种数据存储到 SSD，多并发用户下查询处理吞吐提升的幅度为 48.9%.



(a) TPC-C性能测试



(b) TPC-H 性能测试

图 9　用 SSD 存储特种数据性能对比

4.2　SSD 作为内存的扩展缓存

随着内存读写速度的不断提升，内存和磁盘之间的性能鸿沟在持续变大. 尽管大型数据中心配置的内存容量越来越高，但数据的增长远超过内存容量的增长，简单的扩充内存容量不仅需要昂贵的财力支出，还要消耗大量能源. SSD 的出现弥合了内存和磁盘之间的性能差距，在内存和磁盘之间添加 SSD 作扩展缓存层可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. SSD 作内存扩展的系统架构见图 10.

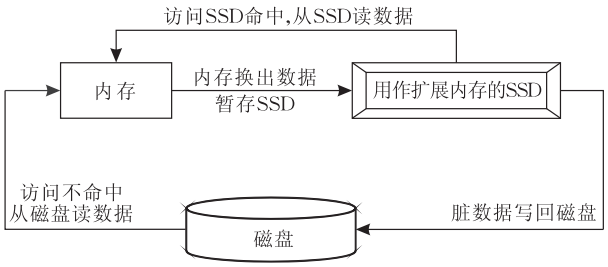


图 10　闪存做扩展缓存框架图

图中 SSD 的作用是暂存从内存置换出的数据，当内存缺页中断时，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 作扩展缓存层最需要解决以下两个核心问题：

(1) 内存中哪些数据在何时缓存到 SSD. 不同于单一类型介质的缓冲区管理，在异质多级缓存环境下，缓冲区替换算法必须考虑数据页的读写状态、数据访问形式和硬件自身特征等因素；

(2) SSD 中的数据怎样组织并如何换出或写回磁盘. SSD 暂存了从内存被驱逐出的数据，合理地组织存储这些数据有助于内存缺页时快速识别并读取 SSD 中的数据. 当 SSD 可用空间低于设定条件时，需要将部分数据页换出 SSD，其中的脏页需要写回磁盘. 设计适合 SSD 读写特性的替换策略对系统性能的影响至关重要.

4.2.1　缓存策略

文献[48]基于 IBM 的 DB2 数据库系统提出了 TAC 缓存策略. TAC 考虑数据访问的冷热特性，以块(32 个数据页)为热度计算单位，每次从磁盘读数据都会计算该数据所在的块是否满足一定的热度条件，若满足则被缓存在 SSD，如果 SSD 可用空间不足则选择热度最低的数据换出 SSD. 对于内存换出的脏页，TAC 同时写入 SSD 和磁盘. 文献[49]提出的 LC 缓存策略区别对待从内存换出的干净页和脏页，提出了 Clean-Write (CW)、Dual-Write (DW)、Lazy-Cleaning (LC) 3 种缓存策略. 这里 CW 表示 SSD 只缓存从内存置换出的干净页；DW 表示缓存内存换出的脏页和干净页，脏页被同时写入 SSD 和磁盘；LC 与 DW 的区别在于处理脏页的策略不同，LC 先将脏页写入 SSD，随后再批量写入磁盘. 3 种缓存策略对不同类型访问负载表现出的性能有较大差异：(1) 对于更新操作密集的负载，LC 算法的性能相对较好. 这是因为 LC 能够将对同一数据页的多次修改经过 SSD 的暂存而变成对磁盘的一次写操作，磁盘写操作的数量较少；(2) 对于读操作密集的负载，CW 表现出更高的命中率，DW 省去了 LC 处理脏页的复杂管理代价，所以在性能上要优于 LC. FaCE 系统[50]综合考虑了 SSD 价格、随机写操作性能差等因素，提出了一种经济有效的缓存管理策略. FaCE 选择 SSD 作为扩展缓存，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，降低系统恢复时间. FaCE 系统的主要特点包括：(1) 对于从内存被驱逐进入 SSD 的数据，FaCE 以顺序追加的方式写入 SSD，对于 SSD 已经存在的旧版本数据 FaCE 予以保留，这避免了对 SSD 的随机写操作；(2) FaCE 扩充了事务检查点机制和恢复模块，以组提交的方式周期性地将记录缓存数据的元信息写入 SSD，通过快速扫描存储在 SSD 中的元

数据,使系统恢复时间显著降低.

Oracle Exadata① 采用统计的方法将数据表、索引等热数据缓存至 SSD,内存换出的干净页也缓存在 SSD 以提高命中率,数据库系统性能提升较大.

hStorage-DB[51] 实现了一个基于混合系统的数据存储管理应用框架. hStorage-DB 底层采用 SSD 和磁盘的混合存储,SSD 用作内存的扩展缓存. hStorage-DB 为每一次 I/O 请求都赋予特定的语义信息,存储管理器响应 I/O 请求时根据语义信息执行缓存策略. hStorage-DB 为操作系统、文件系统以及数据库系统添加了一个访问优先级生成模块,该模块提取 I/O 请求携带的语义信息生成并特定的服务质量——QoS(Quality of Service),这里 QoS 实际上指的是页面进入或换出 SSD 的优先级. 存储管理器根据 QoS 对 SSD 和磁盘中的数据进行写入、读取、迁移等操作. hStorage-DB 弥补了存储系统和数据库系统之间 I/O 语义信息的缺失,存储处理不受负载访问动态变化的影响. hStorage-DB 对数据库系统特别是复杂的 OLAP 应用和高并发负载访问性能提升幅度比较大,对不同类型负载的查询处理性能均有提升. 表 7 列出了目前主要的几个缓存系统.

表 7 SSD 作为扩展缓存性能对比分析

缓存架构	页类型	数据替换	数据同步	主要特性
TAC[48]	Both	热度	Write-through	优点:缓冲区访问性能提升明显,数据同步代价低 缺点:基于 DB2 数据库,通用性不强;热度模型设计复杂,元数据管理代价高
LC[49]	Both	LRU-2	Write-back	优点:适用于多种访问负载;有效减少磁盘写操作的次数 缺点:SSD 数据替换引发较多随机写操作;数据库检查点设置复杂,恢复代价大
FaCE[50]	Both	FIFO	Write-back	优点:优化了写操作,系统吞吐高;恢复性能提升明显 缺点:多版本数据存储影响缓冲区命中率;数据写磁盘的代价高,空间利用率低
Exadata①	Clean	LRU	Write-through	优点:算法简单;SSD 和磁盘数据迁移代价少 缺点:冷热数据识别的准确性低; I/O 代价大

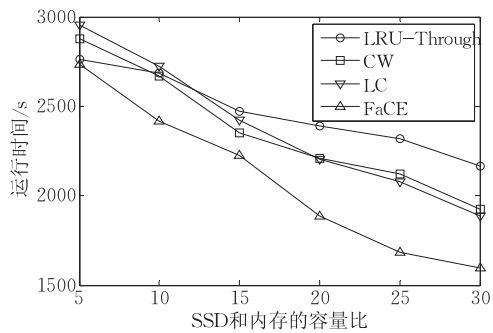
4.2.2 性能对比

为了验证多级缓存系统的性能,本文对已有的部分缓存策略进行了实验对比,我们基于 disksim4.0② 和 SSDModel[52] 设计实现了一个多级缓存实验仿真平台. 实验代码使用 C 实现,操作系统为 Ubuntu12.10,页面大小 4 KB,缓冲区大小 4 MB. 验证缓存策略包括 LRU-Through、CW[49]、LC[49] 和 FaCE[50]. 其中 LRU-Through 部分实现了 Exadata 的管理策略,内存置换出的脏数据将被同步写回闪存和磁盘,CW 和 LC 是文献[49]提到的两种策略. 实验所用的两个数据集基于 TPC-C 标准,采用 BenchmarkSQL 运行 PostgreSQL 获得. 数据集的统计信息如表 8 所示.

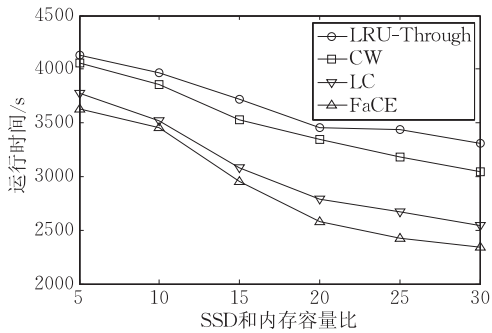
表 8 实验所用测试数据的统计信息

数据集	访问次数	读次数	写次数	读写比
T1	2193476	1805230	388246	82.3%/17.7%
T2	1988563	1262737	725826	63.5%/36.5%

在不同的 SSD 和内存的容量比下,实验统计了 T1 和 T2 的运行时间,图 11(a)描述了数据集 T1 在 4 种缓存架构下的运行时间. 从结果来看,FaCE 缓存策略的性能相对较好,因为 FaCE 系统采用顺序追加的方式处理从内存置换出的数据,SSD 数据管理的代价相对较低. 其它 3 种缓存策略的差异并不是很大,特别是 CW 和 LC 替换策略,原因在于数据集 T1 是读操作比较密集的负载,基于干净页的缓存策略降低了算法执行的复杂度,系统资源消耗少. LRU-Through 以 LRU 方式管理 SSD 中数据,



(a) T1



(b) T2

图 11 T1 和 T2 在不同缓存策略下的运行所用时间

① http://www.oracle.com/us/products/database/exadata
② http://www.pdl.ainu.edu/DiskSim

缓存性能不如 CW 的 LRU-K 算法. 数据集 T2 比 T1 有更多的写操作，正如图 11(b)所示，LC 和 FaCE 策略在处理脏页方面的性能优势体现得更加明显. 特别是 FaCE 系统，随着 SSD 容量的增大，系统运行时间显著减少，当 SSD 和内存的容量比超过 20 之后，性能提升幅度开始变小，这是因为 SSD 对无效数据页回收的代价影响了系统性能的提升.

5　未来工作的展望

结合新硬件发展趋势和现有的闪存数据管理特点，我们认为以下几个方面是未来的研究方向.

5.1　基于闪存的超大数据管理

随着越来越多应用技术的出现和发展，数据的增长已经完全失去控制，对业务运行也造成了影响，构建面向企业级应用的超大数据管理系统是解决海量数据存储和管理等问题的必由之路. 主要研究内容包括如下几个方面：

(1) 高性能、高扩展. 基于闪存的新硬件特性和大数据呈现的多样性，合理的数据分布会加快数据的访问性能. 数据管理体系的各个环节必须相应的调整管理策略以适应新型存储的特性. 关系数据库技术在可扩展性方面存在不足. 近年来，在超大数据管理系统方面，基于闪存的 NoSQL 数据库技术成为热门的研究点[53-54]；

(2) 低能耗、高吞吐. 低能耗已逐渐成为计算机存储系统的一项关键挑战. 将闪存引入大规模据管理系统非常有助于解决能耗和高吞吐等方面的严峻挑战. 一方面，引入闪存使得缓存空间显著增大，能获得更高的缓存命中率从而显著降低对磁盘的访问密度，在存储层内部可以通过倾斜的数据分布，使部分磁盘存储冷门数据来增加空闲时间达到节能的目的，另一方面，引入闪存可以显著减少内存的使用，从而降低缓存层自身的能耗；

(3) 可靠性、可用性. 闪存有限的使用寿命是构建大数据管理必须要考虑的问题. 设计闪存友好的算法和应用有利于保证系统的可靠性和可用性.

5.2　混合存储数据库系统

基于闪存的混合存储系统为构建高性能、可扩展的海量数据管理提供了一种经济有效的途径. 文献[55]中，作者基于闪存构建的多级缓存系统极大提升了数据库系统的恢复性能. 混合存储数据管理必须能够动态的调整外存系统中保存的数据，根据数据和闪存特性合理地分配数据存储位置来获得性能和价格的最佳比. 数据分类、数据分布、数据迁移等是混合数据库系统需要解决的关键技术.

(1) 数据分类. 在混合存储数据管理系统中，闪存设备常作为内存扩展以满足大规模数据应用对系统高吞吐的需求. 实现内存扩展这一功能的核心问题是如何对缓存数据进行有效分类管理，过滤冷数据，把对系统性能影响较大的热数据存储到闪存以提高命中率，数据分类算法必须能够有效地鉴别当前访问负载的冷热程度；

(2) 数据分布. 不同厂商和接口的 SSD 产品在读写速度、价格等方面的差距非常大，因此数据管理中心通常会选择不同类型和性能的 SSD 产品来构建多级异质的混合存储系统. 充分发挥各级存储介质的性能特性，合理的数据布局是实现高性价比数据库系统的重要因素. 如何智能有效地监测访问负载变化是数据合理分布的一大挑战；

(3) 数据迁移. 在混合存储系统中，为了完成数据的有效分布，数据迁移是不可避免的. 数据迁移是指数据在不同介质之间移动. 系统通常会根据实时的负载访问将读操作密集的数据迁移到 SSD，以提高数据的查询性能，修改比较频繁的数据则被存储到磁盘. 迁移策略必须充分考虑迁移过程中占用 CPU、总线、内存等系统资源而产生的系统代价.

5.3　新型存储技术对闪存数据库系统的影响

近年来，一些新兴的存储技术为数据库系统带来新的机遇和挑战. 相变存储器、磁阻式随机存储器等一系列技术相继进入产品开发阶段. 它们可以按位访问且非易失，读写速度与内存相当，而存储密度比内存至少有一个数量级的提升. 这些新型存储技术被称为存储级内存(Storage Class Memory，SCM). 相变存储器(Phase Change Memory，PCM)是最具市场发展潜力的 SCM 设备，PCM 利用材料在不同温度下可逆转的相变来存储信息. 相比闪存，PCM 具有更高的读写速度、更低的能耗和更长的使用寿命. 表 9 对几种主要的存储技术进行了对比.

表 9　各种存储技术性能对比[56]

	页大小	读能耗	写能耗	写次数	读延迟	写延迟
内存	64 B	0.8 J/GB	1.2 J/GB	∞	20～50 ns	20～50 ns
PCM	64 B	1 J/GB	6 J/GB	10^6～10^8	～50 ns	～1 μs
闪存	4 KB	1.5 J/GB	17.5 J/GB	10^4～10^5	～25 μs	～500 μs
磁盘	512 B	65 J/GB	65 J/GB	∞	～5 ms	～5 ms

PCM 目前已经在通信设备、消费电子、PC 和其它嵌入式领域得到应用. 随着 PCM 技术的飞速发展，将 PCM 引入闪存数据管理也是未来的一个研

究热点,具体可分为以下几个研究点.

(1)SSD 和 PCM 混合存储.针对 SSD 随机写性能差和 PCM 可按位访问的特点,将 PCM 用作 SSD 的扩展可以提升数据库系统性能.文献[57-58]将 PCM 和 SSD 组成混合系统,利用 PCM 做 SSD 的日志存储区,所有对数据的修改都以日志的方式写入 PCM,日志写满后将数据进行合并产生新数据.将 PCM 和 SSD 组合应用可以有效降低系统能耗,延长 SSD 的可用寿命;

(2)PCM 用作内存的扩展缓存.PCM 支持更细粒度的数据访问,用 PCM 缓存上层应用对数据库元组级的修改不但可以减少数据写入量,还可以提高并发访问度.在文献[59]中,作者提出的 PCM-Logging 缓存策略支持元组级并发访问,数据库事务处理性能得到极大提升.PCMLogging 采用类似于影子页的事务处理办法,当事务对数据执行更新操作时,PCMLogging 不添加显式的日志记录,每次更新都会在内存保留数据的旧版本,当内存可用空间低于设定的阈值,再将数据写到 PCM.PCM 存储的数据会批量写入 SSD.元数据记录了数据更新的历史,在执行事务处理的时候,通过修改存储在内存和 PCM 中的元数据来保证事务的一致性和原子性.

5.4 软硬件一体化的数据库系统

传统的数据库技术,数据由磁盘读入内存进行查询、对比或修改等操作.随着企业业务数据的不断增长,大量数据需要从外存读取到服务器内存,这已经成为制约系统性能的瓶颈.另一方面,企业数据管理或者数据中心搭建数据业务处理平台的过程也是相当复杂,需要将操作系统、数据管理软件、存储硬件以及网络等组件优化整合在一起,来满足用户持续工作的高可用业务支持.将数据库、存储和服务器集成在一起不仅能够为企业提供便捷安装、易于维护的业务开发运营平台,还能以一种优化的方式协调资源分配、实现均衡负载、有效降低应用的复杂性、提高数据管理的性能.数据库机(Database Machine)的出现为解决上述问题提供了一个理想方案.早期的数据库机可以很好地保证数据库系统和操作系统的可靠性,但磁盘低速的 I/O 性能使得数据库机技术没有得到推广和应用.闪存的出现解决了数据库机 I/O 瓶颈问题,将闪存引入数据库机存储体系,采用智能缓存技术将经常访问的热数据透明的缓存在闪存和内存中,而活动性低的数据则被保留在低成本的磁盘.目前 Oracle、IBM 等相继推出软硬件集成产品,可以预见在大数据处理和分析领域,运用闪存技术将软硬件融合的数据库机一定会成为未来数据管理的重要技术.

6 结束语

硬件技术发展驱动软件技术革命,闪存等新型存储器件的出现为提高数据管理性能带来了机遇和挑战.应用新型存储介质解决海量数据管理是数据库技术未来的主要发展方向.因此,针对硬件特性,设计数据库体系结构和算法来解决数据库技术领域中查询优化,索引管理、事务恢复与并发控制等关键问题是非常必要的.本文首先讨论了闪存的特性和闪存转换层机制;对基于闪存的数据库系统相关研究进行了归类总结;详细讨论了闪存在混合存储数据管理中的重要作用.研究适合闪存特性的数据管理技术是数据管理研究领域的一个热点,目前该领域仍有许多问题需要做深入的研究.

参 考 文 献

[1] Chang L P, Kuo T W. Efficient management for large-scale flash memory storage with resource conservation. IEEE Transactions on Storage, 2005, 1(4): 381-418

[2] Wu C H, Kuo T W. An adaptive two-level management for the flash translation layer in embedded systems//Proceedings of the International Conference on Computer Aided Design (ICCAD'06). San Jose, USA, 2006: 601-606

[3] Gray J. Tape is dead, disk is tape, flash is disk, RAM locality is king//Proceedings of the Gong Show Presentation at Third Biennial Conference on Innovative Data Systems Research: Pacific Grove, USA, 2007: 1-12

[4] Lee S W, Moon B. Design of flash-based DBMS: An in page logging approach//Proceedings of the ACM SIGMOD International Conference. Beijing, China, 2007: 55-66

[5] Canim M, Bhattacharjee B, Mihaila G A, et al. An object placement advisor for DB2 using solid state storage. Proceedings of the VLDB Endowment, 2009, 2(2): 1318-1329

[6] Lee S W, Moon B, Park C, Kim J M, Kim S W. A case for flash memory SSD in enterprise database applications//Proceedings of the ACM SIGMOD International Conference. Vancouver, Canada, 2008: 1075-1086

[7] O'Neil E J, O'Neil P E, Weikum G. The LRU-K page replacement algorithm for database disk buffering//Proceedings of ACM SIGMOD International Conference. Washington, USA, 1993: 297-306

[8] Jiang S, Zhang X D. LIRS: An efficient low inter reference recency set replacement policy to improve buffer cache performance//Proceedings of the ACM SIGMETRICS Conference on Measurements and Modeling of Computer Systems. California, USA, 2002: 31-42

[9] Park S Y, Jung D, Kang J, Kim J S. CFLRU: A replacement algorithm for flash-memory//Proceedings of the International Conference on Compilers, Architecture and Synthesis for Embedded Systems. Seoul, Korea, 2006: 234-241

[10] Jung H, Shim H, Park S, et al. LRU-WSR: Integration of LRU and writes sequence reordering for flash memory. IEEE Transactions on Consumer Electronics, 2008, 54(3): 1215-1223

[11] Jung H, Yoon K, Shim H, et al. LIRS-WSR: Integration of LIRS and Writes Sequence Reordering for Flash Memory//Proceedings of International Conference on Computational Science and Its Applications. Kuala Lumpur, Malaysia, 2007: 224-237

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

[13] Jin P Q, Ou Y, Härder T, Li Z. AD-LRU: An efficient buffer replacement algorithm for flash-based databases. Journal of Data & Knowledge Engineering, 2012, 72(2): 83-102

[14] Lv Y F, Cui B, He B S, Chen X. Operation-aware buffer management in flash-based systems//Proceedings of the ACM SIGMOD International Conference. Athens, Greece, 2011: 13-24

[15] Chen F, Koufaty D A, Zhang X D. Understanding intrinsic characteristics and system implications of flash memory based solid state drives//Proceedings of the ACM SIGMETRICS International Conference on Measurements and Modeling of Computer Systems. Seattle, USA, 2009: 181-192

[16] Jo H, Kang J, Park S Y, et al. FAB: Flash-aware buffer management policy for portable media players. IEEE Transactions on Consumer Electronics, 2006, 52(2): 485-493

[17] Kim H, Ahn S. BPLRU: A buffer management scheme for improving random writes in flash storage//Proceedings of the 7th USENIX Conference on File and Storage Technologies. San Jose, USA, 2008: 239-252

[18] Seo D, Shin D K. Recently evicted first buffer replacement policy for flash storage devices. IEEE Transactions on Consumer Electronics, 2008, 54(3): 1228-1235

[19] Ou Y, Harder T, Jin P Q. CFDC: A flash-aware buffer management algorithm for database systems//Proceedings of the 5th International Workshop on Data Management on New Hardware (DaMoN). Rhode Island, USA, 2009: 435-449

[20] Jin P Q, Su X, Yue L H. A flexible simulation environment for flash aware algorithms//Proceedings of the 18th ACM Conference on Information and Knowledge Management. Hong Kong, China, 2009: 2093-2094

[21] Agrawal D, Ganesan D, Sitaraman R K, et al. Lazy adaptive-tree: An optimized index structure for flash devices. Proceedings of the VLDB Endowment, 2009, 2(1): 361-372

[22] On S T, Hu H B, Li Y. Flash-optimized B^+-Tree. Journal of Computer Science and Technology, 2010, 25(3): 509-522

[23] Xu C, Shou L D, Chen G, et al. Update migration: An efficient B-tree for flash-storage//Proceedings of the 15th International Conference on Database Systems for Advanced Applications. Tsukuba, Japan, 2010: 1-4

[24] Kang D W, Jung D, Kang J U. μ-Tree: An ordered index structure for NAND flash memory//Proceedings of the 7th ACM & IEEE International Conference on Embedded Software Salzburg. Salzburg, Austria, 2007: 144-153

[25] Zeinalipour D Z, Lin S, Kalogeraki V, et al. MicroHash: An efficient index structure for flash-based sensor devices//Proceedings of the 3th USENIX Conference on File and Storage Technologies. California, USA, 2005: 31-44

[26] Cui K, Jin P Q, Yue L-H. HashTree: A new hybrid index for flash disks//Proceedings of the 12th Asia-Pacific Web Conference. Busan, Korea, 2010: 45-51

[27] Li Y, He B S, Luo Q, Yi K. Tree indexing on flash disks//Proceedings of the 25th International Conference on Data Engineering. Shanghai, China, 2009: 1303-1306

[28] Yin S Y, Pucheral P, Meng X F. PBFilter: Indexing flash-resident data through Partitioned Summaries//Proceeding of the 17th ACM Conference on Information and Knowledge Management. California, USA, 2008: 1333-1334

[29] Do J, Patel J M. Join processing for flash SSDs: Remembering past lessons//Proceedings of the 5th International Workshop on Data Management on New Hardware. Rhode-Island, USA, 2009: 144-153

[30] Bausch D, Petrov I, Buchmann A. On the performance of database query processing algorithms on flash solid state disks//Proceedings of the 1st International Workshop Conference on IT Service Management and Its Support. Toulouse, France, 2011: 139-144

[31] Bausch D, Petrov I, Buchmann A P. Making cost-based query optimization asymmetry-aware//Proceedings of the 8th International Workshop on Data Management on New Hardware. Scottsdale, USA, 2012: 24-32

[32] Ailamaki A, DeWitt D J, Hill M D, Skounakis M. Weaving relations for cache performance//Proceedings of the 27th International Conference on Very Large Data Bases. Roma, Italy, 2001: 169-180

[33] Shah M A, Harizopoulos S, Wiener J L, Graefe G. Fast scans and joins using flash drives//Proceedings of the 4th International Workshop on Data Management on New Hardware. Vancouver, Canada, 2008: 17-24

[34] Tsirogiannis D, Harizopoulos S, Shah M A, et al. Query processing techniques for solid state drives//Proceedings of the ACM SIGMOD International Conference. Rhode Island, USA, 2009: 59-72

[35] Li Y, On S T, Xu J L, et al. DigestJoin: Exploiting fast random reads for flash-based joins//Proceedings of the 10th International Conference on Mobile Data Management. Taipei, China, 2009: 152-161

[36] Liang Zhi-Chao, Zhou Da, Meng Xiao-Feng. Sub-Join: Query optimization algorithm for flash-based database. Journal of Frontiers of Computer Science and Technology, 2010, 4(5): 401-409(in Chinese)
(梁智超，周大，孟小峰. Sub-Join：面向闪存数据库的查询优化算法. 计算机科学与探索，2010，4(5)：401-409)

[37] Myers D. On the use of NAND flash memory in high performance relational databases [M. S. dissertation]. Massachusetts Institute of Technology, Massachusetts, USA, 2008

[38] Lu Ze-Ping, Meng Xiao-Feng, Zhou Da. HV-Recovery: A high efficient recovery technique for flash-based database. Chinese Journal of Computers, 2010, 33(12): 2258-2266(in Chinese)
(卢泽萍，孟小峰，周大. HV-Recovery：一种闪存数据库的高效恢复方法. 计算机学报，2010，33(12)：2258-2266)

[39] Lu Z P, Qi X Y, Cao W, Meng X F. LB-Logging: A highly efficient recovery technique for flash-based database// Proceedings of the 13th International Conference of Web-Age Information Management. Harbin, China, 2012: 375-386

[40] Chen S M. FlashLogging: Exploiting flash devices for synchronous logging performance//Proceedings of the ACM SIGMOD Conference. Rhode Island, USA, 2009: 73-86

[41] On S T, Xu J L, Choi B, et al. Flag Commit: Supporting efficient transaction recovery in flash-based DBMSs. IEEE Transactions on Knowledge and Data Engineering, 2012, 24(9): 1624-1639

[42] Gokul S, Vijayan P, Mahesh B, Ted W. Extending SSD lifetimes with disk-based write caches//Proceedings of the 8th USENIX Conference on File and Storage Technologies. San Jose, USA, 2010: 101-114

[43] Yang P Y, Jin P Q, Yue L-H. Hybrid storage with disk based write cache//Proceedings of the DASFAA2011 Workshop on Flash-based Database Systems. Hong Kong, China, 2011: 190-201

[44] Ioannis K, Stratis V. Flashing up the storage layer. Proceedings of the VLDB Endowment, 2008, 1(1): 514-525

[45] Yang Q, Ren J. I-CASH: Intelligently coupled array of SSD and HDD//Proceedings of the 17th International Conference on High Performance Computer Architecture. San Antonio, Texas, USA, 2011: 278-289

[46] Chen F, David A, Zhang X D. Hystor: Making the best use of solid state drives in high performance storage systems// Proceedings of the 25th International Conference on Super Computing. Tucson, USA, 2011: 22-32

[47] Chen S M, Gibbons P B, Nath S. PR-join: A non-blocking join achieving higher early result rate with statistical guarantees//Proceedings of the ACM SIGMOD Conference. Indiana, USA, 2010: 147-158

[48] Canim M, Mihaila G A, Bhattacharjee B, et al. SSD buffer-pool extensions for database systems. Proceedings of the VLDB Endowment, 2010, 3(2): 1435-1446

[49] Do J, Zhang D H, Patel J, et al. Turbocharging DBMS buffer pool using SSDs//Proceedings of the ACM SIGMOD Conference. Athens, Greece, 2011: 1113-1124

[50] Kang W H, Lee S W, Moon B. Flash-based extended cache for higher throughput and faster recovery. Proceedings of the VLDB Endowment, 2012, 5(11): 1615-1626

[51] Luo T, Lee R, Mesnier M, et al. hStorage-DB: Heterogeneity-aware Data Management to Exploit the Full Capability of Hybrid Storage Systems. Proceedings of the VLDB Endowment, 2012, 5(10): 1076-1087

[52] Agrawal N, Prabhakaran V, Wobbe T. Design tradeoffs for SSD performance//Proceedings of the 2008 USENIX Annual Technical Conference. Berkeley, CA, USA, 2008: 57-70

[53] Ungureanu C, Debnath B, Rago S, Aranya A. TBF: A memory-efficient replacement policy for flash-based caches// Proceedings of the 29th International Conference on Data Engineering. Brisbane, Australia, 2013: 1117-1128

[54] Debnath B K, Sengupta S, Li J. SkimpyStash: RAM space skimpy key-value store on flash-based storage//Proceedings of the ACM SIGMOD Conference. Athens, Greece, 2011: 25-36

[55] Do J, Zhang D F, Patel J M, DeWitt D J. Fast peak-to-peak behavior with SSD buffer pool//Proceedings of the 29th International Conference on Data Engineering. Brisbane, Australia, 2013: 1129-1140

[56] Chen S M, Gibbons P, Nath S. Rethinking database algorithms for phase change memory//Proceedings of the 5th Biennial Conference on Innovative Data Systems Research. Asilomar, California, USA, 2011: 21-31

[57] Lee S W, Moon B, Park C, et al. Accelerating in-page logging with non-volatile memory. IEEE Data Engineering Bulletin, 2010, 33(4): 41-47

[58] Sun G Y, Joo Y, Chen Y B, et al. A hybrid solid-state storage architecture for the performance, energy consumption, and lifetime improvement//Proceedings of the International Conference on High Performance Computer Architecture. Bangalore, India, 2010: 1-12

[59] Gao S, Xu J L, He B S, et al. PCMLogging: Reducing transaction logging overhead with PCM//Proceedings of the 20th Conference on Information and Knowledge Management. Glasgow, Scotland, UK, 2011: 2401-2404

WANG Jiang-Tao, born in 1978, Ph. D. candidate. His current research interests include flash-based database systems, hybrid storage systems.

LAI Wen-Yu, born in 1989, M. S. candidate. His interests include query processing and optimization in flash-based database systems.

MENG Xiao-Feng, born in 1964, Ph. D., professor, Ph. D. supervisor. His research interests include web data management, cloud data management, mobile data management, social computing, flash-based systems, privacy-preserving.

Background

Flash memory emerges as a new electronic device with low access latency, no mechanical latency, low energy consumption and high resistance to shock when compared with magnetic disks. NAND flash memory has been widely used as storage alternative for embedded systems. In the past years, with the increasing of capacity and decreasing of unit price, flash-based solid state disks (SSDs) has been steadily expanded into personal computer and enterprise server markets. Flash memory is considered as the main storage device instead of disk in the next generation. Today, many different types of flash devices are finding their way into the storage hierarchy of database management systems (DBMS). However, flash device is not perfect; some of its physical properties, such as erase-before-write and limited write-erase cycles, which not only degrade the I/O efficiency on flash memory, but also wear the flash data blocks out quickly. Traditional disk-based database cannot make full use of high I/O performance of flash memory if we transfer it to flash without modification. Consequently, it is necessary to design and implement flash-aware database systems which can handle flash chip constraints elegantly during data managements.

In order to solve the above problems, more and more attentions have been given to this area. Recent years, researchers proposed several key technologies which are needed to be addressed on flash-based database systems. There are a lot of research topics still ahead, which need more people jump on the flash-based database researches. The content of this paper mainly provides a summary for related works and helps researchers discover the interesting issues.

The work was partially supported by the grants from the National Natural Science Foundation of China (61070055, 91024032, 91124001), the Fundamental Research Funds for the Central Universities, and the Research Funds of Renmin University of China (11XNL010), the National High Technology Research and Development Program (863 Program) of China (2012AA010701, 2013AA013204).