文章编号: 1671- 251X(2009) 06- 0056- 06

面向 Flash Memory 的高性能数据存储引擎的研究

周晓云[1,2], 覃雄派[3], 徐 钊[1]

(1. 中国矿业大学信电学院, 2. 徐州师范大学计算机科学与技术学院, 江苏 徐州 221008;
3. 中国人民大学信息学院, 北京 100872)

摘要: 传统的数据存储引擎对Flash Memory 数据的修改是通过页内更新技术实现的, 这将导致 Flash Memory 的性能下降及其磨损加剧。针对该问题, 文章提出了一种面向Flash Memory 的采用页外更新技术的多版本数据存储引擎 MV4Flash。该数据存储引擎采用多版本存储和垃圾回收机制, 所有数据的更新和修改都通过文件追加的方式进行, 适应了 Flash Memory 先擦除后写入的特点, 延长了设备寿命。采用 NDB Bench 对该数据存储引擎进行测试的结果表明, MV4Flash 与传统的 InnoDB 相比, 事物处理性能有较大的提升, 更适合于数据规模大、实时性要求高的应用系统。

关键词: Flash Memory; 数据存储引擎; 页内更新; 页外更新; 多版本; 垃圾回收; NDB Bench
中图分类号: **TP333.8** 文献标识码: **A**

Research of a Data Storage Engine with High Performance Oriented Flash Memory

ZHOU Xiao-yun[1,2], QIN Xiong-pai[3], XU Zhao[1]

(1. School of Information and Electrical Engineering of CUMT., Xuzhou 221008, China.
2. College of Computer Science and Technology of Xuzhou Normal University, Xuzhou 221008, China.
3. School of Information of Renmin University of China, Beijing 100872, China)

Abstract: Traditional data storage engine updates data by in place update technique which would induce performance reduction and wear aggravation of Flash Memory. Aiming at the problem, the paper proposed a multi version data storage engine MV4Flash oriented Flash Memory which used out place update technique. The data storage engine uses the mechanism of multi version storage and garbage collection and updates and modifies all data by file superaddition mode, which is suitable for the characteristic of erase before write of Flash Memory and can prolong its life. The result of performance test for the data storage engine with NDB Bench showed that compared with traditional InnoDB, the MV4Flash has higher performance of transaction processing, so the MV4Flash is more suitable for application system with large data volume and high real-time demand.

Key words: Flash Memory, data storage engine, in place update, out place update, multi version, garbage collection, NDB Bench

0 引言

Flash Memory[1] 由若干个块(容量一般为 128 KB)组成, 每个块包含一定数量的页面(容量一般为 2 KB)。读写操作的最小单位是页面, 而擦除操作的最小单位是块。页面只有在擦除以后才能改写。Flash Memory 在擦除一定次数之后即损坏。如果每次更新数据都要擦除一整块, 改变块内某个页面的内容, 然后更新整个块, 这种页内更新(In Place Update)技术导致的后果则是加速 Flash Memory 的损坏。针对 Flash Memory 的特点, 一般采用页外更新(Out Place Update)技术更新数据, 读取需要更新的页面, 然后改变页面内容, 写入到新

收稿日期: 2009- 02- 19

作者简介: 周晓云(1971-), 女, 副教授, 中国矿业大学信电学院在读博士研究生, 研究方向为数据库查询优化、并行数据库、内存数据库。E-mail: zhouxy@xznu.edu.cn

的页面地址, 最后将老的页面标记为无效页面(或者死页面)。经过一段时间的读写操作和擦除操作之后, Flash Memory 上的空闲空间减少, 同时有很多的无效页面没有得到利用。将无效的页面重新回收利用称为“垃圾回收”。在当前的技术条件下, Flash Memory 大约经过 1 000 000 次的擦除之后即告损坏, 损坏的块也可以写入数据, 但是发生错误的几率很高[2]。因此, 必须采用某种损坏水平(Wear Leveling)管理技术将 Flash Memory 的擦除操作均匀地分布到整个 Flash Memory 的各个块上, 使得每个块具有大致相同的擦除记录, 延长整个设备的寿命。

面向 Flash Memory 的文件系统主要有 2 类: 一类是块设备模拟策略, 比如 Compact Flash[3]、FTL/FTLite[3~4]、SmartMedia; 另一类是 Native 文件系统策略, 如 JFFS/JFFS2、YAFFS/YAFFS2。Native 文件系统策略的思想来源于传统的 Log Structured 文件系统[5]。这些文件系统针对 Flash Memory 的特点设计, 避免使用页内更新技术更新数据, 而是采用页外更新技术更新数据。

本文将提出一种基于页外更新技术的面向 Flash Memory 的多版本数据存储引擎 MV4Flash (Multi Version for Flash Memory), 该数据存储引擎采用多版本存储和垃圾回收机制, 所有数据的更新和修改都通过文件追加的方式进行, 适应了 Flash Memory 先擦除后写入(Erase before Write)的特点, 延长了设备寿命, 同时通过精心设计的数据结构提高了事物处理性能。

1　MV4Flash 存储引擎介绍

1.1　系统结构

图 1 为一般的 DBMS(Database Management System, 数据库管理系统)层次结构, 上层为 SQL 引擎, 下层为存储引擎。本文所讨论的数据存储引擎位于 SQL 引擎之下, 负责数据的存储、垃圾回收、系统恢复以及存储引擎一级的操作并发控制任务。与传统的数据存储引擎的主要区别是该数据存储引擎通过多版本记录数据的更新, 新记录和修改的记录都在文件末尾进行添加, 无效的老记录通过垃圾回收机制进行处理。

1.2　数据结构

为了在文件里写入数据并且保持查询的高效率, 必须对数据文件(Data File)的组织方式进行精心的安排[6]。MV4Flash数据文件的组织方式: 所

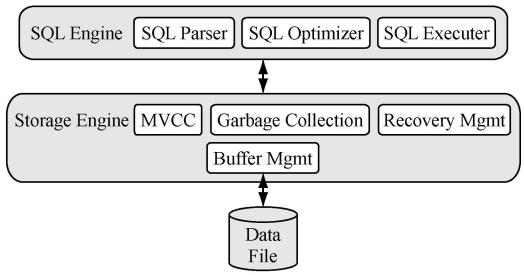

图 1　DBMS 的层次结构图

有记录顺序地写入文件的末尾, 记录可以是定长的, 也可以是不定长的, 并且原有记录的所有更新都以新记录的方式写入, 老记录在相关事务提交后, 成为可以回收的垃圾空间。

很显然, 这样的组织方式如果没有增加其它的辅助存取机制, 其查询的效率是很低的。为了提高查询的效率, 对于每张数据库表, 在 Flash Memory 上维护 1 个以该表名命名的表文件(Table File)。表文件里存放一系列定长的句柄, 句柄指向属于该表的数据文件, 并且唯一地指向 1 行记录。每个句柄具有一个定长的结构, 在表文件里存放的就是一系列的定长句柄, 通过文件偏移量可以快速地寻找到记录的句柄, 所以表文件的记录句柄编号(Table File Number)具有不变性, 可以当作记录的 Row ID。为了简化缓冲区的管理, 可以利用操作系统的 mmap()函数将整个表文件影射到内存的地址空间, 这样可以加快句柄的存取速度。表文件与数据文件的关系如图 2 所示。

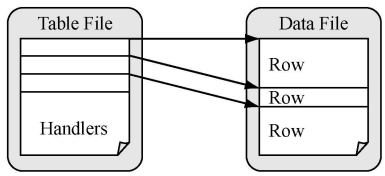

图 2　表文件与数据文件的关系图

表文件里的记录句柄由 3 个部分构成, 分别是数据文件号(Data File Number, 4 B)、文件偏移量(Data File Offset, 6 B)以及记录的大小(Record Size, 4 B)。

1.3　事务处理

所有记录的更新都是通过在数据文件的末尾添加新记录的方式实现的。这样做有以下优势: 在空间回收之前, 数据文件里保存记录的以前版本, 对于复制(Replication)、备份以及恢复来讲, 可以很容易地构造一个一致的快照(Snapshot)。

结合以上的数据结构, 在并发控制方面采用多版本的控制方法[7]。该方法的最大优势: 对于读取事务来讲, 完全避免了加锁的需要。而基于锁的并

发控制方法,死锁的主要来源就是读锁(Read Lock),通过避免读事务的加锁,可以提高系统的吞吐量。

为了找到同一个记录的不同版本,并且为多版本并发控制提供支持,只需要对记录句柄进行简单扩展即可。实现的机制:记录的不同版本通过链接连起来,需要在记录句柄里加上 2 个域,即下一个记录版本的句柄以及事务的 ID。当事务要读取 1 个记录时,该事务顺着该链接查找记录的不同版本,通过对提交该记录的事务 ID 和当前存取该记录的事务 ID 的判别:如果该记录对本事务是可见的,且该事务在本事务开始之前提交,则返回该记录。

为了提交事务,需要维护一个事务文件(Transaction File)。每个事务的提交需要往事务文件写 2 个记录,分别是 BEGIN 记录和 END 记录。END 记录有 2 种类型:COMMIT 和 ABORT,表示事物最后是提交还是退出。对于完整提交的事务,该事务更新的记录对于其它事务来讲立即可见;而对于退出的事务,该事务所更新的记录对于其它事务来讲不可见,并通过垃圾回收机制进行空间回收。

1.4 垃圾处理

垃圾处理的目的是对存储空间进行回收,主要回收 2 类记录的空间,分别是一定时间以前的版本以及失败事务更新的记录。垃圾处理过程:标记需要回收的记录,回收线程扫描事务文件,对于每个事务,查看是否有需要释放的符合上述 2 类记录条件的记录,如果有,则设置其垃圾标志位、释放句柄空间、从记录版本链表里删除该记录版本、修改索引,最后增加数据文件头的垃圾计数器。该计数器用来确定这个数据文件的垃圾记录是否已经到达一个阈值,从而进行真正的空间回收。当事务文件里面的所有事务都进行了相应的标记处理以后,即可删除该事务文件。

当数据文件的垃圾记录数量达到一定的比例时,需要进行垃圾回收,回收的办法是将有效记录重新复制到另外一个文件,处理完成后,删除老的数据文件。虽然把记录从一个文件拷贝到另外一个文件是一个 I/O 密集的操作,但是垃圾回收线程的优先级是比较低的,不会对事务处理构成性能上的大影响,而且通过从 Row ID 到句柄以及从句柄到记录的两级影射,保证了拷贝过程不影响记录的访问。这种垃圾回收机制,数据被拷贝到新的文件(存储空间),老文件由于连续存储,可以以块(Flash Memory Block)为单位进行擦除操作和重新归入空闲空间加以利用。

1.5 数据恢复

传统的数据存储引擎采用数据文件和日志文件相结合的方式实现数据恢复。当系统失败重启时,首先读取检查点文件,然后利用日志文件撤销失败的事务,重做成功提交的事务,从而保证数据库的原子性和持久性[8]。

在系统失败重启的情况下,MV4Flash 存储引擎的恢复过程是即时完成的。因为提交事务时,数据已经到达存储设备,当系统重启时,不需要做额外的工作来保证事务的持久性,通过多版本的管理,事务的历史信息已经在数据文件里;而对于没有真正提交成功的数据,则由垃圾处理机制进行标记和回收。由此可以看出,使用该技术的 DBMS 重新启动的速度是非常快的。

对于介质失败的数据恢复,需要采用存储冗余的方法来避免数据丢失[8]。

1.6 MV4Flash 存储引擎的优势

(1) 减少 I/O 操作,提高事务处理效率。

(2) 只需要顺序地将数据写入到数据文件里面,实现简单;利用句柄技术,完全可以支持不定长的记录;通过索引技术、句柄技术,可以提高数据查找的效率。

(3) 修改记录采用的方法是添加新的记录,结合多版本并发控制机制,避免了读事务的加锁需要,同时利用垃圾空间回收技术,可以保证无用记录空间的释放。

(4) 由于提交事务时,数据的修改已经到达 Flash Memory,且完成了持久性(Durability)的保证,所以缓冲区里的记录是最新的,这样可以简化缓冲区的管理工作。

(5) 由于算法固有的特点,数据恢复是即时完成的,重启动的过程比传统的持久化技术更快,因为传统的持久化技术必须扫描日志文件进行失败事务的回滚和已提交事务的重做。

(6) 数据文件保留的是对数据库的数据修改历史,可以利用这种持久化机制获得一个历史数据的一致视图。

2 实验及结果

笔者采用 Network Database Benchmark(简称 NDB Bench)[9] 对 MV4Flash 存储引擎进行测试。NDB Bench 是 Nokia 公司开发的针对电信类应用的数据库性能测试软件。除了 Nokia 公司,Solid

Tech 公司也开发了类似的测试软件——Telecom One(简称 TM1)[10]。

NDB Bench 对移动电话网络的本地用户注册(Home Location Register, HLR)数据库进行模拟性能测试。HLR 数据库存储用户的属性数据、该用户订购的服务以及对移动网络使用的历史信息。该测试包含了与性能紧密关联的典型查询。根据该测试标准,读类型的事务实时响应时间在 5 ms 之内、更新类型的事务实时响应时间在 100 ms 之内是可以接受的。

HLR 模拟数据库包括 4 个表: Subscriber 表、Access Info 表、Special Facility 表和 Call Forwarding 表。Subscriber 表记录用户的基本信息; Access Info 表记录用户对网络的使用情况; Special Facility 表记录用户订购的服务; Call Forwarding 表记录每个服务的呼叫转移信息。HLR 模拟数据库结构如图 3 所示。

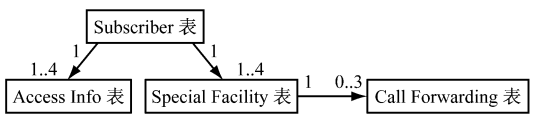

图 3　HLR 模拟数据库结构图

NDB Bench 定义了 7 类主要的事务,在测试软件运行过程中(一般选择 2 h),这些事务的混合比例关系符合事先定义的比例要求。测试的性能指标有 2 个:一个是描述系统的吞吐量,为最大有效吞吐量(Maximum Qualified Throughput, MQTh);另一个是各类事务的响应时间分布情况。事务分为读事务和写事务 2 类,读事务占 80%(包括 35% 的 Get Subscriber Data, 10% 的 Get New Destination, 35% 的 Get Access Data),写事务占 20%(包括 2% 的 Update Subscriber Data, 14% 的 Update Location, 2% 的 Insert Call Forwarding, 2% 的 Delete Call Forwarding)。上述事务的编号: 1 号为 Delete Call Forwarding、2 号为 Insert Call Forwarding、3 号为 Update Location、4 号为 Update Subscriber Data、5 号为 Get Access Data、6 号为Get New Destination、7 号为 Get Subscriber Data。

实验采用的硬件平台以及相关的软件配置如表 1 所示。

表 1　实验采用的硬件平台以及相关的软件配置表

硬件平台	硬件型号: HP ProLiant DL380 G4 CPU: Intel Xeon 3 600 MHz × 2, L1 Cache 16 KB, L2 Cache 2 048 KB MEM: 5 GB DISK: MSD-P35 NAND Flash based Solid State Disk
软件配置	OS(Kernel): Red Hat Enterprise Server (2.6.9-5.ELsmp) Database: MySQL 5.027 Beachmark: NDB Bench

笔者采用MySQL 作为测试的数据库平台。MySQL 具有清晰的存储引擎接口,针对不同的应用类型,可以挂接不同的存储引擎。MySQL 内置的 InnoDB 存储引擎采用的是传统的数据存储引擎技术,笔者将 MV4Flash 存储引擎与 InnoDB 存储引擎进行对比测试,以考察两者的性能差别。

实验的数据量是主表 Subscriber 表,其具有 5 000 000行数据,其它关联表按照测试标准自动生成。实验的结果如表 2、表 3 和表 4 所示。表中, Tx 为 Transaction,如 Tx7 表示第七号事务; Avg 为事务平均响应时间, 90% 百分点表示 90% 的事务的响应时间,如 Avg/90% 百分点(6.15/9)表示事务平均响应时间为 6.15 ms, 90% 的事务的响应时间在 9 ms之间; MQTh 为系统吞吐量,如 881 tr/s 为每秒钟881 个事务; Thread Numbers 为线程数量,即并发事务数量。

表 2　InnoDB 存储引擎总吞吐量及其响应时间分布表

事务编号		Threads Numbers				
		2	8	16	32	64
Tx1	Avg/90% 百分点	8.31/10	19.07/25	45.64/99	90.20/264	170.27/364
Tx2	Avg/90% 百分点	6.15/9	25.01/38	72.40/137	265.13/905	622.00/1000
Tx3	Avg/90% 百分点	9.26/8	18.86/22	31.49/48	54.67/201	102.26/297
Tx4	Avg/90% 百分点	12.45/11	27.23/32	43.00/76	138.66/496	162.77/357
Tx5	Avg/90% 百分点	1.29/2	10.31/15	23.03/35	51.40/158	107.95/301
Tx6	Avg/90% 百分点	3.11/4	20.65/19	34.47/43	81.27/319	102.16/293
Tx7	Avg/90% 百分点	1.35/3	11.68/16	25.88/38	54.12/188	111.83/308
MQTh		881 tr/s	896 tr/s	899 tr/s	896 tr/s	848 tr/s

表 3 MV4Flash 总吞吐量及其响应时间分布表

事务编号		Threads Number				
		2	8	16	32	64
Tx1	Avg/90% 百分点	4.25/2	11.51/4	31.72/9	73.40/126	140.70/264
Tx2	Avg/90% 百分点	4.13/3	26.39/7	45.92/86	89.94/265	186.18/379
Tx3	Avg/90% 百分点	0.67/0	4.18/2	6.91/4	23.05/10	103.56/41
Tx4	Avg/90% 百分点	1.72/3	22.14/5	33.85/14	78.40/254	141.18/450
Tx5	Avg/90% 百分点	1.00/0	1.72/1	6.75/4	12.83/8	24.05/19
Tx6	Avg/90% 百分点	6.51/5	14.71/6	28.29/11	50.19/166	122.87/270
Tx7	Avg/90% 百分点	0.11/0	0.15/2	0.81/4	7.87/9	21.00/18
MQTh		1 483 tr/s	1 511 tr/s	1 478 tr/s	1 307 tr/s	1 296 tr/s

表 4 最大有效吞吐量比较表

Thread Numbers	2	8	16	32	64
InnoDB(5 000 000 行)	881 tr/s	896 tr/s	899 tr/s	896 tr/s	848 tr/s
MV4Flash (5 000 000 行)	1 483 tr/s	1 511 tr/s	1 478 tr/s	1 307 tr/s	1 296 tr/s
Performance Improve Ratio	68.37%	68.57%	64.37%	45.77%	52.74%

使用 InnoDB 存储引擎,5 000 000 行数据的装载时间为 16 h 31 min。运行 NDB Bench,当线程的数量达到 8 以上时,读事务的响应时间大于 5 ms 的实时限制要求,而当线程的数量达到 32 以上时,写事务的响应时间大于 NDB Bench 的 100 ms 的实时限制要求。当线程数量为 16 时,达到最大吞吐量,为每秒钟 899 个事务。

而使用 MV4Flash 存储引擎,5 000 000 行数据的装载时间仅为 4 h 7 min。当线程数量达到 16 时,读事务的响应时间仍然满足 5 ms 的实时限制要求(Get New Destination 事务的响应时间除外,线程数为 8 时,即达到 14.71 ms),当线程的数量达到 32 以上时,写事务的响应时间即大于 NDB Bench 的 100 ms 的实时限制要求。当线程数量为 8 时,达到最大吞吐量,为每秒钟 1 511 个事务。从结果数据可看出,MV4Flash 存储引擎在 Get New Destination 事务上的响应时间增长比较快,当线程数量为 8 时已经大于 5 ms 的限制要求,这是在没有进行系统优化的基础上获得的性能数据,可以通过索引和缓存的优化来进一步提高其性能。

另外,通过表 4 可以看出,MV4Flash 存储引擎与传统的 InnoDB 存储引擎相比,最多有 69% 的性能提升(8 threads),最少也有 46% 的性能提升(32 threads)。

3 结语

MV4Flash 存储引擎采用适用于 Flash Memory 的数据管理策略和事务处理方法,不仅减少了事务的响应时间,还极大地提高了系统的总吞吐量。实验结果表明,该存储引擎比传统的存储引擎更加适合于 Flash Memory,满足数据规模大、事务类型简单、实时性高的应用要求。

笔者下一步的研究计划:首先考虑利用数据的划分,将数据分布到不同的设备,也就是将 1 个表的单个数据文件分解成多个数据文件,进行并行的 I/O 操作,以利用硬件的 I/O 带宽提高性能;然后利用 CPU 的多核心和硬件多线程技术提高系统的并发度,并且通过隔离不同线程的 I/O 操作,进一步提高系统的性能。

参考文献:

[1] KIMURA K, KOBAYASHI T. Trends in High Density Flash Memory Technologies [C]//IEEE Conference on Electron Devices and Solid State Circuits, 2003, Hong Kong: 45~ 50.

[2] HACHMAN M. New Samsung Notebook Replaces Hard Drive with Flash[EB/OL]. [2006- 06- 20]. http://www.extreametech.com.

[3] INTEL CORPORATION. Understanding the Flash Translation Layer (FTL) Specification [EB/OL]. [2005- 12- 01]. http://www.intel.com.

[4] INTEL CORPORATION. FTL Logger Exchanging Data with FTL System[EB/OL]. [2006- 06- 05]. http://www.intel.com.

[5] ROSENBLUM M, OUSTERHOUT J K. The

综述

文章编号: 1671- 251X(2009) 06- 0061- 04

新型液位检测技术的现状与发展趋势

杨朝虹[1], 李 焕[2]

(1. 北京矿冶研究总院, 2. 北京慧点科技开发有限公司, 北京 100044)

摘要: 文章介绍了工业生产过程中采用的光纤液位计、磁致伸缩液位计、音叉液位限位开关、差压式液位计、雷达液位计、伺服型浮子液位计等几种较新的液位检测仪表及其检测方法, 并对几种液位计的特点进行了分析比较。文章最后指出, 液位自动检测技术一方面需采用新的测量原理, 开发新的液位检测仪表, 扩大检测的手段, 另一方面需朝着微机化和智能化方向发展。

关键词: 液位检测; 液位计; 液位传感器; 自动检测

中图分类号: **TP216** 文献标识码: **A**

Present Situation and Developing Trend of New Type of Measuring Technology for Liquid-level

YANG Chao-hong[1], LI Huan[2]

(1. Beijing General Research Institute of Mining Metallurgy, Beijing 100044, China.
2. Beijing Smartdot Science and Technology Co., Ltd., Beijing 100044, China)

Abstract: The paper introduced some new kinds of measuring instruments for liquid-level used in industry production process and their measuring methods, such as fiber liquid-level meter, magnetostriction liquid-level meter, fork liquid-level limit switch, differential pressure type liquid-level meter, radar liquid-level meter and servo type float liquid-level meter, and analyzed and compared the characteristics of the several liquid-level meters. At last, it indicated that automatic measuring technology for liquid-level on the one hand must use new measuring principle, develop new measuring instruments for liquid-level and expand measuring way, on the other hand, it must develop facing on micro-computerization

收稿日期: 2009- 02- 19

作者简介: 杨朝虹, 男, 硕士, 工程师, 现主要从事仪器检测产品的研发工作。E-mail: yang_zhh@bgrimm.com

Design and Implementation of a Log-structured File System [J]. ACM Transactions on Computer Systems, 1992, 10(1): 26~ 52.

[6] GERHARD W, GOTTFRIED V. Transactional Information Systems: Theory, Algorithms and the Practice of Concurrency Control and Recovery[M]. New York: Morgan Kaufman Publishers, 2001.

[7] GRAY J, MCJONES P, BLASGEN M, et al. The Recovery Manager of the System R Databasc Manager [J]. ACM Computing Surveys (CSUR), 1981, 13(2): 223~ 242.

[8] PATTERSON D A, GIBSON G, KATZ R H. A Case for Redundant Arrays of Inexpensive Disks (RAID) [C]//ACM SIGMOD International Conference on Management of Data, 1988, New York: 109~ 116.

[9] NOKIA. Network Database Benchmark on Open Source Project[EB/OL]. [2006- 11- 25]. http://hoslab.cs.helsink.fi/savane/projects/ndbbenchmark.

[10] STRANDELL T. Open Source Database Systems: Systems Study, Performance and Scalability [D]. Helsinki: University of Helsinki, 2003.