基于列式存储的闪存数据库查询优化策略

邢玉钢[1],王曼丽[1],王翰虎[1,2],陈 梅[1]
(1.贵州大学 计算机科学与信息学院,贵州 贵阳 550025;
2.贵州星辰科技开发有限公司,贵州 贵阳 550001)

摘 要:随着闪存的性能和容量的提高,闪存数据库的研究受到了广泛的关注。闪存具有随机读快、擦写代价大的特性,如何利用闪存的这个特点,减少查询时写闪存的频率,提高数据库的查询效率是闪存数据库查询研究的重要问题。已有的很多查询优化算法主要是按传统的行式存储方式来进行优化的,有一定的局限性。文中采用列式存储,提出了一种新的连接算法。该算法最小化了中间临时表,使投影数据量大大减少,达到了少写闪存、减小擦除代价的目的,从而提高了查询的效率。通过与传统行式算法的比较实验,证明了该算法的优越性。

关键词:闪存;闪存数据库;按列存储;查询优化;索引表
中图分类号:TP311.13 **文献标识码**:A **文章编号**:1673-629X(2011)12-0131-04

Query Optimization Strategies of Flash Memory Database Based on Column Storage

XING Yu-gang[1],WANG Man-li[1],WANG Han-hu[1,2],CHEN Mei[1]
(1. College of Computer Science and Information,Guizhou University,Guiyang 550025,China;
2. Guizhou Xingchen Science and Technology Development Company Limited,
Guiyang 550001,China)

Abstract:With flash memory to improve performance and capacity,flash memory database research has been widespread concern. Flash-memory has the characteristic of random read fast and erased characteristics of a large cost. It's an important issues of flash memory database query that how to use flash feature and reduce the frequency of queries written. Many query optimization algorithms have been the main line according to the traditional way. This has limitations. In this paper,column storage,a new connection algorithm is proposed that minimizes the staging table,so that projection data is greatly reduced. It achieves the purpose of less write flash memory and reducing the cost of erasing,so this algorithm improves the query efficiency. A experiment with the comparison of a traditional algorithm proves the superiority of this algorithm.

Key words:flash memory;flash database;stored by column;query optimization;index table

0 引 言

闪存具有随机读快、非易失、能耗低、携带方便、无噪音的优良特性,而且闪存价格不断下降,容量不断增长,Jim Gray曾预测"闪存是磁盘,磁盘是磁带"[1],因而闪存逐渐取代磁盘作为存储介质成为未来的发展趋势。

目前,闪存数据库查询优化方面的研究还比较少,已提出的查询优化算法基本都是基于行存储模式的。其中Jaeyoung Do回顾了传统的连接算法[2],并进行了磁盘与SSD的性能比较,获得了令人惊讶的结果。Mehul A. Shah通过实验分析了传统的外连接和散列方法在闪存上的性能。Daniel Myers提出了subset算法[3]来提高查询的外排序性能。在Daniel Myers的基础上,Li Yu提出的针对固态硬盘设计的DigestJoin算法[3]充分发挥了闪存随机读快的优点,通过增加读操作来提高连接的性能。梁智超提出了Sub-Join算法[4],通过构建子连接表来减少数据的读取,并将数据的随机写转化为连续写,从而提高了连接速度。Mehul A. Shah还提出了将关系按列存储(PAX Layout)在闪存上的RARE-join算法[5],该算法通过减少执行投影和连接时需要读取的数据量来加快连接速度。此外还有很多研究有一定的借鉴意义,如Z. Li的使用连

收稿日期:2011-05-22;**修回日期**:2011-08-25
基金项目:贵阳市2010年科技攻关项目([2010]筑科工合同字第28号);贵州大学2011年研究生创新基金资助项目(校研理工[2011039])
作者简介:邢玉钢(1982-),男,河南洛阳人,硕士研究生,CCF会员,研究方向为数据库、软件工程;王翰虎,教授,研究方向为数据库系统、分布式系统、面向对象方法;陈 梅,教授,研究方向为数据库、软件工程。

接索引的快速连接[6]，T. H. Merrett 等的连接操作的取页策略[7]，李建中等研究了基于三级存储器的海量关系数据库的 Join 算法[8]，基于数据仓库的星型模式，蒋旭东给出了一种新的多表连接算法[9]，李静等针对现有列存储系统中列的连接存在的问题，提出了一种连接策略选择方法[10]，苏金泉等通过分析 Merge-Join 算法和 SDC 算法[11]，提出一种在处理速度上更快的 Join 算法[12]。

文中从闪存擦写代价大的特性出发，提出一种基于按列存储的查询优化算法，通过变行式存储为列式存储来最小化查询临时表，从而达到减少闪存擦写次数的目的，并通过实验验证该算法的性能。

1 列式存储简介

传统的行式数据库，是根据元组按行存储的，维护大量的索引和物化视图无论是在时间（处理）还是空间（存储）方面成本都很高。而列式数据库恰恰相反，列式数据库的数据是根据属性按照列存储，每一属性列单独存放。

投影数据时只访问查询涉及的属性列，大大降低了系统 I/O 开销。又由于列式存储的数据具有相同的数据类型，相邻存储的数据之间相似性比较高，具有更好的压缩率，而压缩的数据更能够减少 I/O 开销。虽然按列存储在连接时增加了随机读取操作，但由于闪存具有比较快的随机读取速度，所以采用按列存储可以有效地提高连接的速度。这里，每一个属性列对应闪存中的一个属性块，每个属性块中包含所需数量的属性页，所有的属性页是定长结构的，由于属性类型不同，因此每个属性页包含的属性值是不同的，如果数据量比较大，存储管理器就会自动分配新的属性页给属性块，属性值在属性块内连续存储，这样就实现了每一属性列单独存放，数据即是索引。文中算法是基于图 1 所示的列式存储结构：

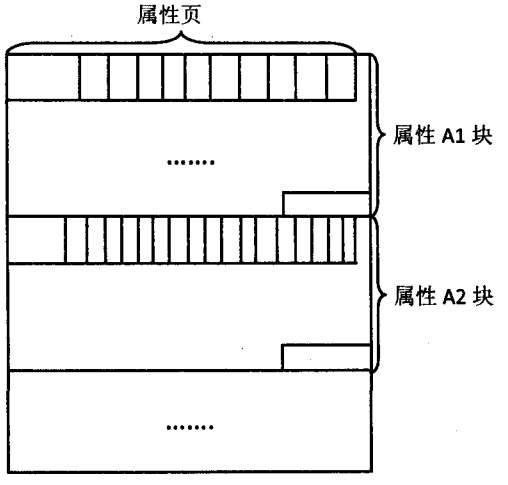

图 1 列式结构

2 基于按列存储的查询优化策略

2.1 算法整体流程

表：$X = (a_1, a_2, \cdots, a_n)$，$Y = (b_1, b_2, \cdots, b_n)$，$a_i, b_i$ 分别是表 X 和表 Y 的属性。tid_x，tid_y 分别为表 X 和表 Y 的元组 ID。算法的总体流程如图 2 所示。首先，根据元组 ID 和连接列投影得到属性-元组表。然后，根据元组 tid 构建页-元组索引表。最后，根据页-元组索引表从原始表中读取其他列的数据，得到最终连接结果。

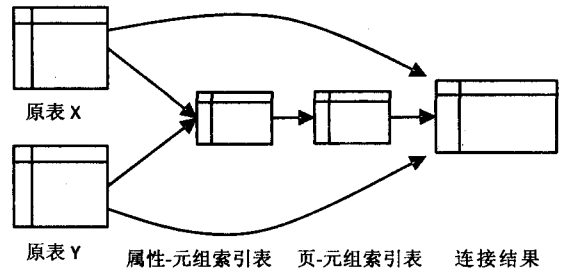

图 2 整体流程图

文中采用在列存储模式下基于页的取页策略来达到加快查询速度，同时减少擦写闪存次数的目的。通过构建属性-元组索引表和页-元组索引表来尽量减小查询处理过程中产生的临时中间表，从而达到减少将中间表写回闪存的次数或避免写中间表到闪存，这样也就减少了擦写闪存的代价。

2.2 构建属性-元组索引表

首先，对表进行投影。由于数据库原始表是按属性列进行存储，参与连接的属性都是连续地存储在一起，因此可将包含属性列的闪存页直接读入内存。若表是按照连接属性列有序，则直接采用折半查找法对有序属性列进行连接；若连接属性列无序，则采用排序-归并算法先对连接属性列进行排序，再在归并阶段同时进行连接，从而得到按属性有序的属性-元组索引表。这与传统的先排序，再归并，最后再连接的过程相比，减少了中间结果表，也就减少了写闪存的次数或避免了写闪存。

属性-元组索引表的存储结构为：ATindex（attr，tid_x，tid_y），其中 attr 是连接属性，tid_x 是 X 表的元组 id，tid_y 是表 Y 的元组 id。构建属性-元组索引表的过程，省去了 DigestJoin 算法中的数据抽取过程，也省去了 Sub-Join 算法的子表构建过程，减少了数据的读取次数。因为属性-元组索引表较小，所以可以将属性-元组索引表尽可能地放入内存缓冲区，从而总体上减少了写闪存的次数。

算法的具体代码如表 1 所示。

2.3 构建页-元组索引表

假设表 X 按行存储有 10000 个元组记录，列属性有 50 个，每个闪存页可以存储 50 个数据，则按列存

储,每个属性列将占用 200 个闪存页。对于这样的情况,如果采用顺序读取每一页来获取所需的数据,I/O 开销比较大,而且可能出现多次重复读取同一页获取数据的情况,这样就会造成额外的 I/O 开销。因此,这一阶段,采用构建页-元组索引表的方法来降低 I/O 开销。

表 1　算法 1 具体代码

算法 1
Input JC_x of T_x, JC_y of T_y
Output JoinProject table T
Project JoinColumn
Sort JC_x to get SJC_x;
Sort JC_y to get SJC_y;
Create table ATindex(attr, tid_x, tid_y)
For each $attr_x$ in SJC_x, $attr_y$ in SJC_y
If($attr_x$ = = $attr_y$)
Insert <$attr_x$, tid_x, tid_y> into ATindex;
Get next $attr_x$, $attr_y$;
Else If $attr_x$ < $attr_y$
Get next $attr_y$
Else
Get next $attr_x$
End
Output join result T

为避免读取无用页,首先分别根据属性-元组索引表 ATindex 的 tid_x 和 tid_y 获取相应元组所在的页号 pageid,剔除掉无用页,同时获取非连接属性列的 ID,构建页-元组索引表。

页-元组索引表的结构为:IndexPT(other_attrID, pageid, tid),其中 other_attrID 是属性列 ID,pageid 是数据库表的元组 tid 所在的闪存页 ID。如果一个属性页的连接属性命中率比较高,那么可能会出现一个页被重复读取的情况,这里采取对索引表按 pageid 排序的方法来避免这种情况的发生。通过排序,生成两张按 pageid 有序的页-元组索引表:$IndexT_x$($attrID_x$, $pageid_x$, tid_x) 和 $IndexT_y$($attrID_y$, $pageid_y$, tid_y)。

算法具体代码如表 2 所示。

2.4　数据回取

利用有序页-元组索引表,根据 other_attrID 定位属性列的位置,然后根据 pageid 顺序读取闪存页到内存,再根据 tid 读取页内的连接元组的相应属性值,之后根据属性-元组索引表中(tid_x, tid_y) 进行元组连接,并将结果插入属性-元组索引表中,得到最终连接结果。

算法的具体代码如表 3 所示。

表 2　算法 2 具体代码

算法 2
Input attribute_tupe index table ATindex, Original table Tx, Ty
Output page_tuple index table PTindex
While scanning tidx and tidy inATindex
scan Tx, Ty;
get page_tuple index table IndexPTx and IndexPTy;
End
Sort IndexPTx and IndexPTy base on pageid
Output page_tuple index table IndexPTx and IndexPTy

表 3　算法 3 具体代码

算法 3
Input　page_tuple index table IndexPTx and IndexPTy,
Original table T_x, T_y
Output join result R
Locate attribute column according to $attrID_y$;
Fetch pagex and pagey according to pageidx and pageidy;
Read attribute values from page according to tid;
Insert attribute values into ATindex;
Output join result R

3　实验结果与分析

3.1　实验环境

本实验使用计算机配置为:Intel ® Core™ Duo Cpu 2. 10GHz,4G 内存。操作系统是 Windows 7 旗舰版,编程语言 C#,实验平台 VS2008,实验设备一个 40GB 固态盘,数据库采用的是 Oracle Berkeley DB。由于文中算法是通过将 DigestJoin 算法中的基于页的取页策略应用到列式存储而来,故此实验选择与 DigestJoin 算法进行比较。实验采用的参数为:页大小为 4kB,数据表大小为 40MB,即 10000 页,连接选择率为 0.05 ~ 0.5,可用内存大小为 2MB,即 500 页。一个闪存页可以存储 50 个属性值,表的属性个数为 50,对于行式存储,总页数为 10000 页,即有 10000 条记录,则转换为列式存储,每一个属性列占用 200 页。实验从缓冲区和选择率两方面对算法性能进行了测试与分析。

3.2　实验结果分析

缓冲区的大小是衡量数据库的查询性能的一个重要的因素,一般情况下,输入缓冲区越大,读入主存中的闪存页就越多,从而可以总体上减少 I/O 开销,查询效率也就越高。由图 3 可知,DigestJoin 算法和 PGIndex 算法总体趋势都在下降,即 I/O 开销在减小,但下降趋势不明显,也就是两种算法受缓冲区的影响较小,但 PGIndex 算法明显优于 DigestJoin 算法,这是因为 PGIndex 算法是基于列式存储的,减少了 DigestJoin 算

法中的构建 Digest 表的过程，从而减少了大量的 I/O 开销。PGIndex 算法虽然在构建页-元组索引表的过程中需要多次读闪存，但其消耗的 I/O 与构建属性-元组索引表之和仍然远远小于 DigestJoin 算法第一阶段的 I/O 开销。

选择率大小对查询连接算法来说至关重要，选择率越高说明所需数据就越多，生成的最终连接表就越大，相应的 I/O 开销就越大，即查询代价就越高，其实验结果如图 4、5 所示。实验结果表明在不同选择率的情况下，PGIndex 算法的查询速度要优于 DigestJoin 算法。其原因仍然在于列式存储并不涉及 Digest 表的构建，从而减少了大量的 I/O 开销。如图 4 所示选择率较小时，两种算法的查询速度受影响较小，但当选择率较大时，两种算法的查询速度明显下降，如图 5 所示，这是因为选择率较大时，产生的数据较大，需要大量的 I/O 处理。

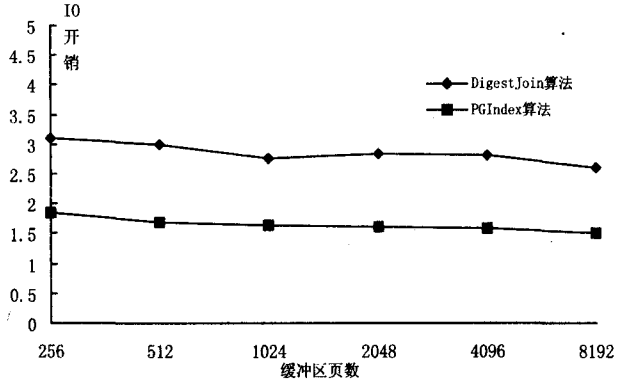

图 3 不同选择率条件下 I/O 开销曲线

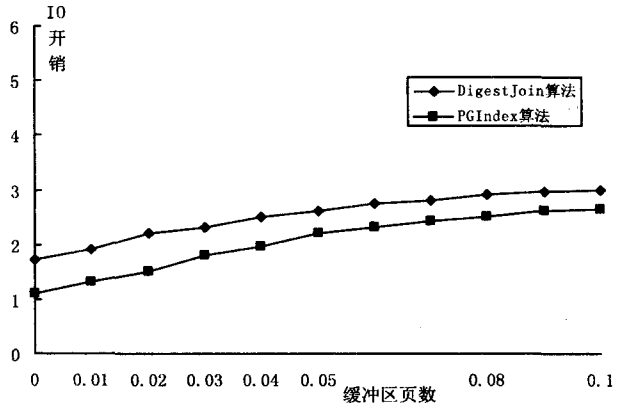

图 4 不同缓冲区条件下 I/O 开销曲线 1

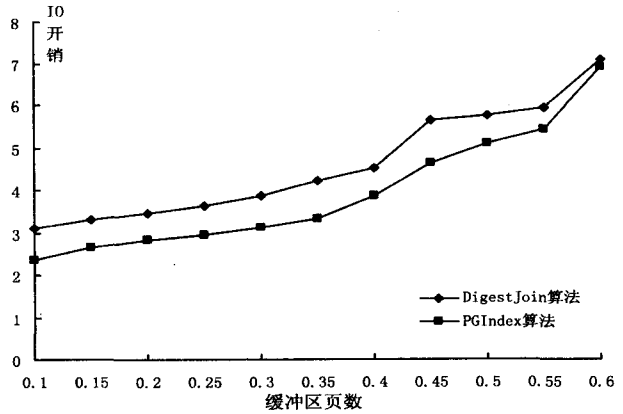

图 5 不同缓冲区条件下 I/O 开销曲线 2

4 结束语

闪存取代磁盘作为存储介质已成为未来发展趋势，当前，关于磁盘数据库的查询优化技术已相当成熟，而闪存数据库的查询优化技术尚处于起步研究阶段，但把基于磁盘的优化算法直接移植到闪存数据库系统，并不能充分利用闪存的读写特性。

针对此问题，文中提出了一种新的查询连接算法，该算法是基于列式存储的，减少了投影阶段的 I/O 代价，从而达到少写闪存的目的，又通过构建页-元组索引表，从而避免了重复读取闪存页，总体上加快了连接的速度。通过与 DigestJoin 算法的比较分析，充分证明了本算法性能的优越性。

参考文献：

[1] Gray J, Fizgerald B. Flash Disk Opportunity for Server-Applications[J]. ACM Queue, 2008, 6(4): 18-23.

[2] Li Y, On S T, Xu J L, et al. DigestJoin: Exploiting fast random reads for flash-based joins[C]//Proceedings of the 10th International Conference on Mobile Data Management. [s. l.]: IEEE Computer Society Press, 2009: 152-161.

[3] Do J, Patel J M. Join processing for flash SSDs: remembering past lessons[C]//Proc. DaMoN. [s. l.]: [s. n.], 2009: 1-8.

[4] 梁智超，周 大，孟小峰. Sub-Join：面向闪存数据库的查询优化算法[J]. 计算机科学与探索，2010，4(5)：401-409.

[5] Shah M A, Harizopoulos S, Wiener J L, et al. Fast scans and joins using flash drives[C]//DaMoN08. [s. l.]: [s. n.], 2008: 17-24.

[6] Li Z, Ross K A. Fast joins using join indices[J]. The VLDB Journal, 1999(8): 1-24.

[7] Merrett T H, Kambayashi Y, Yasuura H. Scheduling of page-fetches in join operations[C]//VLDB'81. [s. l.]: [s. n.], 1981: 488-498.

[8] 李建中，张冬冬，张艳秋. 基于三级存储器的 Join 算法[J]. 软件学报，2003，14(5)：947-956.

[9] 蒋旭东，周立柱. 数据仓库处理中的一种多表连接算法[J]. 软件学报，2001，12(2)：190-195.

[10] 李 静，孙 莉，王 梅. 列存储数据查询中的连接策略选择方法[J]. 计算机科学与探索，2010，4(9)：850-858.

[11] 孙文隽，李建中. 排序合并 Join 算法的新结果[J]. 软件学报，1999，10(3)：264-269.

[12] 苏金泉，苏厚勤. 一种关系式 JOIN 算法的研究与实践[J]. 计算机应用与软件，2007，24(6)：145-146.