A Column-Oriented Storage Query Optimization for Flash-based Database

Jiulei Jiang[1,2], Jiajin Le[1], and Yan Wang[1]

[1)] School of Computer Science and Technology, Donghua University, Shanghai, China (jiangjl@mail.dhu.edu.cn)
[2)] School of Computer, Beifang Ethnic University, Ningxia,Yinchuan, China (wjjl2010@gmail.com)

***Abstract*—The existing database systems are designed based on the physical characteristics of the magnetic disks, and they have failed to obviously improve the performance of solid-state disks (SSD). Therefore, a new connection algorithm, namely CSub-Join, is proposed in this paper. It is a query optimization algorithm for flash databases based on a column storage model. The algorithm first reads columns from two joining tables as CSub tables, and then executes joint operations on CSub tables to generate a join index table and a fetch sequence table. Finally, according to the two generated tables, the query results are taken back from the two joining tables by columns. Using the small size of column oriented storage and the high-speed random read of SSD, the algorithm overcomes the speed mismatch between database systems and flash memories. The results of experiments show that CSub-Join algorithm clearly outperforms CSub-Join algorithm under various selectivities and memory sizes.**

***Keywords*— flash memory, database, column oriented storage, CSub-Join**

基于列存储的闪存数据库查询优化

姜久雷[1,2]　乐嘉锦[1]　王艳[1]

[1)] 东华大学计算机科学与技术学院，上海，中国
[2)] 北方民族大学计算机科学与工程学院，银川，宁夏，中国

摘　要　现有数据库系统都是基于磁盘的物理特性设计，其在固态硬盘的性能未能获得明显的提升，为此提出了一种新的基于列存储模式的闪存数据库查询优化连接算法 CSub-Join。该算法首先读取连接列的文件作为 CSub 表，然后对 CSub 表进行连接操作，生成连接索引表，最后根据连接索引表和数据载入顺序表从原始数据表中按列回取查询结果。CSub-Join 算法充分利用列存储的存储粒度小和固态硬盘高速随机读的优点，有效克服了数据库系统与闪存之间的速度不匹配问题。实验结果表明 CSub-Join 算法在不同选择率和内存下均比 Sub-Join 算法具有明显的性能提升。

关键词　闪存，数据库，存储，CSub-Join

1．引言

自从 1956 年硬盘的出现，磁盘一直都被用作大量数据的存储介质。随着计算机的普及应用和网络技术的发展，数据的存储和查询需求也急剧增长。磁盘在过去的几十年里尽管从存储容量到 I/O 能力均有了很大的提升，但磁盘的机械特性决定了它的I/O速度很难继续提升，从而与CPU和总线的速度越加不匹配。成为大型数据库和服务器 I/O 性能的重要瓶颈。1988 年出现了称作 flash 的新型的存储介质，由于它具有读写速度快、抗震能力强、体积小、存储密度大、噪声小、耐高温等优异特性，其应用开始广泛而迅速地渗透到各种小型数据存储领域。目前适用于大量数据存储介质的是 NAND 型 Flash—固态硬盘（Solid State Disk,SSD），它采用闪存芯片作为数据存储设备[1]。固态硬盘是一种纯电子存储设备，通过电子电路来存取数据，不存在机械移动带来的延迟，所以固态硬盘具有很高的随机读取速度。由于固态硬盘提供和磁盘相同的访问接口，故可以直接应用于现有的数据库系统。由于其高速的 I/O 性能和迅速增加的容量，闪存被普遍认为是取代磁盘成为新一代数据存储设备的理想选择。

虽然固态硬盘从存储介质特性上解决了 I/O 性能瓶颈，但是如果将目前的数据库直接应用在闪存上，现有的

国家科技支撑计划项目支持（资助号：2007BAD33B03）

数据库性能却未能获得相应幅度的提升。由于现有的数据库系统都是基于磁盘的物理特性而设计的，它的机械式读写特点决定了自身 I/O 性能的局限。而磁盘和固态硬盘在物理特性上的巨大差别使得现有的数据库不能很好地利用固态硬盘的高速读性能。

对应于传统的以记录或行数据为单位进行存储的行存储数据库，列存储数据库是以关系数据库中的属性或列为单位进行存储,数据表记录中的同一属性值存储在一起,而一条记录中不同的属性值则分别存储于不同的文件中。与行数据库相比列数据库具有以下优势[2]:①它的存储粒度小,能够根据需要从磁盘中仅读取需要的属性,从而节省 I/O 带宽；②同一个文件中存储的是相同属性的数据，由于具有相同的数据类型,相邻数据之间的相似性很大,从而有利于数据压缩；③由于数据都以列的形式存储, 在 SQL 语句执行过程中, 节省了行数据库中映射运算的开销；④对于列存储数据库轻量级的压缩态数据, 便于进行直接操作而无需解压处理。列数据库主要适用于数据挖掘、决策支持和地理信息系统等分析型和查询密集型系统中。

2．闪存技术介绍

目前各种 Flash 中最适合用作大量数据存储介质的是 NAND 型 Flash。和磁盘类似，NAND Flash 读写数据的基本粒度为页（page），但是读写数据所需的时间并不一样，一般写入时间是读出的 3 到 10 倍。此外，Flash 不允许数据的直接覆盖写，必须首先擦除原有数据才能写入新的数据，而擦除的粒度为块（block），擦除的速度是很慢的，而且每一块的擦除次数也有限，平均为 100 万次。一个具体的 NAND Flash 组织结构图如图 1 所示。

图 1 是三星 K9K8G08U0M 固态硬盘的内部组织结构[3]。从图中可以看出，整个 Flash 的容量为 8448Mbit，其中含有 8192 个物理块，每块包含 64 个页，每页除了 2K 的数据区之外，还有 64 字节的额外空间用于存放错误校验码等信息。除了 Flash 的主体，我们还可以看到一个一页大小的数据寄存器。每一次 Flash 的读写操作都需要使用这个寄存器来完成。例如，读操作实际上分成两个阶段，首先是

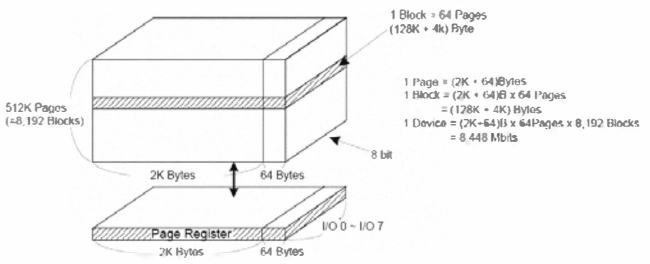

图 1　NAND Flash 组织结构图

把指定地址中的整页数据载入到寄存器中（时间为 20 微秒），然后再从寄存器输出数据，可以连续输出，也可以根据指定偏移量随机输出，且随机输出的次数不限（每个字节的输出周期为 20 纳秒）；而写操作也分为两个阶段，第一阶段是把数据从外界输入到寄存器，可以整页输入，也可以随机输入，且随机输入的次数不限，每字节的输入周期也是 20 纳秒，第二阶段便是把寄存器中的数据固化到 Flash 上，时间为 200 微秒，每一页原则上只能有一次固化操作，然而大部分 NAND Flash 允许在同一页中的几个片断按照先后顺序分几次写入，本例中为 4 次，这种情况叫做 partial page programming。

另外，最新推出的 Sun Storage F5100 闪存阵列[4]在单个机架单元（1.75 英寸）内具有高达 2TB 的固态闪存容量以及高达 12.8GB/s I/O 带宽性能，却仅消耗 300W 的功率。由于其闪存存储系统配备了优化软件，经相关测试，现有 Oracle 和 MySQL 数据库在基于 Sun Storage F5100 的工作处理速度得到了从 65%到高达 2 倍的改进，显示出在企业级应用和高性能计算领域的竞争优势。

3．相关工作

目前的闪存数据库研究主要是针对运行在以闪存芯片为存储介质的固态硬盘上的数据库，所以闪存数据库的查询优化研究工作需要研究如何充分利用固态硬盘的物理特性来提高查询的速度。已有的研究工作都致力于在查询的过程中充分利用固态硬盘的随机读性能，尽量减少对数据的写操作。不同的研究工作采取不同的处理方法。

Mehul A. Shah 提出了一种基于 PAX Layout 方法来提高连接的速度[5]。采用 minipage 列存储的方式，开始查询时只需要读取参与连接操作的属性列，从而有效地减少了数据的读入量。虽然在连接过程中需要比按行存储执行更多的随机读取操作，但是由于固态硬盘的高速随机读特性，能够有效地提高 PAX Layout 方法在连接过程中对表的扫描速度。Mehul A. Shah 提出一种 RARE-join 的连接算法。RARE-join 首先根据连接列构建连接索引，然后根据连接索引的结果去原表中读取只与结果相关的列和行中的数据，从而有效地减少数据的读取量。RARE-join 充分利用了固态硬盘的随机读取性能提高了查询的性能。

Daniel Myers 提出采用 subset[6]的方法来提高查询中的外排序性能。Daniel Myers 只将外排序的中间结果的连接列的数据写入到存储设备，从而有效地减少数据的写操作，在排序结束后再从原始表中获取其它数据，完成最初的排序过程。Daniel Myers 充分利用了固态硬盘的高速读性能和减少其低下的写性能来提高外连接的效率。

在 Daniel Myers 的基础上，Yu Li 提出针对固态硬盘

特点而优化的 DigestJoin 查询优化算法[7]。DigestJoin 首先从查询连接表中抽取出连接列的 Digest 表，然后对 Digest 表进行嵌套循环连接。再应用基于查询图(Join Graph)的方法读入连接查询中位于原始表的数据，从而有效地减少数据页的重复读的次数。此外，DigestJoin 采取页载入(Page Loading)方式来读入原始表的数据，在读取数据时同时将该页的其它数据也读入内存。当随后就需要这些数据的话，就可以减少读入次数，从而有效地提高查询的效率。

国内提出了一种子连接查询优化算法 Sub-Join[8]，算法通过对数据表的连接列进行投影建立子表，并在子表上进行连接查询。最后根据子表连接结果去原始数据表中获得其他的查询数据。通过减少数据的读取，并将数据的随机写转化为连续的写,来提高连接的速度。

以上研究主要是行存储模式的 RDBMS 在闪存下的连接算法，PAX Layout 方法属于页内列存储模式，是对行存储的局部改良，兼具二者的优点。但该算法不适用于海量数据存储环境，而目前在数据仓库环境下的列存储模式下的连接算法的探索还鲜有报道。本文从固态硬盘的物理特性出发，提出了一种新的面向列存储数据库的查询连接算法：CSub-Join，该算法是基于列存储模式下闪存数据库优化，可以提高数据库系统和固态硬盘之间的适应性，发挥闪存的高速读性能。

4．CSub-Join 连接算法

4.1 CSub-Join 算法概述

对于表 X= {$Attr_{x_1}$,$Attr_{x_2}$, . . . ,$Attr_{x_m}$} and Y = {$Attr_{y_1}$,$Attr_{y_2}$, . . . ,$Attr_{y_n}$},分别以 tid_x,tid_y 来表示 X,Y 的元组 ID 号。本文所讨论的是表 X 和表 Y 的连接，即对于 $Attr_{x_i}$=$Attr_{y_i}$ 的元组进行连接，组合为新的表 Z。

在行存储模式下，算法首先计算得到仅包含连接属性和元组 ID 号的投影表（在文献[8]中命名为 Sub 表），相应的投影表是 X'= {$Attr_{x_i}$, tid_x}与 Y'= {$Attr_{y_i}$, tid_y},然后再应用常用的连接算法生成连接索引表 T，形式为 T={$Attr_{x_i}$,tid_x,tid_y}，如果连接结果大于内存则将会顺序写入闪存。算法流程如图 2 所示[6]。该算法开始查询时只对参与连接的属性列进行操作，有效地减少了数据的读入量和中间结果，而在连接的数据回取过程中则执行了更多的随机读取操作，这正是利用了固态硬盘的优势。

本文提出的 CSub-Join 算法则是基于列存储模式下闪存数据库的优化。在列存储数据库中，由于表 X,Y 的数据本来就按属性列分别存储，参与连接的属性列数据被连续存放在一起，所以投影表 X'和 Y'（即 CSub 表）可直接快速读入内存，不用像 Sub-Join 算法那样需要在元组中抽取出属性列数据作为子表（即 Sub 表）。其算法流程如图 3 所示。CSub-Join 算法的执行分为两个阶段：CSub 表连接阶段和数据回取阶段。具体则由 2 个算法（算法 1: CSub Table Join，算法 2: column Data Retrieval）分别实现，其中表 1 说明了算法中涉及到符号及其含义。

4.2 CSub 表连接

CSub-Join 算法的第一阶段是实现连接结果的索引表。算法首先从外盘直接读入连接列文件 CSub1，CSub2（即 T1.Attr1,T2.Attr2），并按属性值进行索引。然后按照先后顺序有序地进行匹配连接，并将结果写入结果表 R 中。这里的每个元组的连接结果仅包含连接属性以及表 T1,T2 的 ID 值对，并不包含原表中其它的列。其具体算法如算法 1 代码所示。

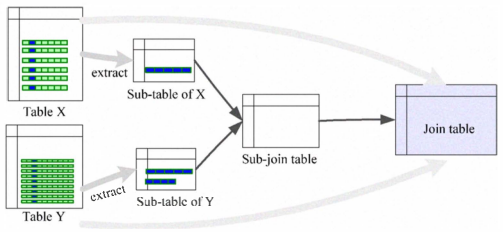

图 2　NSM 模式下的 Sub-Join 连接算法流程

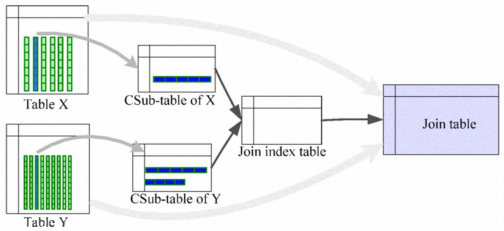

图 3　DSM 模式下的 CSub-Join 连接算法流程

表 1 算法中涉及的符号及含义

符号	含义
T1, T2	原表 T1, T2
R	连接索引表
CSub1, CSub2	T1,T2 的 CSub 表
Attr1, Attr2	T1, T2 表中的连接属性
Id1,Id2	表 T1,T2 的 ID 号
JR	连接结果表
ft1,ft2	数据读取顺序表
jct1,jct2	连接备用表

Algorithm 1: CSub Table Join

```
Public Table CSubTableJoin(Table CSub1, Table CSub2){
    Create Index for CSub1, CSub2;
    For each Attr1 in CSub1 and For each Attr2 in CSub2{
        If(Attr2= Attr1){
            Insert <Attr1, Id1,Id2> into R;
            Get next Attr1,Attr2;
        }else{
            If(Attr2< Attr1)
                Get next Attr2;
            Else Get next Attr1;
        }
    }
    Return R;
}
```

Algorithm 2: column Data Retrieval

```
Public Table ColumnDataRetrieval(Table R, T1, T2){
    Create JR,ft1,ft2,jct1,jct2;
    while scanning R do   output Id1,Id2 to ft1,ft2;
    Sort ft1 and ft2 based on id;
    For each column in T1,T2 {
        Fetch attribute values according to ids of ft1,ft2;
        output attribute values to jct1 and jct2;
    }
    For each <Id1,Id2> in R {
        Get all columns of a tuple from jct1 and jct2;
        Output the tuple to JR;
    }
    Return JR;
}
```

该算法中应用索引技术，仅需要进行一趟查找操作，使连接效率得到大幅的提高，其时间复杂度由原嵌套连接算法的 $O(n^2)$降低为 O(n)。而且数据采用列式存储，只需读入所需连接列，明显减少了 I/O 带宽。

4.3 基于列文件的数据回取

算法 1 得到的连接结果{ $Attr_{x_i}$,tid_x,tid_y}只告诉了我们满足连接的元组，为了输出完整的连接结果，就必须从原表根据 ID 号读取相应的元组中其它列的数据。从原表读取元组在 RDBMS 中一般以页为粒度进行的，这样就有了第二个阶段：页读取。在页读取过程中采用的提取策略是确保在回取数据时每一页至多访问一次。为此就要在读取一页中数据时就把该页中用到的数据一次都抽取出来。

算法首先扫描连接索引表，取出表中的 ID 号 Id1,Id2，并保存到数据读取顺序表 ft1,ft2，ft1,ft2 表最后按 ID 完成排序，这样在数据回取过程中按 ID 号逐页有序读取数据，就避免了页的重复读取，从而减少了页的读取数量。然后对原表的每一个列文件按照读取顺序表 ft1,ft2 中 ID 的顺序分别进行数据读取，读取的数据添加到连接备用表 jct1,jct2；最后对表 jct1,jct2 进行连接。按照连接索引表 R 的<Id1,Id2>值对，顺序读出 jct1,jct2 中相应的各属性值，合并为完整的元组后写入连接结果表 JR。具体方法如算法 2 代码所示。

5．实验结果与分析

本节中，我们将 CSub-Join 和行存储数据库下的连接算法 Sub-Join 进行对比实验，在 SSD 存储介质下对算法的性能进行详细的比较。本实验主要评估算法的 I/O 代价以及在随机读方面的性能。

5.1 实验环境

本实验中使用的 PC 配置为：Intel Core 2 Pentium 4 Duo CPU 2.83GHz,2GB 内存。操作系统为 Windows XP sp2，编程语言为 C 语言。数据库采用的是开源的 Oracle Berkeley DB。选择开源数据库的原因是我们能够非常方便地将参加测试的连接算法在其中实现，便于在同一环境下对连接算法进行比较，从而获得有说服力的结果。CSub-Join 和 Sub-Join 算法的算法思想比较相似，主要是不同存储模式下的适应性设计。

实验中使用的固态硬盘为三星 K9K8G08U0M，容量为 8GB。一个块包含 64 个页，每个页为 2KB。在实验中我们分别测试了选择率和内存大小对本算法性能的影响。

5.2 实验结果分析

由于在读取连接列阶段基于列存储的 CSub-Join 明显要快于 Sub-Join，而在回取数据阶段则 CSub-Join 由于随机读次数较多而效率有所下降。但当选择率较低时影响较小。

a) 选择率　选择率的大小对查询连接算法来说至关重要。选择率越高说明产生的数据越多，查询代价就越高，其实验结果如图4所示。实验表明本方法在各种不同选择率

的情况下比Sub-Join方法均要快。其原因在于本方法通过对连接列的直接读取而取代了行存储下需要在全部数据中抽取出连接列来进行连接计算，减少了原始数据的读取量，从而提高了性能。当选择率较低时，两种方法的性能均有明显改善，充分反映出SSD的高速随机读特性。

b) 内存大小　缓冲区是影响连接查询性能的重要因数。由图 5 可见，随着缓冲区的增大，两种算法的性能均有提高，说明该方法能够充分利用缓冲区。另外可看出，该方法受缓冲区大小的影响程度较小，即在缓冲区和内存不足的情况下，算法也有较好的表现，在性能上并未出现大幅波动，反映出该算法对缓冲区和内存的依赖性较弱，其原因主要是 SSD 的 I/O 性能的提高，与内存的匹配度显著提高。

6．总结

本文提出了一个新的连接算法 CSub-Join，实现在列存储模式下对闪存数据库连接查询的优化。该算法利用列存储按属性单独存储数据的特点，能够直接载入所需的连接属性列数据而避免载入元组的其它属性，从而节省了 I/O 代价。而根据连接索引表从原始数据表中抽取查询结果的元组数据则进一步降低了数据读取量和 I/O 代价。算法中设置了数据载入顺序表，按 ID 号对将要载入的元组有序随机读取，从而减少了页的载入次数，提高了算法的效率。由于 SSD 比传统的磁盘拥有更快的随机读速度，基于列存储的数据库就获得了更好的性能。实验结果表明 CSub-Join 算法在 SSD 上的性能优于传统的算法 Sub-Join。

在后续的研究工作中，我们计划进一步研究在不同投影率，以及多元连接条件下的优化算法，以及列存储下压缩态数据的连接优化问题。

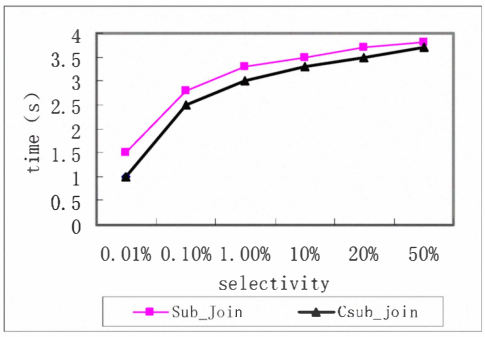

图 4　算法在不同选择率下的性能比较

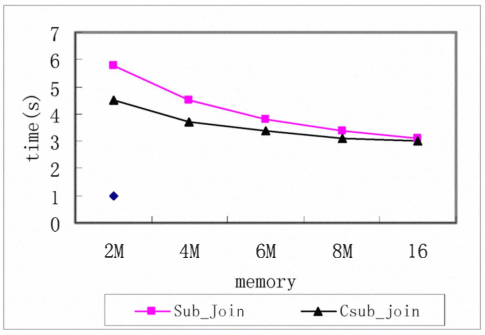

图 5　算法在不同内存下的性能比较

参考文献

[1] Mtron. Solid state drive msd-sata 3035 product specification [EB/OL]. [2009-7-19]. http://mtron.net/Upload_Data/Spec/ASiC/MOBI/SATA/MSD-SATA3035_rev0.4.pdf, 2008.

[2] D. Abadi, et al., "Column-Stores vs. Row-Stores: How different are they really?," in Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, New York, ACM, 2008 pp. 967-980.

[3] SAMSUNG. K9K8G08U0M (1Gx8Bit/2Gx8Bit/4Gx8Bit NAND Flash Memory) [EB/OL]. [2005-11-01]. http://www.alldatasheet.com/datasheet-pdf/pdf/104336/SAMSUNG/K9K8G08U0M.html

[4] ORACLE Datasheet. Sun Storage F5100 Flash Array[EB/OL] http://www.oracle.com/us/products/servers-storage/storage/disk-storage/043967.html

[5] M. Shah, et al., "Fast scans and joins using flash drives," in Proceedings of 4th Workshop on Data Management on New Hardware, New York,ACM,2008, pp. 17-24.

[6] D. Myers, "On the use of NAND flash memory in high-performance relational databases," [D]. Citeseer University, 2007.

[7] Y. Li, et al., "DigestJoin: Exploiting Fast Random Reads for Flash-based Joins," in Proceedings of the 10th International Conference on Mobile Data Management. TaiPei,IEEE Computer Society Press, 2009, pp. 152-161.

[8] Z.C. Liang, D. Zhou and X..F. Meng, “Sub-Join：Query Optimization Algorithm for Flash-Based Database”. Journal of Frontiers of Computer Science and Technology, 2010,4(5):401-409.