

---

# Sub-Join: 一种闪存数据库的查询优化算法

梁智超, 周大, 孟小峰

中国人民大学 信息学院, 北京 100872

## Sub-Join: A Query Optimization Algorithm for Flash-based Database\*

LIANG Zhi-chao, ZHOU Da, MENG Xiao-feng<sup>+</sup>

Information School, Renmin University of China, Beijing 100872, China

+ Corresponding author: E-mail: xfmeng@ruc.edu.cn

**Abstract:** Compared with Hard Drive Disk (HDD), SSD has a lot of advantages, such as high random read performance, low power consumption and lightweight form. Therefore it is envisioned to be next generation data storage instead of HDD. However, the enhancement of query performance for flash-based database is not the same as the IO ratio of SSD to HDD. The reason is existing databases which are designed for HDD can not take full advantage of high IO performance of SSD. In this paper, a new join algorithm, Sub-Join, is proposed. Sub-Join first projects the column of join and primary key as Sub-Table, and then executes join operations on Sub-Tables. Finally results are gotten from original table according to the result of join on Sub-Tables. The compared experiments with Oracle Berkeley DB show Sub-Join outperforms original indexed nested-loop join at the ratio of about 40%~100%. The result strongly shows the high efficiency of this method.

**Key words:** Solid State Disk; flash memory; Flash-based Database; Query Optimization; Sub Join

**摘要:** 和磁盘(HDD)相比, 固态硬盘具有高速的随机读取速度、低功耗、体积小等特点, 因而被认为将取代磁盘成为新一代的数据存储设备。但是闪存数据库的查询性能的提高却远小于固态硬盘相比于磁盘IO性能的提高。其原因在于现有的数据库是基于磁盘而设计的, 使得现有的数据库不能充分发挥固态硬盘的高速性能。因此文章提出一种名为子连接(Sub-Join)的连接算法。子连接算法首先将数据表的连接列和主键投影为新的子表, 然后对子表进行连接操作, 最后根据子表的连接结果再从原始数据表中回取查询结果。通过和开源数据库Oracle Berkeley DB的比较实验, 结果表明子连接算法比原有的算法性能提高40%~100%, 充分说明它的优越性。

**关键词:** 固态硬盘; 闪存; 闪存数据库; 查询优化; 子连接

**文献标识码:** A      **中图分类号:** TP311

## 1 引言

在过去的几十年里, 磁盘一直是最常用的数据存储介质。然而, 随着计算机及网络技术的飞速发展, 数据呈现出爆炸性增长, 数据库系统数据访问能力的需求也远超出磁盘的IO能

\* the Natural Science Foundation of China under grant number 60833005, 60573091(国家自然科学基金项目); National High-Tech Research and Development Plan of China (No:2007AA01Z155, 2009AA011904)(国家863计划); The Ph.D. Programs Foundation of Ministry of Education of China No.200800020002(教育部博士点基金项目)

力。众所周知, 磁盘需要移动磁头来访问数据, 这一过程中存在磁头的机械运动。由于磁盘的机械寻道特性, 磁盘的I/O速度很难继续提高, 导致低速的磁盘与高速的CPU、总线之间的不协调问题已越来越严重。在过去的20年间, CPU处理速度增加570倍, 而磁盘的速度却只增加了20倍<sup>[1]</sup>。可见, 低速的磁盘已经成为大型数据库IO性能的瓶颈。

在这种背景下, 闪存(Flash Memory)技术应运而生。目前广泛使用的固态硬盘(Solid State Disk, SSD)就是采用闪存芯片作为数据存储设备<sup>[2]</sup>。固态硬盘提供和磁盘相同的访问接口, 因此现有的数据库系统无需修改即可移植到固态硬盘上。和磁盘不同的是, 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。因为在读取数据时, 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。如表1所示, 固态硬盘的随机读取速度是磁盘的大约20倍。这也是固态硬盘相比于磁盘最具优势的物理特性。除此之外, 固态硬盘还具有耗电量低, 抗震性强, 体积小等优点。这些优点使得固态硬盘被广泛的认为即将取代磁盘而成为新一代的数据存储设备, 从而解决现有的大型数据库数据访问IO性能不足的现状。

**Table 1** Access throughput and Join performance comparison between HDD and SSD

**表 1** 磁盘和固态硬盘读带宽及连接性能比较表

		Hard Disk Driver(HDD)	Solid State Driver(SSD)	Performance ratio
Read throughput(MB) 1	Random read	3.80	70.76	18.62
	Sequential read	37.45	73.08	1.95
Write throughput(MB)	Random write	3.73	3.71	0.99
	Random write	37.69	68.52	1.82
Time of Join for Oracle Berkeley DB (s) 2		2.71	1.96	1.38

虽然固态硬盘从存储介质特性上解决了IO性能瓶颈, 但是现有的数据库性能却未能获得相应幅度的提升。如表1所示, 相比于磁盘, 固态硬盘在随机读和连续读上提高了大约20倍和2倍, 而其上的数据库在连接查询时只提高了不到40%。其原因在于随机读操作密集型的连接查询中, 现有的数据库需要写入大量的中间结果, 这使得连接查询在不擅长写操作的固态硬盘上无法取得更好的性能。同时现有数据库系统为了提高性能, 尽量避免磁盘上代价高昂的随机读操作。而这恰恰是固态硬盘的优势所在, 所以现有的数据库系统并没有充分利用固态硬盘的高速随机读取速度。总之, 现有的数据库系统都是基于磁盘的物理特性而设计的, 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的利用固态硬盘的高速读性能。

针对现有数据库系统和固态硬盘之间不相适应性, 本文从固态硬盘的物理特性出发, 设计了一种全新的查询连接算法: 子连接(Sub-Join)。子连接算法首先将进行连接查询的相关数据表在主键和连接列上进行投影, 形成子表, 然后在子表上进行连接操作。最终的查询结果是根据子表连接查询的结果再从原始表中回取获得。总的来说, 本文提出的子连接查询优化算法的主要贡献如下:

- 有效地减少了数据的读取。现有的数据库读取数据表的数据, 进行匹配以得到连接结果。而本方法仅从子表中读取数据进行匹配, 从而大量减少了需要读取的数据量。
- 有效地减少了连接过程中的随机写操作。本方法不仅在生成子表时采取连续写, 而且在写出子表连接结果时也采取顺序写的方式, 从而减少了固态硬盘因为随机写而带来的性

1 测试工具为IOMeter 2008, 测试硬盘空间大小为1GB, 测试时间为2分钟, 数据访问粒度为32KB。

2 测试数据库为Oracle Berkeley DB, 索引为B+tree, 记录大小为4K, 两连接表的大小分别为10000和2000行, 连接算法为索引嵌套循环连接。

能降低。

- 高效地利用了固态硬盘的高速随机读性能。从子表的连接结果去获取最终的查询结果, 需要到原始数据表中去读取数据。这个过程需要大量的随机读取操作, 也正是本方法最大的特点所在。这个过程是现有数据库所尽量避免的, 但是对于闪存数据库来说却充分利用了固态硬盘的高速随机读取速度。

本文的组织结构如下: 第 2 节分析了现有连接算法和固态硬盘之间的不相适应性; 第 3 节介绍相关的工作; 第 4 节具体介绍子连接的相关算法; 第 5 节用实验证明了本方法的优越性; 最后第 6 节进行了总结。

## 2 现有连接算法性能分析

现有的闪存数据库采取的查询连接方法主要有嵌套循环连接和索引嵌套循环连接。本节将分析现有数据库连接算法与固态硬盘之间的不相适应性。

### 2.1 数据读取代价大

对于磁盘来说, 读取一个元组中的一个列和整个元组代价基本一致。此外如果只读取其中一个列的话, 磁盘如果需要读取下一条元组, 需要进行一次随机读。而读取整个元组的话, 在读下一条元组时只需要进行连续读, 所以读取整个元组对磁盘来说代价更小。但是这种方法对于固态硬盘来说并不适合。对于固态硬盘, 如表 1 所示, 随机读和连续读具有基本一致的读取速度。那么读取元组中一个列的代价要小于读取整个元组的代价, 其比例相当于该列占整个元组的比例的大小。因此固态硬盘适合采取按列读取所需数据的策略。如果采取按列读取数据的话, 固态硬盘在读取下一条元组时虽然需要进行一次随机读, 但是由于高速的随机读性能, 其代价和连续读取下一个元组中的该列基本一致。所以按列读取适合于固态硬盘的物理特性。

### 2.2 数据写入代价大

对于嵌套循环连接来说, 在内存不足的情况下, 需要将连接的结果不停地写入到磁盘。这个过程中, 需要对磁盘进行大量的随机写。虽然磁盘的随机写性能较差, 但是和随机读相比, 两者基本一致。此外读入的数据为进行连接操作的整个元组, 而写出的为需要查询的列, 所以写出的数据小于读入的数据。在以读为主的查询连接中, 数据写的代价不是整个连接中的主要部分。但是对于固态硬盘来说, 由于具有高速的随机读性能。根据表 1, 其速度是随机写的 20 倍左右。此外, 由于采取只读查询列的方法, 读取的数据就明显减少, 进一步减少了数据读取的代价。在这种情况下, 随机写的代价就在整个查询连接中占有很大的比例, 成为提高连接性能的瓶颈。

## 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提出一种按列存储(PAX Layout)<sup>[3]</sup>的方式来提高连接的速度。传统的数据库主要采用行存储模式, 即数据表根据元组按照行的方式进行存储。采用按列存储的方式, 查询时只需要扫描进行连接操作的列, 从而有效地减少了读入的数据量。虽然这个过程中需要比按行存储执行更多的随机读取操作, 但是由于固态硬盘具有高速的随机读取速度, 所以

PAX Layout方法能够有效地提高在连接过程中对表的扫描速度。Mehul A. Shah同样提出一种RARE-join (Random Read Efficient Join) 的连接算法。RARE-join首先对连接列构建索引, 然后根据索引连接的结果去原表中读取数据, 其中只与结果相关的列和行才被读取出来, 从而有效地减少读取的数据。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<sup>[4, 5]</sup>的方法来提高查询中的外排序性能。传统的外排序算法将元组读入内存后进行排序, 然后将排序后的所有元组的所有列全部写入存储设备。这个过程中非连接列的数据也会被多次写入到存储设备中。Daniel Myers针对固态硬盘的特性在外排序的中间结果写出上进行优化, 只将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。在从原始表中获取其它数据时, 需要进行大量的随机读, 这刚好是固态硬盘的最大优点之一。Daniel Myers充分利用了固态硬盘的高速读性能和避免其低下的写性能, 从而提高了外连接的效率。

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<sup>[6]</sup>。DigestJoin首先将两个进行查询连接的表进行抽取, 获取连接列的Digest表, 然后对Digest表进行嵌套循环连接。在得到连接结果后, 采取基于查询图(Join Graph)的方法来读入连接查询中位于原始表的数据。基于查询图的方法能够有效地减少数据页的重复读, 从而减少读的次数。此外, DigestJoin采取页载入(Page Loading)方式来读入原始表的数据。页载入的方法在读取数据时同时将位于同一页的其它数据也读入内存。如果接下来刚好需要这些数据的话, 就可以减少读入次数, 从而有效地提高查询的效率。

此外也有很多工作有很大的借鉴意义, 比如对SSD数据库性能测试方面的文章<sup>[7]</sup>及索引方面的文章比如Intel FTL<sup>[8]</sup>、NFTL<sup>[9]</sup>、BFTL<sup>[10]</sup>等。Lee 等充分揭示了现有的数据库系统在固态硬盘上所获得的性能。FTL主要作用是将闪存模拟成磁盘, 向上层文件系统提供按块读写的接口。NFTL则提出了一种粗粒度的对应关系, 即一个逻辑地址对应一个物理的块。BFTL通过组提交来提高写的速度。

## 4 子连接

本节首先整体介绍子连接算法的设计框架, 然后分别介绍子连接的三个模块: 子表构建模块、子表连接模块以及数据回取模块。

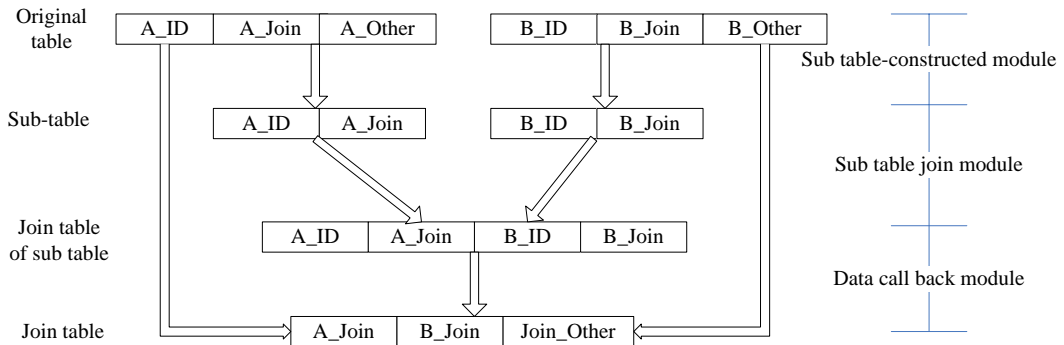


Fig. 1 The Framework of Sub-Join Algorithm

图 1 子连接算法框架图

### 4.1 整体结构

子连接算法的整体结构如图 1 所示。首先从原始表中根据主键和查询列投影得到子表,

称为子表构建模块。其次将子表进行连接得到子表的连接结果表, 称为子表连接模块。最后再将查询结果中需要的其它列的数据从原始表中读取, 得到最终的查询结果, 称为数据回取模块。

## 4.2 子表构建

根据之前的分析, 为了充分利用固态硬盘的物理特性, 在连接操作中只读取连接列可以提高闪存数据库连接操作的性能。基于这一思想, 我们提出采用子表来减少数据的读取代价。

子表就是将进行连接操作的原始数据表按照主键和连接列投影形成的新表。由于新表只包含原始数据表的一部分, 所以称之为子表。子表的抽取过程如表 2 算法 1 所示。首先获得原始表的主键和连接列, 然后对原始表进行投影操作。将投影的结果插入到子表中, 并将子表按照连接列进行排序。

通过建立子表, 我们可以有效地减少数据的读取量。在循环嵌套连接中, 需要读入整个元组, 而在连接过程中需要的仅仅是连接列。通过建立子表, 每个元组只包含原始表的主键和连接列, 这样再读入的数据都是进行连接操作的有效数据。不仅如此, 如果嵌套循环只读取连接列的话, 需要进行随机读取, 而本方法只需要连续的读取子表即可, 从而克服随机读取带来的性能降低。

## 4.3 子表的按列存储

为了进一步减少数据的读取代价, 我们创新性地提出对子表进行按列存储的方式。现有的数据库采取的按行存储在读取数据的列数越多性能越好。当只需要读取数据表中很少一部分列时, 效果较差。

本文中的子表包含连接列和原始表中的主键。我们在进行连续读取时, 不仅将连接列读入了内存, 同时也对原始表的主键也进行了读取, 从而导致读取代价的增加。因此本文采取一种新的数据存储方式, 即按列存储。将子表的连接列和原始表的主键列分开存储。采取按列存储的方式, 我们在进行连接操作时, 读取的数据只包含连接列, 从而减少了数据的读取量, 进而提高了读取的效率。此外由于数据是按列进行存储, 所以连接列的数据都是存储在相邻的位置, 可以对数据进行连续读取, 在一定程度上进一步降低数据读取的代价。由于子表的建立过程中, 我们需要将原始表的数据插入到建立的子表中, 这会导致大量的写操作。为了减少写的代价, 所有的子表数据采取连续写的方式。当缓冲区满的时候将其中的连接列和主键列写入到固态硬盘中。如果固态硬盘中已经存在子表的部分数据, 则将缓冲区的数据和这部分数据进行合并, 然后就可以将合并后的结果按照连续写的方式写入到固态硬盘中。虽然这个过程中会增加数据的读取量和连续写操作, 但是可以避免固态硬盘上代价高昂的随机写操作。所以从总体上减少写的代价。

## 4.4 子表连接

在建立子表之后, 我们对原始表的连接操作采用对子表的连接操作来代替。子表连接操作不仅如前面所提到的可以减少数据的读取量而提高连接的效率。更重要的是, 在给定的内存中, 元组数据量的减少可以提高内存中容纳的元组的个数。随着元组个数的增加, 在连接过程中 IO 的次数就大幅减少, 从而提高连接的效率。子表的连接操作如表 2 算法 2 所示。首先对从外层表中取出 `Key_A`, 再从内层表中读取所有 `Key_B` 等于 `Key_A` 的元组。并记录具有相同 `Key_B` 值的第一个元组的位置 `First_Dup_Key`。如果外层表的下一个值仍然是 `Key_A`, 那么直接从 `First_Dup_Key` 开始读取所有具有相同值的 `Key_B`。重复此过程直到遍

历完外层表所有 key 值。算法二利用 `First_Dup_Key` 巧妙地减少了连接中的数据读取量。对于外层表的一个键值, 算法二将读取所有内层表与之具有相同键值的元组。当外层表的指针向下移动时, 理论上需要再去内存表中得到与之键值相同的所有元组。在算法二中, 首先下一个键值会被判断是否与前一个键值相等, 如果相等, 则利用前一个键值连接的结果, 如果不相等, 则去内层表中读取相关的元组。从而大幅减少对内存表的读取。本算法在外层表重复键值较多时, 因为减少更多对内层表的读取, 所以效率更高。

Table2 Detail algorithms of sub-join

表 2 子连接的具体算法

<i>Algorithm 1: Sub Table Projection</i>	<i>Algorithm 2: Sub Table Join</i>
<pre>Public Table SubTableProjection (Table OriginalTable, String JoinColumn ){     Get primary key of OriginalTable;     Get Column ID of JoinColumn;     Create SubTable(JoinColumn, primary Key);     For each record of OriginalTable{         Get value of JoinColumn, primary key;         Insert into SubTable;}     Return SubTable;}</pre>	<pre>Public Table SubTableJoin (Table SubTable_A, Table SubTable_B){     Create Index for SubTable_A, SubTable_B;     Create SubJoinTable;     int Prev_Key_A, Key_A, First_Dup_Key, Key_B;     For each record in SubTable_A{         If(Key_A!=Prev_Key_A){             Get the first key_B where             Key_B=Key_A;             First_Dup_Key=Key_B;             Get all Key_B;             Insert result into SubJoinTable;         }else{             Key_B= First_Dup_Key;             Get all Key_B;             Insert result into SubJoinTable;}}     Return SubJoinTable; }</pre>
<i>Algorithm 3: Original Data Retrieval</i>	
<pre>Public Table OriginalDataRetrieval (Table STable, OTable_A, OTable_B){     Create JoinTable;     For each record in STable {         If(Key_A!=Prev_Key_A){             Get other column from OTable_A;}         If(Key_B!=Prev_Key_B){             Get other column from OTable_B;}         Insert other column into JoinTable;}     Return JoinTable;}</pre>	

#### 4.5 数据回取

通过子表连接之后, 可以得到连接列的查询结果。而实际的查询过程中, 往往还需要查询其它列的数据, 因此需要到原始表中去进行查询。在我们的设计中, 回原表读取数据并不是对原表进行扫描而获取数据, 而是根据已知的主键来获得需要的数据。根据前面的介绍, 我们的子表中含有原始表中的主键, 因此在子表连接之后, 同理得到了原始表的主键值。众所周知, 根据主键获取数据对数据表具有很高的效率。其过程如表 2 算法 3 所示, 首先读取子连接表的元组, 根据其中的主键回原始表中回取查询的其它数据。

数据回取算法充分利用了固态硬盘的高速随机读取速度, 也是子连接算法的进行查询优化的设计依据之一。利用子表连接结果去原始表中回取查询的其它数据, 这个过程中需要进行大量的随机读操作。如果在磁盘上, 由于随机读需要进行磁头移动而代价高昂, 因此大量的随机读导致此算法在磁盘性能甚至不如已有的算法。但是对于固态硬盘来说, 随机读和连

续读具有几乎一致的速度, 所以回取其它数据相当于连续读取其它数据。由于回取过程中, 只读取子连接表中的结果, 因此取出的数据量较少。所以在固态硬盘上, 由于回取的数据量较少且速度快, 所以回取效率高, 也证明了回取算法充分利用了固态硬盘的高速随机读性能。

### 5 实验结果与分析

在本节中我们将子连接算法和现有数据库的索引嵌套循环连接算法进行了对比实验。本节首先介绍了实验环境, 最后展示了实验结果以及进行解释说明。

#### 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, 编译环境为 VS2005。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的将子连接算法在其中实现, 并和现有的数据库连接算法进行比较, 从而获得有说服力的结果。实验中使用的固态硬盘为 Mtron SATA-3035-016, 容量为 16GB。实验中选择和经典的索引嵌套循环连接算法进行比较。在实验中我们分别测试了选择率, 缓冲区和元组大小对本算法性能的影响。

#### 5.2 实验结果分析

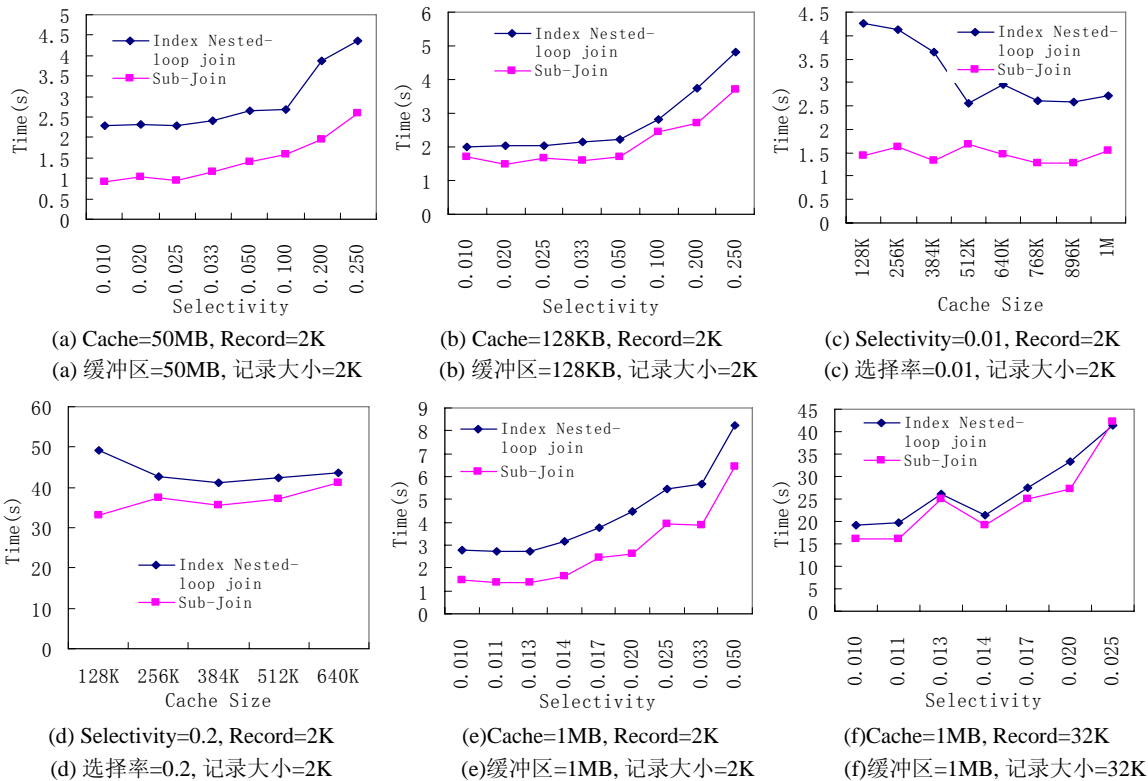


Fig. 2 The comparison performance between Indexed nested-loop Join and Sub-Join

图 2 索引嵌套循环和子连接算法性能比较图

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 2 (a)和(b)所示。实验表明本方法在各种不同选择率的情况下比索引嵌套循环方法均要快。其原因在于本方法通过对子表来进行连接计算, 减少了原始数据的读取量和中间结果的写出, 从而提高了性能。当缓冲区较大时, 比如(a)图, 本方法的效果更加明显, 性能提高达到 100%左右使缓冲区得到了更加充分的利用。

**缓冲区。**缓冲区是影响连接查询性能的重要因数。由图 2 (c)和(d)可见, 本方法不受缓冲区大小的限制, 性能均优于索引嵌套循环连接。对于 2 (c)缓冲区相对较大来说, 我们的方法效率更高, 说明本方法能够充分利用缓冲区。另外索引嵌套循环连接对内存依赖较大, 原因在于其需要缓冲大量的数据。所以在缓冲区增加时性能提高。本方法虽然能够充分利用缓冲区, 但是并不对缓冲区产生很强的依赖, 如图 2(d)所示。在缓冲很小的情况下, 本方法仍然优于现有的方法。这主要得益于我们通过子表进行连接, 而子表相对来说较小, 产生的临时数据也较少。

**元组大小。**元组的大小也是影响连接算法的性能的一个因素。由图 2 (e)和(f)可见, 本方法始终快于索引嵌套循环连接。原因在于本方法能够充分的利用 SSD 的随机读取性能来提高连接的效率。当元组越小, 数据的随机读取越多, 所以本方法的效果也更加明显。由图 2 (e)可见, 在元组大小为 2KB 时, 需要进行更多的随机读取, 所以本方法能够得到更加充分的发挥。即使在元组较大时, 连续读取较多, 本方法因为减少了临时数据的写出量, 从而在性能也还是比现有的方法要好。

## 6 总结

磁盘是目前数据库系统数据存储的主要设备。随着数据呈现爆炸性增长, 应用系统对数据库的数据访问能力提出了更大的需求。然而由于磁盘本身存在的机械延迟成为现有的数据库提高数据访问能力的瓶颈。固态硬盘由于具有高速的访问能力而为解决这一问题提供了契机。然而现有的数据库都是基于磁盘而设计的, 而固态硬盘和磁盘存在物理特性上的巨大差别, 导致现有的数据库并不能充分利用固态硬盘的高速性能。本文总结了现有数据库连接查询和固态硬盘之间的不相适应性, 提出了一种新颖的子连接查询优化算法。子连接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它查询的数据。这个过程中充分减少了数据的读取, 并将数据的随机写转化为连续的写, 从而有效地提高了连接的速度。通过和现有的数据库系统的比较, 充分显示了子连接算法性能的优越性。

## References

- [1] Gray J. Tape is dead, Disk is tape, Flash is disk, RAM locality is king [EB/OL]. 2006[2009-7-19]. [http://research.microsoft.com/en-us/um/people/gray/talks/flash\\_is\\_good.ppt](http://research.microsoft.com/en-us/um/people/gray/talks/flash_is_good.ppt).
- [2] Mtron. Solid state drive msd-sata 3035 product specification [EB/OL].2008[2009-7-19]. [http://mtron.net/Upload\\_Data/Spec/ASiC/MOBI/SATA/MSD-SATA3035\\_rev0.4.pdf](http://mtron.net/Upload_Data/Spec/ASiC/MOBI/SATA/MSD-SATA3035_rev0.4.pdf), 2008.
- [3] Shah M A, Harizopoulos S, Wiener J L, et al. Fast scans and joins using flash drives [C]// Luo Q, Ross K A. Proceedings of 4th Workshop on Data Management on New Hardware. New York: ACM Press, 2008: 17-24.
- [4] MIT Database Group. MIT CSAIL Research Abstracts [EB/OL].2007[2009-7-19]. <http://publications.csail.mit.edu/abstracts/abstracts07////flashdb-abs/flashdb-abs.html>.
- [5] Myers D. On the Use of NAND Flash Memory in High-Performance Relational Databases[D]. Massachusetts:MIT,2008.

- [6] Li Y, On S T, Xu J L, Choi B, et al. DigestJoin: Exploiting Fast Random Reads for Flash-based Joins [C]// Lee W C, King C T, Pitoura E. Proceedings of the 10th International Conference on Mobile Data Management. Taipei: IEEE Computer Society Press, 2009: 152-161.
- [7] Moon B, Park C, Lee S W. A Case for Flash Memory SSD in Enterprise Database Applications [C]// Wang J T. Proceedings of the ACM SIGMOD International Conference on Management of Data. New York: ACM Press, 2008: 1075-1086.
- [8] Intel Corp. Understanding the Flash Translation Layer (FTL) Specification: AP-684 [R]. California: Intel Press, 1998.
- [9] Kim J, Kim J M. A Space-Efficient Flash Translation Layer for Compact-Flash Systems. IEEE Transactions on Consumer Electronics, 2002, 48(2): 366-375.
- [10] Wu C H, Chang L P, Kuo T W. An Efficient B-Tree Layer for Flash-Memory Storage Systems [C]// Chen J, Hong S. Proceedings of the 9th International Conference on Real-Time and Embedded Computing Systems and Applications. LNCS 2968, Tainan: Springer-Verlag, 2003: 199-212.

### 作者简介:



**Liang Zhi-chao** was born in 1986. M.Sc. Students in computer science from Renmin university of China, Beijing, China. His current interests include query processing of Flash-based database systems.

梁智超(1986—), 男, 天津市蓟县人, 硕士研究生, 主要研究方向为闪存数据库查询处理。



**Zhou Da** was born in 1980. Ph.D. candidate in computer science from Renmin university of China, Beijing, China. His current interests include indexing, query processing and transaction processing of Flash-based database systems.

周大(1980—), 男, 湖南省祁东县人, 博士研究生, 主要研究方向为闪存数据库存储, 索引和事务处理。



**Meng Xiao-feng** was born in 1964. He received the Ph.D degree in Computer Application and Technology from Institute of Computing Technology Chinese Academy of Sciences in 1999. Now he is a professor and doctoral supervisor at Renmin University. His research interests include web data management, native XML databases, mobile data management, etc.

孟小峰(1964-), 男, 河北邯郸人, 1999年于中国科学院获计算机应用技术专业工学博士学位, 目前是中国人民大学教授, 博士生导师, 主要研究领域为 Web 数据管理、XML 数据库、移动数据管理。