混合散列连接算法随机 I/O 消除①

刘明超[1], 杨良怀[1], 周为钢[2]

[1](浙江工业大学 计算机科学与技术学院, 杭州 310014)
[2](杭州市公安局 交通警察局科研所, 杭州 310014)

摘 要: 混合散列连接算法(HHJ)是数据库管理系统查询处理中一种重要的连接算法. 本文提出通过缓存优化来减少随机 I/O 的缓存优化混合散列连接算法(OHHJ), 即通过合理优化分区阶段桶缓存的大小来尽量减少分区过程中产生的随机 I/O. 文章通过对分区(桶)大小、桶缓存大小、可用缓存大小、关系表大小与硬盘随机 I/O 访问特性之间的关系进行定量分析, 得出桶大小以及桶缓存大小最优分配的启发式. 实验结果表明 OHHJ 可以较好地减少传统 HHJ 算法分区阶段产生的随机 I/O, 提升了算法性能.

关键词: 混合散列连接; 随机 I/O; 桶缓存; 查询处理

Towards Eliminating Random I/O in Hybrid Hash Joins

LIU Ming-Chao[1], YANG Liang Huai[1], ZHOU Wei-Gang[2]

[1](School of Computer Science and Technology, Zhejiang University of Technology, Hangzhou 310014, China)
[2](Insititute of Traffic Control, Hangzhou Municipal Public Security Bureau Traffic Police Division, Hangzhou 310014, China)

Abstract: HHJ is one of the mostly used core join algorithms for query processing in a database management system. This paper proposes a buffer-optimized hybrid hash join algorithm(OHHJ) by optimizing the bucket buffer to reduce the random I/O in hash join, i.e., to minimize the random I/O by optimizing the bucket buffer size in partition phase. By quantitatively analyzing the relationship between the bucket size, bucket buffer size, available memory size, relation size and random I/O access characteristics of hard disk, we have derived the heuristics for allocating the optimal bucket and bucket buffer sizes. The experimental results demonstrate that OHHJ can effectively reduce random I/O in HHJ during partition phase, and thus enhance the performance of the algorithm.

Key words: hybrid hash join; random I/O; bucket buffer; query processing

1 引言

在数据库系统中连接操作是使用最为频繁的查询操作, 连接算法性能直接影响数据库系统的整体性能, 因而连接算法优化一直是数据库系统研究领域的热点问题之一. Leonard 在文献[1]中提出了一种基于普通散列连接算法和 GRACE 散列算法的混合散列连接算法(HHJ). 作者在文中对数据库中四种常见的连接算法即循环嵌套连接[2]、归并排序连接[3]、简单连接算法、GRACE 散列算法[4]与新提出的 HHJ 算法性能进行比较. 根据以上连接算法的连接过程计算出各个连接算法的连接代价, 通过比较发现在大多数情况下 HHJ 性能要优于其它连接算法. Leonard 在设计 HHJ 的时候从减少数据在硬盘和内存之间传输的角度考虑: 尽量使驻留在内存那个分区较大, 因而其余需要写回硬盘的分区的桶缓存尽量小, 文中桶缓存设定值为一个块(block)大小. 实际上减少数据传输和减少 I/O 代价并不相同, 由于硬盘在随机访问时有磁头的寻道时间和盘片的旋转延迟, 因此在访问数据量相同的情况下随机访问的代价要远高于顺序访问的代价.

本文对页面传输量和随机 I/O 产生代价进行综合考虑, 提出了基于桶缓存优化的缓存优化混合散列连接算法(OHHJ). 为了尽量消除 HHJ 中产生的随机 I/O,

① 基金项目:国家自然科学基金(61070042);浙江省自然科学基金(Y13F020114,Y1090096)
收稿时间:2012-12-10;收到修改稿时间:2013-01-15

OHHJ 桶的缓存不再固定为一个块, 而是根据可用缓存、表的大小以及硬盘随机访问的特性进行调整, 文中也将给出桶大小、桶缓存大小最优分配的启发式.

2 相关工作

众多数据库研究人员对于如何精确地计算出各个连接算法的I/O代价做了大量的研究和工作. Blasgen[3]通过页面访问时间乘以页面访问量得出嵌套循环连接和归并排序连接的 I/O 代价; Selinger[5]采用和前面相同的方式在分布式关系型数据库管理系统中计算连接算法的 I/O 代价. Hagmann[6]用 I/O 请求访问次数作为I/O 代价评估的主要标准, 基于新的 I/O 代价模型对嵌套循环连接算法和散列连接算法的缓存分配方案都重新进行分配, 得出了一些比较有趣的结论: 基于上述代价模型嵌套循环连接算法最佳缓存分配方案是 2 张表的输入缓冲区平均分配. 表明 I/O 代价模型的改变的确会对连接算法最终 I/O 代价的评估有较大影响, 但是并没有在文中证明该代价模型的准确性. Graefe在文献[7]中阐述了散列连接和归并排序连接的相同之处和二者之间的差异, 在比较二者性能的过程中发现: 增加 I/O 访问的单位, 散列连接和归并连接性能都会有显著提升. 当I/O访问单位取32KB时算法能够获取较大的性能提升, 文中并没有提及或者计算最佳的I/O访问单位应该为多大. Haas 等[8]提出了一种更加详细更加真实的 I/O 代价模型, 该代价模型包括了页面传输代价、寻道时间以及I/O请求次数. 利用代价模型分析并最终得出数据库 5 种常用连接算法的I/O代价, 并根据最终计算出的 I/O 代价进一步推导出这 5 种连接算法最佳的缓存分配方案, 最后实验显示根据真实代价模型得出的最佳缓存分配方案要优于以往的缓存分配方案. 但是对于 HHJ, 上文里面给出的最佳缓存分配方案在某些情况下并不合适, 本文正是基于此提出新的HHJ缓存分配方案. 另外 Lo[9]也做了和本文相似的研究: 消除散列连接中的随机 I/O. 文中分区阶段利用批写技术(batch-write), 连接阶段采取读组(read-group)和写组(write-group)策略来减少散列连接算法在分区阶段和连接阶段产生的随机 I/O.

3 缓存优化的混合散列连接算法

HHJ 基于 GRACE 散列连接和简单散列连接算法进行了改进,故HHJ和GRACE同样分为2个阶段: 分区阶段和连接阶段. 它们之间的差异在于分区阶段: GRACE每个分区都需要写回硬盘, 而 HHJ 小表在分区阶段第一个分区驻留在缓存并根据连接键值为第一个分区建立其散列表, 其余的分区均写回硬盘. 大表分区阶段对属于第一个分区的元组会探测小表第一个分区的散列表, 如果键值匹配则和对应小表第一个分区的元组进行连接并生成连接结果, 不匹配的元组则直接丢弃. 大表的其余分区则和小表一样全部写回硬盘的分区文件. HHJ 连接阶段和GRACE完全一致. 根据上面的分区过程, HHJ分区阶段总缓存可分为以下三个部分: 工作区(work space)、桶缓存、输入缓存. 工作区用于保存分区时小表的第一个分区和其对应的散列表, 桶缓存用作分区时其余分区的输出缓存, 输入缓存则用于分区阶段读取小表和大表需要进行分区操作的元组. 原始的HHJ从减小页面访问量角度考虑会使桶缓存尽量小, 一般取操作系统I/O 访问的最小单位(块=4KB). 以往的研究表明在桶缓存较小的情况下, 散列连接算法随机I/O现象严重.

为了尽量减少混合散列连接中产生的随机 I/O, 本文认为应该在缓存允许的情况下尽量增加桶缓存大小. 那么桶缓存到底应该增加至多大合适, 桶缓存和可用缓存以及小表之间的关系是怎样的呢? 下面将进行分析, 结合硬盘随机 I/O 访问特性给出一个桶缓存的最佳分配方案.

设参与连接的小表为 R, 小表大小为$|R|$、缓存的大小为$|M|$(这里不包括输入缓存, 在算法实际执行阶段输入缓存取一个固定大小)、每个散列桶输出缓存大小为 O, 修正因子为 F(一般取值为1.2). 假设分区之后小表每个分区的大小为$|r|$, 那么分区数为:

$$N = |R|/|r| \tag{1}$$

缓存$|M|$由工作区和桶缓存组成, 那么有如下表达式成立:

$$F|r| + (N-1)*O = |M| \tag{2}$$

其中$F|r|$是保留1个分区在内存时所需内存. 代入公式(1)有:

$$F|r|^2 - (O + |M|)r + |R|O = 0 \tag{3}$$

上式看作$|r|$的二元一次方程, 该方程有解的充要条件为:

$$(|M| + O)^2 - 4F|R|O \geq 0 \tag{4}$$

在实际场景中$|M|$远大于 O, 故可以对上式作如下近似处理:

$$M^2 - 4F|R|O \geq 0 \tag{5}$$

即:

$$O \le \frac{M^2}{4F|R|} \tag{6}$$

从消除随机 I/O 的角度考虑, 桶缓存越大越好, 因此桶的缓存的最佳取值应为:

$$O = \frac{M^2}{4F|R|} \tag{7}$$

另外桶缓存的大小至少应该为 1 个块大小, 即 O ≥1, 故有:

$$M^2 > 4F|R| \tag{8}$$

那么是否桶缓存越大越好? 答案是否定的. 因为根据硬盘随机 I/O 访问的特性来看, 桶缓存的大小超过某个值以后, 继续增加其大小, 连接算法性能将不会有比较明显的改善. 另外, 工作区大小是固定的, 桶缓存增加必然会使留在内存中第一个分区变小. 桶缓存增加至某一个值的时候, 再继续增加, 消除的随机 I/O 和(因缓存 1 个更大的分区)减小的页面传输比较起来可能并不占优. 因此应该为缓存的大小预先设定一个阈值, 该阈值大小应和硬盘的随机访问特性相关. 本文该阈值大小取 256KB, 参考希捷硬盘(7200RPM)随机 I/O 访问特性, 该硬盘的随机访问性能见表 1.

表 1 硬盘随机存取性能

随机存取数据大小(KB)	IOPS / 秒	平均速度 (MB/S)
0.5	56	0.027
4	56	0.222
64	54	3.388
1024	40	40.673

实验部分表明桶缓存阈值设定为 256KB 时性能和原始的散列连接算法比较已能获得较大的提升. 因此在所提混合散列连接算法中, 桶缓存的大小通过以下方式设定: 首先由公式(7)计算出一个值, 当计算出来的值小于 64(256/4)页的时候桶缓存的大小为公式计算出来的值, 反之桶缓存大小设定为阈值 64. 在确定桶缓存的大小之后根据公式(3)求得每个分区大小为:

$$|r| = \frac{|M| + O + \sqrt{(|M|+O)^2 - 4F|R|O}}{2F} \tag{9}$$

对缓存优化后的算法称为优化的混合散列连接算法 OHHJ.

4 性能评价

原始混合散列连接算法和基于桶缓存优化的混合散列连接算法用 C++进行实现, 算法没有采取其它的优化手段. 为避免操作系统文件缓存对算法性能影响, 实验中所有数据均是从自己的缓冲区直接写回硬盘. 实验中使用 Intel 酷睿 2 四核 Q8200 处理器, 2.33GHz 主频, 4GB 内存, Windows 7 操作系统, 硬盘希捷(7200RPM). 实验所用数据采用 TPC-H 测试基准生成, 选取 CUSTOMER 表主键和 ORDERS 表外键做等值连接.

表 2 数据集大小

数据规模 / 关系表	10	20	30
CUSTOMER(MB)	257	516	774
ORDERS(MB)	1344	3789	5691

4.1 不同数据规模下算法性能比较

本实验中使用 TPC-H 数据生成程序生成三种规模的数据, 分别为 10、20、30. 3 种不同数据规模下 CUSTOMER 表和 ORDERS 表对应大小见表 2. 实验中原始的混合散列连接使用的缓存分配方案参考文献[1], 新散列连接使用优化的缓存分配方案, 实验 1 使用的缓存大小为 6MB.

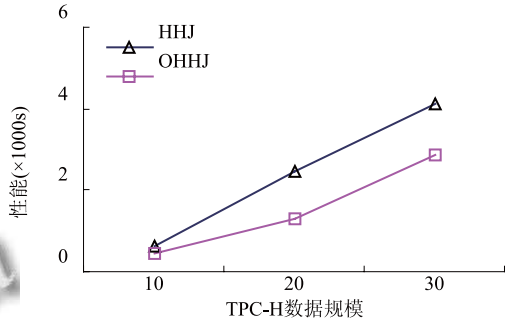

图 1 三种数据规模下两种算法性能(6M 缓存)

实验中, 连接算法总的代价忽略了连接结果的生成和输出, 因为一般认为在相同连接结果集下不同的连接算法在生成连接结果阶段消耗的时间相等. 由图 1 可以发现连接算法都是随着 TPC-H 数据规模的提升连接所消耗的时间随之增加, 另外还可以看到 OHHJ 性能一直要优于原始 HHJ 的性能: 数据规模为 10、20 和 30 时性能分别提升了 25%、48%和 30%. 这是因为 OHHJ 使用了优化之后的缓存分配方案, 较好的消除了分区阶段产生的随机 I/O, 改善了原有连接算法性能. 从下图中还可以观察到数据规模愈大 OHHJ 和原始 HHJ 之间性能差异愈大, 分析是因为数据规模越大

分区阶段生成的分区文件越多, 即原始 HHJ 分区阶段发生随机 I/O 的次数越多. 此随机 I/O 消耗代价占总连接代价较大一部分, 而 OHHJ 能够尽量消除随机 I/O, 因此后者在性能上能获得较大提升.

4.2 缓存对算法性能影响

实验 2 选取 TPC-H 数据规模为 10 和 20 的组数据在 5 种不同的缓存下进行连接操作, 观察缓存的变化对 2 种连接算法性能变化趋势以及在不同缓存下 2 种连接算法性能的比较. 从图 2 和图 3 可以看出, 两种数据规模下做连接随着缓存增加两种算法性能变化趋势有一定差异: 对于原始的 HHJ, 随着缓存增加, 留在缓存第一个分区增加且缓存增加分区数随之变小一定程度上能减少随机 I/O 发生, 因此随着缓存增加原始 HHJ 连接代价整体上呈现下降趋势. 但是会有一些反常现象的点存在, 在数据规模为 10 缓存为 24MB 和数据规模为 20 缓存为 30MB 的情况下性能反而会比前面缓存小的情况下有所下降. 这点和分区时选取的散列函数有关, 实验中分区散列函数是小表连接键值除以分区数取余, 对于上面的异常点分区时由于键值分布的原因小表落在第一个分区元组比较少, 此时原始 HHJ 不能较好达到减少页面传输的效果, 从而导致缓存增加性能反而有所下降.

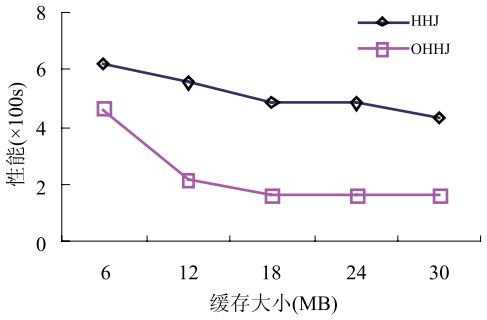

图 2　数据规模为 10 时两种算法性能

OHHJ 随着缓存的增加算法性能会逐步提升, 最后当缓存超过某个范围之后算法性能呈稳定状态. 这是因为由公式(7)可知随着缓存增加在数据规模固定的情况下, 桶的缓存逐渐增加对于消除分区阶段产生的随机 I/O 是有利的, 而当缓存增加至更大的情况下桶的缓存已达到限定阈值, 这个时候增加缓存算法性能不会有明显改善. 另外原始的 HHJ 由于桶缓存大小一直为 1 个块, 故依然存在较多的随机 I/O, OHHJ 在性能上面和原始 HHJ 比较一直处于优势. 在二者性能趋于稳定的状态下, OHHJ 和 HHJ 相比性能提升了 66%(数据规模为 10)和 64%(数据规模为 20).

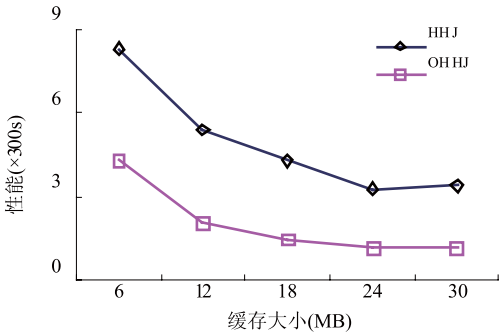

图 3　数据规模为 20 时两种算法性能

5　小结

本文对原始 HHJ 分区阶段进行分析, 针对其缓存分配方案的不足之处提出了基于桶缓存优化的 OHHJ. OHHJ 从消除分区过程中的随机 I/O 入手, 并结合硬盘随机访问的 I/O 特性, 优化了分区阶段的缓存分配方案. 实验显示 OHHJ 比原始 HHJ 比较有较大的性能优势. 未来可以考虑在固态盘和硬盘结合的情况下设计更高效的混合散列连接算法.

参考文献

1 Leonard DS. Join Processing in Database Systems with Large Main Memories. ACM Trans. on Database Syst., 1986: 239–264.

2 Wolf JL, Iyer BR, Pattipati KR, Truek J. Optimal Buffer Partitioning for the Nested Block Join Algorithm. ICDE, 1991: 510–519.

3 Blasgen MW, Eswaran KP. Storage and access in relational databases. IBM Systems Journal,16(4):363–377.

4 Massaru K, Tanaka H, Tohru M. Application of hash to data base machine and its architecture. New Generation Computing-NGC, 1983,1(1):63–74.

5 Selinger P. Access Path Selection in a Relational DataBase Management System. SIGMOD, 1979: 23–24.

6 Hagmann R. An Observation on Database Buffering Performance Metrics. VLDB, 1986: 289–293.

7 Graefe G, Linville A, Shapiro L. Sort versus hash revisited. IEEE Trans. on Knowledge and Data Eng., 1994,6(6):934–944.

8 Haas LM, Garey MJ, Livny M, Shukla A. SEEKing the Truth about Ad-Hoc Join Costs. The VLDB journal, 1997,6(3): 241–256.

9 Lo ML, Ravishankar CV. Towards eliminating random I/O in hash joins. Proc. of Int'l Conf. of Data Eng.,1996: 422–429.