文章编号:1001 - 9081(2010)08 - 2013 - 04

基于 Bin 位图索引的多维查询优化算法

王黎明,程 晓,柴玉梅
(郑州大学 信息工程学院,郑州 450001)
(chengxiao118@163.com)

摘 要:在属性基数(该属性可能的取值数)很高的情况下,简单位图索引需要占用太大存储空间。Bin 位图索引可以很好解决这个问题。这种索引不像简单位图索引那样建立在不同的属性值上,而是建立在属性范围上,但候选检查往往占用大部分的查询时间。为了提高查询性能,提出一种排序方法来对各属性进行排序,以减少候选检查数目,并在此基础上提出动态预扫描算法。实验结果表明,排序和动态预扫描算法都取得了良好的效果。

关键词:数据仓库;位图索引;多维查询;编码
中图分类号: TP18 **文献标志码**:A

Multi-dimensional query optimization algorithm for bitmap index with binning

WANG Li-ming, CHENG Xiao, CHAI Yu-mei
(*College of Information Engineering, Zhengzhou University, Zhengzhou Henan* 450001, *China*)

Abstract: The storage requirements for simple bitmap index is unacceptable when cardinality (the number of distinct values) of the attribute is very high. A common approach to solve this problem is to build bitmap index with binning. This technique builds a bitmap for each bin in a certain attribute range rather than distinct values. But this approach cause the time waste during candidate check because candidate check usually dominates the most of query processing time. In order to improve query performance, a new sort method for attributes was propsed to reduce the number of candidate check. And then a dynamic pre-scan algorithm was proposed. The experimental results show that our approach has a significant improvement over traditional strategies.

Key words: data warehouse; bitmap index; multi-dimensional query; encoding

0 引言

位图索引因为结构简单,并且二进制位运算效率高,被广泛使用于数据仓库、联机分析处理和科学应用。简单位图索引[1]占用的字节数为 $C \times (N/8)$,C 表示基数,N 表示记录数。在基数较高的情况下,位图索引需要占用大量存储空间。因此位图索引往往被认为只有在基数较低的情况下才适合使用。现今学者们已经提出很多方法来克服这个难题。这些方法从大的方面来说可以分为 3 类,即编码、bin 和压缩。

所有编码方法中,二进制编码[2]需要最少的位图数。但是这种编码查询效率较低,响应任何查询都几乎需要访问所有位图。Chan 和 Ioannidis 相继提出了范围编码[3]和 interval 编码[4]。范围编码对于单边查询效率很高,而 interval 编码则适合双边查询。

压缩是减少位图索引占用空间的重要方法。常用的位图压缩算法是字节对齐位图编码[5](Byte-aligned Bitmap Code, BBC)和字对齐混合编码[6](Word-Aligned Hybrid code, WAH)。WAH 虽然比 BBC 多占用约 50% 的空间,但响应时间却比 BBC 快 12 倍[7],近年来得到了广泛的应用。在基数非常大的情况下,用编码和压缩联合也因占用太大空间而不可行[8]。

Bin 方法可以很好地减少位图索引所需要的位图数,它的每一位不再代表单个不同的属性值,而代表一个属性值区间。用户可以根据需求来定索引的位图数,十分灵活。另外 Bin 位图索引还很适用于属性值包含浮点数的情况。但它同时带来了另一个难题,就是候选检查,而候选检查往往主导着整个查询时间。

本文提出一种基于 Bin 位图索引的多维查询优化算法,以减少多维情况下候选检查数目,提高查询效率。

1 Bin 位图索引

Bin 位图索引和大家熟知的简单位图索引相似,不过它不再像简单位图索引那样建立在不同的属性值上,而是建立在一个个的属性范围(bin_i)上。下面给出一个例子,采用了范围编码。

属性值	bin_0 [0,10)	bin_1 [0,20)	bin_2 [0,30)	bin_3 [0,40)
0	1	1	1	1
6.6	1	1	1	1
10.0	0	1	1	1
25.0	0	0	1	1
39.0	0	0	0	1
12.5	0	1	1	1
33.0	0	0	0	1

图 1 范围编码的 bin 位图索引

收稿日期:2010 - 02 - 01。

作者简介:王黎明(1963 -),男,河南浚县人,教授,博士,主要研究方向:分布式人工智能、机器学习、数据挖掘; 程晓(1984 -),男,河南南阳人,硕士研究生,主要研究方向:数据仓库、数据挖掘; 柴玉梅(1964 -),女,吉林长春人,副教授,硕士,主要研究方向:人工智能、机器学习、数据库。

如图1所示,该属性的取值范围为[0,40)。图中它分为4个bin,分别表示为bin_0、bin_1、bin_2、bin_3,其对应的取值范围分别是[0,10),[0,20),[0,30),[0,40)。当属性值落在哪个bin,那个bin相对应的位就置1;反之则置0。例如6.6落在[0,10)区间,则把bin_0的位图相应的位置1。因为所有值都落在bin_3内,实际中它可以省略。

2 一维查询

如图1所示,假定有一查询$x < 23$,则符合查询的值都落在bin_2中。但是落在bin_2中的值未必都符合查询,称bin_2为查询$x < 23$的边界bin。为求得准确的结果,还需要额外的步骤。对bin_1和bin_2进行异或运算,得到[20,30)之间的值,称这个区间为查询$x < 23$的真边界bin。落在真边界bin内的记录,不能只根据索引来确定其是否符合查询,还需要查看它的原始值。这额外的步骤就叫做候选检查。$x < 23$是单边查询,本文以下用的都是双边查询。双边查询可以看作单边查询的并。例如查询$5 \leqslant x < 23$,可以拆分为$x \geqslant 5$和$x < 23$。本例中查询$5 \leqslant x < 23$的边界bin为bin_0和bin_2,有3个值需要候选检查,其中只有6.6符合查询。

在一维的情况下,对每一个查询,边界bin的个数可能为0,1或2。如上例的查询$5 \leqslant x < 23$,边界bin个数为2。而对查询$0 \leqslant x < 20$,边界bin的个数为0。为了方便论述,假定本文以下各查询的边界bin个数都为2,且不是第一个或最后一个bin。真边界bin的位图分别用b_{c1}和b_{c2}表示。例如对查询$5 \leqslant x < 23$,b_{c2}就表示bin_1和bin_2进行异或运算的结果。

候选检查往往比较耗时,进行少量的候选检查也可能需要大量的磁盘存取操作。因为数据库中从磁盘上读取数据通常是页级的(典型的是4 KB或8 KB)。研究表明,候选检查是影响查询性能的重要因素[9]。在数据均匀分布的情况下,无论采用哪种bin策略,都有超过80%的查询时间花费在候选检查上。Rotem等人提出通过最优化bin边界来减少候选检查数目[9],此后又推广到多维情况[10]。Wu等人提出一种新的索引结构OrBiC[11],通过对各个bin内的数据进行聚集,以减少候选检查所需要的时间。所以总希望候选检查数目尽可能的少。

3 多维查询

首先给出假定。给定一数据集D,它有N个记录。每个记录有t个属性,分别为$A_1,A_2,\cdots,A_t$。每个记录$R \in D, R = v_1, v_2,\cdots,v_t$,其中$v_i$表示该记录在属性$A_i$上的值。对于多属性查询集$Q$,有查询$q \in Q, q = \bigcap_{i=1}^{t} r_i, r_i = [l_i,u_i)$,$r_i$表示在属性$A_i$上的查询范围。如果对所有$v_i, 1 \leqslant i \leqslant t$,都有$v_i \in r_i$,那么称该记录满足查询$q$。

为了便于描述,给出以下定义。

定义1 选择率。假定属性A_i中满足查询q(即属性值落在r_i区间)的记录数为n_s,则A_i的选择率$s_i = n_s/N$。

定义2 近似排除率。对于属性A_i,假定除去落在真边界bin内的记录,不符合查询q的记录数目为n_f,总记录数为N,则对应于A_i的近似排除率$e_i = n_f/N$。

定义3 近似选择率。属性A_i的近似选择率$s_i' = 1 - e_i$。

定义4 总近似选择率。有x个属性$A_1,A_2,\cdots,A_x$。这x个属性的总近似选择率为:

$$S = \prod_{i=1}^{x} s_i' 。$$

Stockinger等人在[12]探讨了一种有效的算法,在此称之为传统查询算法:它把查询分为t个步骤。第1步,对A_1进行查询,得到一个位图b_1。b_1的位数为N,它的第j位是$1(1 \leqslant j \leqslant N)$,表示第$j$个记录满足查询$q$在属性$A_1$的查询,即其$v_1 \in [l_1,u_1)$。第2步,首先用$b_1$和查询$A_2$用到的位图进行与运算,$b_1$中为假的位相应的记录不用再进行查询,从而减少了候选检查数目。以此类推,在进行第$i+1$步时,$1 < i < t$,先和第i步得到的位图b_i进行与运算。该算法的性能和各属性的查询顺序相关,为了得到更好的效果,Rotem等人提出了一个最优排序[9],排序权值为g_i,其中$g_i = w_i/(1 - s_i), 1 \leqslant i \leqslant t$。$w_i$为对$A_i$进行查询所需要的I/O花费,$s_i$为$A_i$的选择率。通过对$g_i$进行从小到大排序,可获得各属性最佳查询顺序。可是排序要在查询前进行,而查询前准确的s_i是未知的。

3.1 基于权值排序方法

假定给一查询q,查询A_i需要n_i个候选检查$(1 \leqslant i \leqslant t)$,属性$A_i$的选择率为$s_i$,各属性相互独立,那么按$A_1$到$A_t$的查询顺序,查询$A_i$需要的候选检查数目的期望是:

$$E(C_i) = \begin{cases} n_1, & i = 1 \\ n_i \prod_{j=1}^{i-1} s_j, & 1 < i \leqslant t \end{cases}$$

总候选检查数目的期望为:

$$E(C) = \sum_{j=1}^{t} C_j$$

定理1 各属性n_i都相等的情况下,选择率低的属性应该优先查询。

证明 不妨设$n_i = n$,s_i按下标值的顺序递增,按A_i下标顺序得到C的期望是:

$$E(C_{opt}) = n(1 + S_1 + S_2 + \cdots + S_{t-1})$$

其中$S_i = \prod_{j=1}^{i} s_j, 1 \leqslant i \leqslant t-1$。

如果有A_j和A_k没按下标顺序,$1 \leqslant j < k \leqslant t$,设此时候选检查总数期望为$C'$,则:

$$E(C' - C_{opt}) = n(s_i - s_j)(S_j' + S_{j+1}' + \cdots + S_{k-1}') > 0$$

其中$S_i' = \dfrac{S_i}{s_j}, j \leqslant i < k$。 证毕。

定理2 各属性s_i都相等的情况下,需要候选检查数目少的属性应该优先查询。

证明 不妨设$s_i = s$,n_i按下标值的顺序递增,则按A_i下标的顺序得到C的期望是:

$$C_{opt} = s(n_1 + n_2 s + n_3 s^2 + \cdots + n_t s^{t-1})$$

如果有A_j和A_k没按下标顺序,$1 \leqslant j < k \leqslant t$,设此时候选检查总数的期望为$C'$,则:

$$C' - C_{opt} = n_k(s^{j-1} - s^{k-1}) + n_j(s^{k-1} - s^{j-1}) = (n_k - n_j)(s^{j-1} - s^{k-1}) > 0$$ 证毕。

假定属性A_i的索引有k_i个bin,在创建索引时,这k_i个bin的端点就确定了。而数据仓库中的数据往往比较稳定,满足只读、批量更新的特性,所以可以在创建索引时记录下各个bin内为1的记录数。假定除去需要候选检查的(即除去落在真边界bin的记录),有n_f个记录不符合查询,b_{c1}和b_{c2}内为1的记录总数为n_c。分别用n_{c1}、n_{c2}表示b_{c1}、b_{c2}中为1的记录数,p_{c1}、

p_{c2} 表示 b_{c1}、b_{c2} 中不符合查询的长度占这个边界 bin 长度的比重。结合定理 1 和定理 2,定义该属性的权值 W_i 为:

$$W_i = (n_f + n_{c1} \times p_{c1} + n_{c2} \times p_{c2})/n_c; \quad n_c \neq 0$$

以此权值来对各属性进行排序,权值越高的属性越优先查询。此权值可以看作是对 g_i 的简化和近似实现。

举例说明,如图 2 所示,q_i 表示对一属性 A_i 的一个查询,该属性的索引分为 5 个 bin。为了描述方便,图中的 $bin_i' = bin_i$ XOR bin_{i-1},$2 \leqslant i \leqslant 5$。那么在该例中 n_f 为落在 bin_1、bin_5' 内的记录数,n_c 为落在 bin_2' 和 bin_4' 内的记录数。所有不满足查询 q_i 的记录数为 n_f 加上落在 ab 和 ef 内的记录数。落在 ab 和 ef 内的记录数在查询前是无法确定的,分别用估计值 $n_{c1} \times p_{c1}$ 和 $n_{c2} \times p_{c2}$ 来代替。在该例中,p_{c1} 即 ab 与 ac 的比值,p_{c2} 为 ef 与 df 的比值。

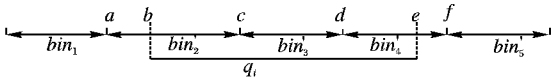

图 2　权值说明示例

基于权值排序无需事先知道工作量,通用性好,并且排序的代价也很小。首先,对每个属性需要一个长度为 k_i 的数组记录落在各个 bin 内的记录数,这 k_i 个数要从磁盘中读入。若采用的 bin 边界划分方法是 Equi-depth binning,则这个花费可以省去。t 表示属性的个数,排序的时间复杂度为 $O(t \log t)$。k_i 和 t 都跟记录数 N 无关,通常都不大。

3.2　动态预扫描算法

假定排序后各属性的优先顺序为 $A_1, A_2, \cdots, A_t$,则用传统查询算法不能减少 A_1 的候选检查数目。这使得 A_1 的候选检查数目很可能占据总候选检查数目的大部。故考虑在对 A_1 查询之前,先对其后面的 x 个属性进行预扫描($1 \leqslant x < t$),得到一个位图 b_f,记录下这 x 个属性中不需候选检查就可以确定为假的记录。用 b_f 和查询 A_1 用到的位图进行与运算,以减少候选检查数目。

此方法的关键在于确定 x 以及这 x 个属性。x 可以静态指定(静态预扫描),但这样没有考虑各查询的特性。本文提出动态预扫描算法,根据各查询的特性,动态确定 x 以及这 x 个属性。动态预扫描算法描述如下。

输入:查询 q。

输出:查询结果。

1)首先计算各属性的 W_i,根据 W_i 对各属性进行排序,不妨假设排序结果为:$A_1, A_2, \cdots, A_t$;

2)计算 $A_2, A_3, \cdots, A_t$ 的近似排除率 e_i,按 e_i 的降序对其排序。不妨假设排序结果为:$A_2', A_3', \cdots, A_t'$;

3)初始化 b_f,总选择率 $S = 1$。

4)FOR i =2 to t

　4.1) $S = S * (1 - e_i)$,预扫描 A_i',获得 b_f,并对 A'_i 做标记;

　4.2)IF($s < \varepsilon$)

　　break;

　END FOR

5)对 A_1 进行查询,但用到的位图先和 b_f 进行与运算;

6)FOR i = 2 to t

　6.1) IF(A_i 被标记了)

　　break;

　6.2)ELSE

　　对 A_i 进行查询,但首先和 n_f 进行与运算;

　END FOR

7)查询被标记的属性;

显然,对近似排除率越高的属性预扫描,越有利于减少候选检查数目。执行第 2)步使得被选中的 x 个属性的近似排除率比未选中的高。

假定单独查询 A_1 需要候选检查数目为 n_1,动态预扫描算法确定最后一个预扫描属性为 A'_{x+1},$0 \leqslant x < t-1$,则查询 A_1 需要候选检查数目的期望:

$$E(n_c) = n_1 S$$

如果再多预扫描一个属性 A'_{x+2},假定 A'_{x+2} 的近似排除率为 e_{x+2},查询 A_1 需要的候选检查数目的期望记为 $E(n'_c)$,则:

$$E(n'_c - n_c) = n_1 S e_{x+2}$$

如果定义的阈值 ε 已经很小,即 S 已经很小,则预扫描 A'_{x+2} 将不能明显减少 $E(n_c)$。

第 6)、7)步实际上是对各属性进行了重排。因为 s_i 与 s'_i 往往相差很小,先查询标记的属性不能明显减少未标记属性的候选检查数目。相比之下,先查询未标记属性可以有效减少已标记属性的候选检查数目。

动态预扫描相比使用排序的传统查询算法需要额外的花销。一是对除 A_1 外 $t-1$ 个属性根据近似排除率排序。近似排除率很容易获得,n_f 在基于权值排序中已经用到,N 是常量。所以这个排序的花费比第一遍排序小。若划分 bin 边界方法采用 Equi-depth binning,此排序可省略。二是对 x 个属性的预扫描。对每个属性进行预扫描,需要读入两个位图,并对其进行操作。虽然这两个位图在之后还会用到,但由于内存空间的关系,往往在用到时还需重新读入。独立查询一个属性需要扫描 4 个位图,预扫描的花销差不多是其一半。尽管候选检查占总查询时间的主导,预扫描的花销仍不能忽视。这就是为什么不把 x 指定为 $t-1$ 的原因,尽管那样可使总候选检查数目最少。

4　实验

为了测试算法的有效性,本文进行了实验。实验中用到了两个数据集。

数据集一:随机生成 1 000 万条记录,10 个属性,数据基本符合均匀分布。各属性的基数从 150 万到 250 万不等,都建立 100 个 bin。

数据集二:Bag of Words 数据集(UCI),选取了部分数据。数据包含 7 个属性,1 000 万条记录。其中 4 个属性的基数在 200 万左右,3 个属性的基数在 10 万 ~25 万,都建立 100 个 bin。

4.1　实验 1

在未知工作量的情况下,常用划分 bin 边界的方法有等宽法(Equi-width binning,每个 bin 有相同的宽度)和等深法(Equi-depth binning,落在每个 bin 内的记录数大致相等)。在数据集一上对基于权值排序进行实验,用传统查询算法,随机生成 50 个查询,统计所需的候选检查数目。因为在数据均匀分布情况下,等宽法和等深法十分相似。在此只给出等深法上的实验结果,如图 3 所示。

从图 3 可以看到,基于权值排序在传统查询算法上取得了良好的效果。维数越多,顺序相对于逆序效果越明显。在 10 维情况,逆序查询所需候选检查数目基本是顺序查询的 2 倍。基于权值排序并不保证是最优排序。在二维数据查询实验中,偶尔会有逆序比顺序还优的情况,不过此时二者所需候选检查数目很相近。可以说此排序是近似最优排序。