论文
《UpBit: Scalable In-Memory Updatable Bitmap Indexing》
概述

计算机科学与技术学院
马玉坤
1150310618

2017 年 5 月 9 日

1 前言

该论文提出了 Upbit——一种支持对数据进行高速修改或查询的位图索引的实现方法。

1.1 位图索引

位图索引 (Bitmap Index) 由 P’ONeil 在 1987 年提出，并在一个商用数据库系统 Model 204 上首次应用。在数据库中，无论是用于科研用途，还是商业用途，位图向量都被广泛应用。[3] 最原始的位图索引利用位向量 (Bit Vector) 来表示某种被索引的属性在数据集中的索引情况。例如在表 1.1中：在“数学成绩”一列中“优秀”这个属性的位图向量为 101，分别代表小 A 拥有此属性、小 B 未拥有此属性，小 C 拥有此属性。将不同属性的位向量进行位逻辑运算，即可以回答各种复杂的信息。

位图索引尽管是被设计来高效地进行大量的数据库查询操作，但相对少量的数据库修改操作有时也是必要的。如何高效的能够让位图索引在能够进行高效查询的同时进行高效的修改，是数据库领域的热门问题。

该论文使用了维护更新向量和分块指针的方法，较好地解决了使用位图向量高效进行查询和修改操作的问题。

表 1: 一个普通的表

姓名	数学成绩	语文成绩
小 A	优秀	及格
小 B	良好	优秀
小 C	优秀	不及格

2 论文思想

2.1 Update Conscious Bitvector

在本论文前，已有一种叫做 Update Conscious Bitvector（下简称 UCB）的支持修改的位图索引技术被提出。该篇论文提出的 Upbit 即是在 UCB 的基础上加以改进的成果。UCB 的主要思想如下：

使用 WAH[5] 算法对位向量进行压缩后，压缩后的位向量上原地修改的效率堪忧，但是容易发现，在压缩后位向量的最后面添加一位的速度是极快的。对于可修改的位图向量的优化，一个直接想法是，既然修改操作并不多，原地修改的效率堪忧，为什么不把修改操作转化为禁用 (删除)+ 添加操作？这即是 UCB(Update Conscious Bitmaps) 的主要思想。

图 1: UCB 图解

在 UCB 中，每行都有一个额外的位向量叫做存在向量 (Existence Bitvector, EB)。当 EB_i 为 1 时，表示该行有效，否则该行无效（Invalid）。如图 1，当我们想要把第二行的值从 20 改为 10 时，我们只需要将 EB_2 设为 0，然后在位向量后新建一位，并将对应的 EB 设为 1。

查询时，我们需要将对应的属性的位向量与存在向量进行“逻辑与”操作。

实验中，UCB 的确能极大地提高伴有少量修改操作的查询效率的提高，但是随着修改操作的积累，存在向量的复杂性将大大提高，UCB 查询的效率将会极大地下降。如何使位图向量查询的效率随着修改操作积累不明显提高？这便是 Upbit 的创新之处。

2.2 Upbit

实际上，上一个问题的解决方案并不难。既然随着修改次数的增加，EB 的复杂度提高，造成了查询的瓶颈。为什么我们不在修改次数达到一定数量级时，通过修改各个属性的位向量来维护存在向量，使存在向量保持高压缩性，提高查询效率？

UpBit 使用了这种策略。UpBit 是一个新提出的位图向量方案，能够在保持修改效率较快的情况下，保持查询的效率。

UpBit 并没有只使用一个存在向量，而是对每个属性都使用了一个更新向量（Update Bitvector，UB）。每一个属性的位向量，用两个位向量经“位异或”计算得出，这两个位向量分别为值向量 (Value Bitvector,VB) 与更新向量 (Update Bitvector,UB)。

由于每一个属性的位向量用两个位向量经“位异或”计算得出。所以无论我们修改 UB 或者 VB，都相当于对属性进行修改。在每次修改操作时，我们直接修改 UB 的值。如图 2，当我们想要把第二行的值从 20 修改为 10，只需要把属性 20 对应的 UB_i 取反（从 0 变为 1 或者从 1 变为 0），然后把属性 10 对应的 UB_i 取反。

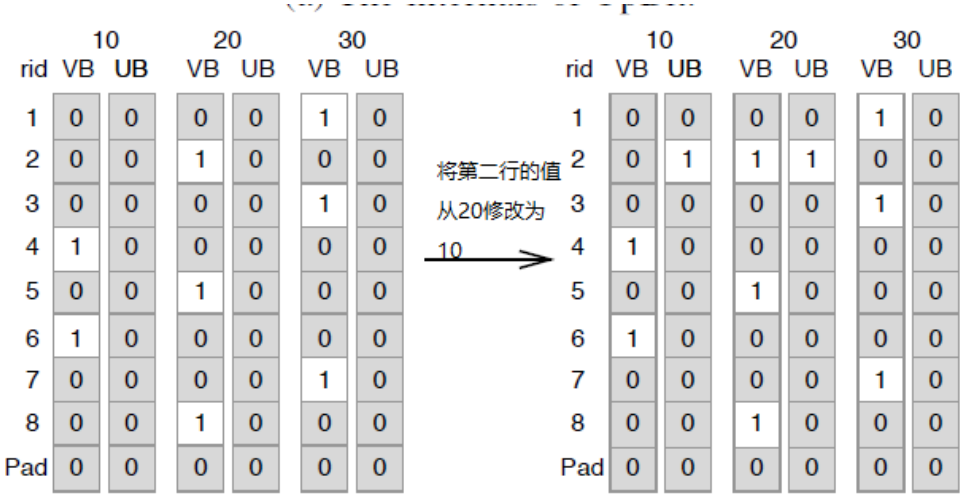

图 2: Upbit 图解

2.2.1 定时维护更新向量

当某个属性的 UB 的修改次数达到一定阈值时，我们将该属性的 VB 与 UB 做异或操作，将结果保存到 VB 中，然后将 UB 设为全 0 的向量，这便是对 UB 的维护。这样的维护可以保证随着修改操作积累，查询的效率仍然可以很快。

2.2.2 维护向量的分块指针

同时，UpBit 还使用了第二种关键的提高效率的方法——块指针 (Fence Pointers)。当我们对某一行的值进行修改时，首先就需要找到这一行修改前的值。在寻找这一行修改前的值的过程中，我们需要在每一个属性的压缩后的 UB 和 VB 中查询这一行对应的二进制位的值。如何在压缩后的位向量中，快速找到第 i 位的值，是一个与效率关系极大的问题。块指针的思想是：将未压缩前的位向量按下标分为若干连续的块，每一块的块大小都接近 g（g 是人为给定的值）。对于每个位向量，维护每一块的起始位置对应的行在压缩后的位向量中的位置，就可以在 O(g+log N/g) 的时间复杂度内快速在每个压缩后位向量中找到任意一行的位置，并可在 O(N/g) 的时间复杂度内维护修改后位向量的块指针。

3 基本算法

3.1 获取某一行的值

获取第 i 行的值时，需要枚举值域，直到找到某个值对应的 UB 和 VB 满足 $UB_i \oplus VB_i = 1$。

get_value (index: *UpBit*, row: *k*)

1: **for** each i $\in \{1, 2, ..., d\}$ **do**
2: $temp\_bit = V_i.get\_bit(k) \oplus U_i.get\_bit(k)$
3: **if** $temp\_bit$ **then**
4: Return val_i
5: **end if**
6: **end for**

图 3: 获取某一行的值

3.2 更新某一行的值

更新某一行的值时，首先要获取当前行的原始的值（可以使用 get_value()），然后再将两个值对应的 UB_i 都分别进行取反。

update_row (index: *UpBit*, row: k, value: *val*)

1: Find the i bitvector that *val* corresponds to
2: Find the old value *old_val* of row k
3: Find the j bitvector that *old_val* corresponds to
4: $U_i[k] = \neg U_i[k]$
5: $U_j[k] = \neg U_j[k]$

图 4: 修改某一行的值

3.3 合并 UB 与 VB

当 UB 的修改次数达到一定阈值（阈值需人为设定）时，需要将 UB 与对应的 VB 合并，然后将 UB 置 0。下图中 FP 为 Fence Pointers。

merge (index: *UpBit*, bitvector: i)

1: $V_i = V_i \oplus U_i$
2: $comp\_pos = 0$
3: $uncomp\_pos = 0$
4: $last\_uncomp\_pos = 0$
5: **for** each i $\in \{1, 2, ..., length(V_i)\}$ **do**
6: **if** $isFill(V_i[pos])$ **then**
7: $value, length += decode(V_i[pos])$
8: $uncomp\_pos += length$
9: **else**
10: $uncomp\_pos++$
11: **end if**
12: **if** $uncomp\_pos - last\_uncomp\_pos > THRESHOLD$ **then**
13: $FP.append(comp\_pos, uncomp\_pos)$
14: $last\_uncomp\_pos = uncomp\_pos$
15: **end if**
16: $comp\_pos++$
17: **end for**
18: $U_i \leftarrow 0s$

图 5: 合并 UB 与 VB

4 算法分析

该算法原理巧妙简洁，并能对位图向量的效率带来极大的提升。

4.1 空间复杂度分析

由于使用了 WAH 压缩，其空间复杂度为 $O(R)$[1]。Fence Pointers 的空间复杂度与 g 有关，但仍小于 $O(R)$。故总空间复杂度为 $O(R)$。

4.2 时间复杂度分析

查询某一行的值的时间复杂度为 $O(C(g+\log\frac{R}{g}))$，修改某一行的值时，由于比查询多了维护 Fence Pointers 的代价，故时间复杂度为 $O(C(g+\log\frac{R}{g}+R/g))$。

4.3 举例

见 2.2。

参考文献

[1] Athanassoulis M, Yan Z, Idreos S. UpBit: Scalable In-Memory Updatable Bitmap Indexing[C]//Proceedings of the 2016 International Conference on Management of Data. ACM, 2016: 1319-1332.

[2] Canahuate G, Gibas M, Ferhatosmanoglu H. Update conscious bitmap indices[C]//Scientific and Statistical Database Management, 2007. SSBDM'07. 19th International Conference on. IEEE, 2007: 15-15.

[3] 程鹏. 位图索引技术及其研究综述 [J]. 科技信息, 2010 (26): 134-135.

[4] Wu K, Ahern S, Bethel E W, et al. FastBit: interactively searching massive data[C]//Journal of Physics: Conference Series. IOP Publishing, 2009, 180(1): 012053.

[5] Wu K, Otoo E J, Shoshani A. Compressing bitmap indexes for faster search operations[C]//Scientific and Statistical Database Management, 2002. Proceedings. 14th International Conference on. IEEE, 2002: 99-108.

[1] 设行数为 R，值域大小为 C。