可修改的位图索引技术

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。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中：

|  |  |  |
| --- | --- | --- |
| 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 |

表1. 一个普通的表

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。

WAH: 一种位图索引压缩算法

如果对一个表中的每一列的每个属性都维护一个不经压缩的位向量，显然所需要的空间复杂度是O(N\*A)，其中N为行数，A为不同列的不同属性的总数量，即使这些位向量中总共只有N个位上是“1”，其他位全部为“0”。因此，对位图索引进行压缩是非常有必要的，不仅是对于存储所需要的空间复杂度，同时也是对于询问所需要的时间复杂度。

在众多位图索引压缩算法中（例如BBC、WAH、Bit—slice方法），WAH是一种较为有效且实现简单的编码方式，本文着重讲解WAH (Word—Alignment Hybrid Code)编码。WAH的主要思想如下：出于对内存对齐的考虑，WAH每个字视作一个压缩单元，也就是说压缩后的编码每个字为一个单元，每个单元相对独立。WAH中有两种字，一种是本字(literal word)，一种是缩字(fill word)。对于一长串连续的“0”或者连续的“1”，我们记录下这一长串的长度，记在缩字中，并将该缩字的最高位标记为“1”，表示这个字是缩字。将计算机字长设为w，对于其他情况，我们把每w-1位（因为最高位要标记该字为本字）放到一个本字中，表示该w-1位未被压缩。

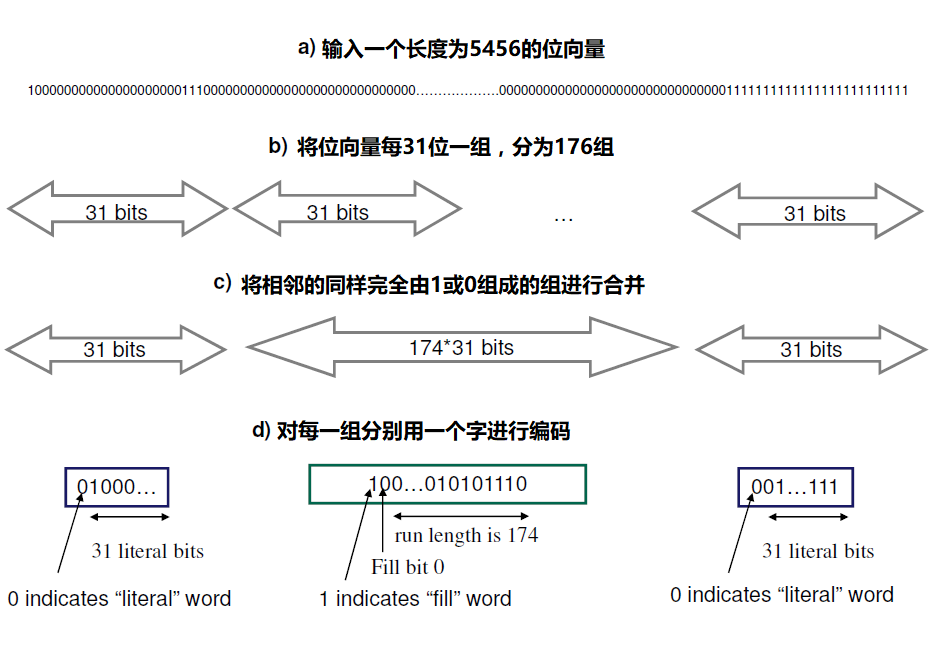


图1. WAH图解

例如，图1中，我们先把5464位二进制数每31位分成一组，然后将相邻的连续的0或1组成的组合并成更大的连续的0或1组成的组。最后连续的0或1组成的组将会用缩字来存储，其他组使用本字来存储。

那么现在我们有了位图索引的压缩方案，可以快速地回答各种询问。但在真实的使用场景中，我们有时候需要对位图索引进行修改。如何使我们的位图索引支持修改？如果使用解压缩-原地修改-重新压缩的方法，随着修改次数的提高，整个位图向量的耗时将明显增大。如何能设计出一种位图向量，使得我们能够支持快速进行少量的修改操作的同时，能够保持位图向量快速进行查询的优势？

Update Conscious Bitmaps

尽管在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。

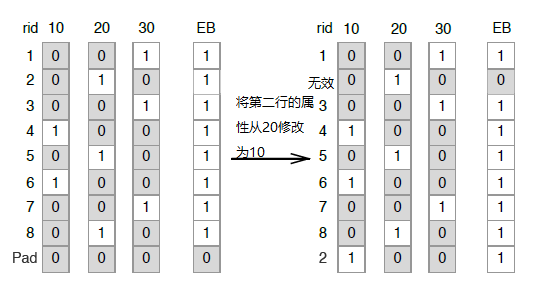


图2. Update Conscious Bitmaps更新原理

在UCB中，每行都有一个额外的位向量叫做存在向量(Existence Bitvector, EB)。当为1时，表示该行有效，否则该行无效（Invalid）。如图2，当我们想要把第二行的值从20改为10时，我们只需要将设为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？

UpBit: 使用内存、可伸缩且可修改的位图索引

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)。

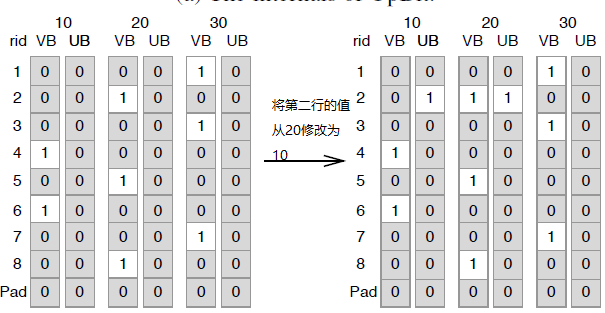


图3. UpBit更新原理

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。如上图，当我们想要把第二行的值从20修改为10，只需要把属性20对应的取反（从0变为1或者从1变为0），然后把属性10对应的取反。

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。

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。

参考文献

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.

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.

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

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.

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.

马玉坤 18845895386