Analysis of Different Implementations of Bitmap Index

Harbin Institute of Technology

Department of Computer Science and Technology

Ma Yukun

1150310618

2017/11/18

Abstract

Bitmap Index is a widely used data structure in the database. It has a very low complexity to program. There are many kinds of bitmap indexes presented by different people. This article mainly focuses on the time and space analysis of these kinds of bitmap indexes and try to offer suggestions about how to choose appropriate implementations of bitmap index. Finally this article will try to show the goodness of the combination of different implementations.

1 Background

1.1 Bitmap Index

Bitmap Index is presented by P. O'Neil in 1987. It was first applied in a commercial database (Spiegler & Maayan 1985). In the application of database, either for scientific purposes or for commercial purposes, bitmap indexes are widely used.

The original bitmap indexes uses *Bit Vector* to indicate the indexed attributes in the database. For example, in Table 1, in the column named "math score", the Bit Vector for value "A" is *101*, indicating that Alice has an A, Bob does not have an A and Dean has an A.

Table 1: An Ordinary Table

name	math score	Chinese score
Alice	A	C
Bob	B	A
Dean	A	D

When we use the Bit Vectors of different attributes to do logical math, we can get different results to get the answers for so many kinds of queries. Take this as another example: We want to know the students who has an A in math and a C in Chinese. Then we can use the result of the *bitwise and* of the Bit Vector of "math A" (which is **101**) and the Bit Vector of "Chinese C" (which is **100**). As the calculation below, the result will be **100** which indicates that Alice is the only one who gets an A in math and a C in Chinese.

$$101 \tag{1}$$

$$\&100 \tag{2}$$

$$=100 \tag{3}$$

The example above is good for showing the basic idea of bitmap indexing. But in the industrial application, the trivial implementation of the Bitmap Index is not often used because of its high complexity of time and space. So there are tons of different

implementations of bitmap indexes that can lower the time complexity and the space complexity.

1.2 Time Complexity and Space Complexity

In the analysis of algorithms, time complexity and space complexity are two methods to measure the efficiency of an algorithm.

The time complexity can measure or estimate the time consumed for running one algorithm. Not specifically, a bigger time complexity usually means more time consumed for the same input. While the space complexity can measure or estimate the space consumed for running one algorithm. Similaryly and not specifically, a bigger space complexity usually means more space consumed for the same input.

In section 2 these implementations of Bitmap Indexes will be revealed. And in section 3 these implementations will be discussed analyzed to show their time complexity and space complexity. Finally we will give the comparison of these implementations of bitmap indexes in Section 4.

2 Different Implementations of Bitmap Indexes

Here, we focus on three kinds of bitmap indexes. They are value-based bitmap index, range-based bitmap index and bit-transposition bitmap index, respectively.

For simplicity, let's have a table T (who has n tuples), and $T = t_1, t_2, \ldots, t_n$. Let

A be a column of T, and there are m different values in column A. We can assume the m different values to be the integers ranged from 0 to $m-1$. Let B be a bit vector $(b_1, b_2, \ldots, b_n)$ whose length is n. Any bit of B b_i can also be referenced as $B[i]$.

2.1 Value-Based Bitmap Index

The value-based bitmap index is the most fundenmental bitmap index. It consists of m Bit Vectors $B_{m-1}, \ldots, B_j, \ldots, B_1, B_0$, it can be regarded as a $n \times m$ matrix who has n rows and m columns, and B_j is the Bit Vector corresponding to the j-th attribute. $B_j[i] = 1$ iff $t_i.A = j$, $B_j[i] = 0$ otherwise.

Figure 1(a) shows the values of different rows on column A of some table T. Figure 1(b) shows the m ($m = 9$) bitmap indexes for each value.

It can be observed that the value-based bitmap index is very space-consuming since it will get each value for each column a bit vector whose length is n (the number of rows).

2.2 Range-Based Bitmap Index

Range-based bitmap index has $m-1$ Bit Vectors $B_{m-2}, \ldots, B_j, \ldots, B_1, B_0$. B_j corresponds to the value j. $B_j[i] = 1$ iff $t_i.A \leq j$. Range-based bitmap index can be regarded as an accmulation of value-based bitmap index. In a value-based bitmap index, every bitmap index corresponds to a indexed value while in a range-based bitmap index every bit vector represents the the indexed values which are less than or equal to the

$\pi_A(T)$
3
2
4
8
5
3
1
6
7
0

(a) Column A of T

B_8	B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0
0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0
0	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1

(b) Bitmap Indexes of Column A

B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0
1	1	1	1	1	0	0	0
1	1	1	1	1	1	0	0
1	1	1	1	0	0	0	0
0	0	0	0	0	0	0	0
1	1	1	0	1	0	0	0
1	1	1	1	1	0	0	0
1	1	1	1	1	1	1	0
1	1	0	0	0	0	0	0
1	0	1	0	0	0	0	0
1	1	0	1	1	1	1	1

(c) Range-Based Bitmap Index

A_1	A_0
1	0
0	2
1	1
2	2
1	2
1	0
0	1
2	0
2	1
0	0

(d) Child Columns of Column A

B_2^1	B_1^1	B_0^1
0	1	0
0	0	1
0	1	0
1	0	0
0	1	0
0	1	0
0	0	1
1	0	0
1	0	0
0	0	1

(1)

B_2^0	B_1^0	B_0^0
0	0	1
1	0	0
0	1	0
1	0	0
1	0	0
0	0	1
0	1	0
0	0	1
0	1	0
0	0	1

(2)

(e) Bit-Transposition Bitmap Index

Figure 1: Bitmap Indexes

corresponding value. Figure 1(c) shows the range-based bitmap indexes on column A.

2.3 Bit-Transposition Bitmap Index

Bit-transposition bitmap index is presented by Chan and Ioannidis (Wong et al. 1986). It represents each value by using the r carry system. In the r carry system, each integer x ranged from 0 to $m-1$ can be represented by a integer sequence $(x_{e-1}, \ldots, x_1, x_0)$, in which $e = |\log_r m|, 0 \leq x_e, \ldots, x_1, x_0 < r, x = x_{e-1} \times r^{e-1} + \ldots + x_1 \times r + x_0$. The integer sequence whose length is e can be regarded as values from e columns. Hence the column A can be divided into e child columns $A_{e-1}, \ldots, A_1, A_0$.

The main concept of bit-transposition bitmap index is to divide each column A to some child columns by representing the values of column A in r carry system and then to build a bit vector for each child column of column A. The number r can be called base of the bitmap index. For the data in Figure 1(a), the column A can be divided into two child columns A_0, A_1 when $r = 3$ as showed in Figure 1(d). As in Figure 1(e), the (1) and (2) are the value-based bitmap indexes for A_1 and A_0 respectively. In Figure 1(e), the column B_p^k indicates the bit vector corresponding to value p in the k-th child column.

3 Analysis of Implementations of Bitmap Indexes

The three different implementations of bitmap indexes, namely, value-based bitmap index, range-based bitmap index and the bit-transpositino bitmap index's main concepts are all described above. It can be seen that there are many differences between them in many ways such as the principle, the space needed for storing the index and the time needed to process queries.

Let's analyze the time and space needed for each implementation of bitmap index. Rather than 3 types of bitmap indexes, we will inspect 4 types of bitmap indexes, namely, trivial value-based bitmap index, trivial range-based bitmap index, value-based bit-transposition bitmap index and range-based bit-transposition bitmap index.

3.1 Space Measurement

Due to the design of the value-based bitmap index and the range-based bitmap index, the two bitmap indexes will need $O(m)$ bit vectors to index a table in a database.

And for the value-based bit-transposition bitmap index and range-based bit-transposition bitmap index, the required number of bit vectors is $O(e \times r)$ where e equals to $\log_r |V|$ and r is the base we choose.

3.2 Time Measurement

For a query, the number of bit vectors involved for a range-based bitmap index is $O(1)$ while the number for a value-based bitmap index is $O(m)$ where m is the number of different values in one column. While the number of bit vectors involved for a bit-transposition bitmap index is much smaller which is in $O(e)$.

4 Conclusion

In general, the value-based bitmap index and the range-based bitmap index are better considering the query efficiency. For a equal-type query (a query in the form of $A = x$), the value-based bitmap index only needs one bit vector, and the range-based bitmap index needs only two bit vector. But in the contrast, when m is large and x is large, the range-based bitmap index and the value-based bitmap index has very poor performance in the query since the involved bit vector increases linearly as the range increases. In the other hand, as the number of kinds of different values of one column A increases, the value-based bitmap index and range-based bitmap index will need more space. When the number m gets larger than eight times as the the length of the tuple, the space needed to store the bitmap index is larger than the data space, which influences its application severely.

The bit-transposition bitmap index has a very strong flexibility. By adjusting the value of the base r, we can easily get the equilibrium point of space the time. By in-

creasing r we can reduce the time needed in query while increase the space needed. In contrast, decreasing r will lead to the increase of the time needed and decrease the space needed. When $r = m$, the bit-transposition bitmap index turns into a value-based bitmap index. Table 2 shows the comparison of the four bitmap indexes.

Table 2: Comparison of Four Kinds of Bitmap Indexes

Name	# of Bit Vectors	Time Consumed for $A = x$	Time Consumed for $A \leq x$
value-based bitmap index	m	1	x
range-based bitmap index	$m - 1$	2	1
r-based value-based bit-transposition bitmap index	$e \times r$	e	$x_{e-1} + \ldots + x_1 + x_0$
r-based range-based bit-transposition bitmap index	$e \times (r - 1)$	$e \times 2$	$2 \times (e - 1) + 1$

References

Spiegler, I. & Maayan, R. (1985), 'Storage and retrieval considerations of binary data bases', *Information processing & management* **21**(3), 233–254.

Wong, H. K. T., Li, J. Z., Olken, F., Rotem, D. & Wong, L. (1986), ‘Bit transposition for very large scientific and statistical databases’, *Algorithmica* **1**(1), 289–309.

URL: *https://doi.org/10.1007/BF01840449*