A Performance Comparison of bitmap indexes[*]

Kesheng Wu Ekow J. Otoo Arie Shoshani
Lawrence Berkeley National Laboratory, Berkeley, CA
{kwu,ejotoo,ashoshani}@lbl.gov

ABSTRACT

We present a comparison of two new word-aligned schemes with some schemes for compressing bitmap indexes, including the well-known byte-aligned bitmap code (BBC). On both synthetic data and real application data, the new word-aligned schemes use only 50% more space, but perform logical operations on compressed data 12 times faster than BBC. The new schemes achieve this performance advantage by guaranteeing that during logical operations every machine instruction performs useful work on words rather than on bytes or bits as in BBC.

1. INTRODUCTION

Bitmap indexes are useful for various database applications such as data warehousing. The data structures used to represent these bitmaps should be designed to provide efficient search operations. For applications that access very large databases, the bitmap indexes can have millions to billions of bits. It is therefore imperative that the indexes be stored as compactly as possible. This leads to the development of various bitmap compression schemes. However, most compression algorithms are designed without consideration of the logical operations on the compressed data. These operations are significantly slower with compression than without compression. For this reason, most commercial implementations of bitmap indexes don't compress their bitmaps. A number of schemes specifically designed to compress bitmap indexes have been studied [2]. They are in general more efficient than the generic compression schemes. One of them, the *byte-aligned bitmap code* (BBC), is very efficient and is used in a commercial system, ORACLE [1]. However, even this scheme can be much slower than the uncompressed bitmaps [2]. In this work, we propose two new word-based schemes that are designed to always perform logical operations on entire words rather than bytes or bits as in BBC. Thus, they are expected to perform logical operations faster.

[*]This work was supported by the Director, Office of Science, Office of Laboratory Policy and Infrastructure Management, of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098.

		bitmap index			
i	**X**	< 1	[1,3]	[4,6]	> 6
1	1	0	1	0	0
2	4	0	0	1	0
3	7	0	0	0	1
4	6	0	0	1	0
		b_1	b_2	b_3	b_4

Figure 1: A sample bitmap index for an attribute X with integer values.

The main conclusion of this paper is that these new word-aligned schemes take only 50% more space than BBC, but perform logical operations 12 times faster on both real application data and synthetic data. At first, we expected a performance gain of about a factor of 4 since the logical operations perform directly on words rather than bytes. However, because BBC was designed to achieve good compression, it frequently needs to check individual bits within bytes during logical operations. For this reason, the new schemes are even more efficient than anticipated [4, 5].

Next, we use an example to illustrate why logical operations are important to bitmap indexes. Figure 1 shows one bitmap index for an attribute of a tiny table **T** consisting of only one attribute and four tuples (rows). The attribute **X** contains integer values that are divided into four ranges (also called bins). In this case, each bit sequence is associated with a bin and represents whether a value falls in the bin or not. To answer a range query, some bit sequences are combined together using bitwise logical operations. To process the query

```
select * from T where X<4;
```

one performs the logical operation (b_1 OR b_2), which consists of a bitwise logical OR between b_1 and b_2. We call the data structure representing the compressed bit sequences *bit vectors*. In this work, we briefly describe the word-aligned compression schemes that define the memory layout of these bit vectors. More details are available in two technical reports [4, 5].

2. REVIEW OF BYTE BASED SCHEMES

In this section, we briefly review three well known schemes for representing bitmaps and introduce terms needed to explain the word-aligned schemes.

A straightforward way of representing a bit sequence is to use one bit of computer memory for one bit of the sequence. We call this the *literal* (LIT) bit vector. It does not compress bit sequences but the logical operations on literal bit vectors are extremely simple and fast.

The second type of schemes in our comparisons are the general purpose compression schemes. We choose gzip as a representative of such schemes. The underlying algorithm gzip, LZ77, is known to be asymptotically optimal.

There are a number of compression schemes that offer good compressions and also allow fast bitwise logical operations as mentioned earlier. One of the best known schemes is the BBC scheme [1, 2]. BBC can perform bitwise logical operations very efficiently compared to other compression schemes. In addition, it compresses almost as well as gzip. We use BBC as the representative of the byte based compression schemes.

3. WORD BASED SCHEMES

All known compression schemes are byte based, that is, they access computer memory one byte at a time. Modern computers are word based. They read data one word at a time and can perform operations on whole words. For this reason, we consider two new word based schemes, the *word-aligned hybrid run-length code* (WAH) and the *word-aligned bitmap code* (WBC). This section contains brief descriptions of these schemes; for details see [4, 5].

Like BBC, the new schemes are based on the basic idea of run-length encoding that represents consecutive identical bits (also called a *fill*) by their bit values and lengths. The bit value of a fill is called the fill bit. If the fill bit is zero, we call the fill a *0-fill*, otherwise it is a *1-fill*.

An essential property of BBC is the byte alignment property. In designing the word-aligned schemes, we need to define the corresponding word alignment property. After some careful study [5], we realize that it isn't enough to ensure that the fills are represented by whole words. An additional requirement is needed to ensure that only whole words are accessed and operated on during bitwise logical operations. Next, we explain this requirement on each scheme separately.

Word-aligned hybrid run-length code. This is based on the *hybrid run-length code* (HRL) that represents long fills using run-length encoding and represents short fills literally. There are two types of code words in HRL: *literal* words and *fill* words. In our current 32-bit implementation, we use the Leftmost Bit (LMB) of a word to distinguish between a literal word (0) and a fill word (1). The lower 31 bits of a literal word contain the bit values of the sequence. The second leftmost bit of a fill word is the fill bit and the 30 lower bits store the fill length. HRL represents all fills and literal bits in whole words, however its performance isn't as good [5]. The *word-aligned hybrid run-length code* (WAH) imposes another requirement on the fills. We regard this as the word alignment requirement and it demands that all fill lengths be integer multiples of 31 bits (i.e., literal word size). We also represent fill lengths in multiples of literal word size, for example, the length of a 62-bit fill is two (2).

Figure 2 on the next page shows a WAH bit vector representing a 128-bit sequence. The second line shows how the bit sequence is divided into 31-bit groups and the third line shows the hexadecimal representation of the groups. The last line shows the values of the words used in WAH coding.

LIT	gzip	BBC	WAH	WBC
12.4	2.01	2.43	3.60	3.50

Figure 3: Total sizes (MB) of the bitmap indexes stored in various schemes.

The first three words are normal code words, i.e., two literal words and one fill word. The fill word `80000002` indicates a fill of two-word long containing 62 consecutive zero bits. The fourth word is a special literal word that represents the four bits that can't fit into regular code words and the last word indicates the number of useful bits in fourth one.

Word-aligned bitmap code. This scheme is designed to mimic the behavior of the BBC scheme. In this case, we first group bits of a sequence into words, then group words into runs. A run contains a fill followed by a group of literal words called a tail. On a 32-bit machine, a literal WBC word contains 32 bits from the sequence it represents. The word alignment requirement demands that all fills be multiples of 32 bits. A header word is used for each run. It contains three pieces of information, the fill bit, the fill length and the tail length. Both the fill length and the tail length are measured in number of words. In our current 32-bit implementation, we use the rightmost 16 bits to store the tail length, the LMB to store the fill bit and the remaining 15 bits to store the fill length.

4. PERFORMANCE COMPARISONS

In this section, we present some performance results on both synthetic data and real application data. The timing values were obtained on a Sun E 450 running 400 MHz UltraSPARC II. The tests were performed on a dataset from a real application and a set of synthetic data.

The table in Figure 3 shows the sizes of the same bitmap index compressed using five different schemes. The index is for a set high-energy physics data from an experiment called STAR [1] [3]. The dataset contains about 840,000 tuples. The index is only for the 12 most popular attributes. From Figure 3, we see that all compressed bit vectors use less than 30% of the space needed by the uncompressed literal scheme (LIT). The two word-aligned schemes use about 50% more space than BBC and they use about 80% more space than the optimal scheme, gzip. This is to be expected since WAH and WBC are not designed to minimize space but rather to minimize logical operation time.

Figures 4 and 5 show the logical operation time on the STAR data and the synthetic data respectively. These two graphs show the logical operation time against the average compression ratio of the two operands involved in the operation. The compression ratio is defined as the ratio of the compressed bit vector size to its uncompressed counterpart. The logical operation time shown always include the IO time. To see plots containing separate IO time and logical operation time, we refer the readers to [4, 5]. Overall, we see that the two word-aligned schemes usually need less time than BBC. The word-aligned schemes are always significantly faster than gzip and also usually faster than the literal scheme.

We have hundreds of test cases on the STAR data. If we

[1]More information about the STAR experiment is available at `http://www.star.bnl.gov/STAR`.

128 bits	1,20*0,3*1,79*0,25*1			
31-bit groups	1,20*0,3*1,7*0	62*0	10*0,21*1	4*1
groups in hex	40000380	00000000 00000000	001FFFFF	0000000F
WAH (hex)	40000380	80000002	001FFFFF	0000000F 00000004

Figure 2: A WAH bit vector. Each WAH code word (last row) represents a multiple of 31 bits from the bit sequence, except the last two words that represent the four leftover bits.

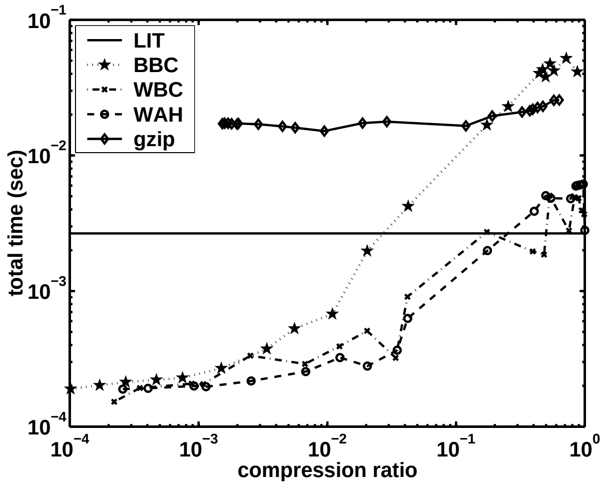

Figure 4: Logical operation time (including IO) on the STAR data (840,000 bits per bit vector).

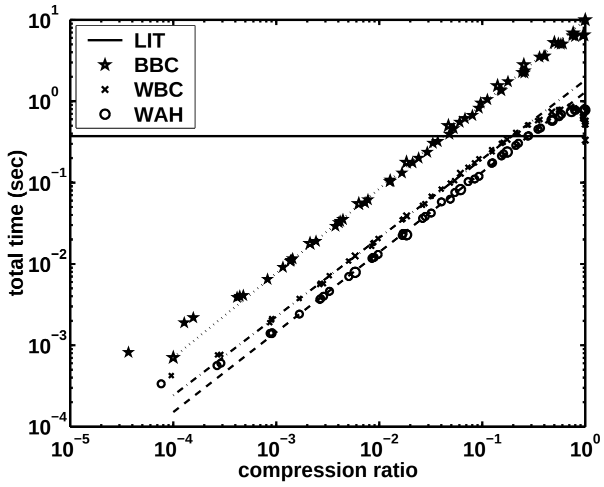

Figure 5: Logical operation time (including IO) on the synthetic data (100 million bits per bit vector).

sum up the time used in all test cases for each compression scheme, the total value for BBC is about 12 times as large as those of either WAH or WBC.

In Figure 4, when the compression ratios are smaller than 10^{-3}, BBC appears to use the same amount of time as WAH and WBC. In these cases, the data files containing the bit vectors are very small (less than 100 bytes) and the logical operation time is dominated by the IO overhead for reading these small files. It is possible to reduce this overhead using various strategies in which case the difference between BBC and the word-aligned schemes would be similar to those cases where the files are larger.

To quantify how well the compressed schemes perform against the uncompressed one, we look at the points where the lines for the compressed schemes cut the LIT line. BBC cuts it at about 0.03 while the word-aligned schemes cut the LIT line at about 0.3. Since the bit vectors, compressed using either WAH or WBC, usually have a compression ratio less than 0.3, logical operations on compressed data are usually faster than on uncompressed data.

As mentioned before, because the word-aligned schemes are designed to take full advantage of the computing hardware, we expected them to run faster than byte-aligned schemes such as BBC. However, the 12-fold difference was a surprise. To verify that the observed performance differences are not unique to the STAR data and to see how the different schemes behave on larger problems, we conducted a number of tests on synthetic data; see Figure 5. In this case, each bit vector represents 100 million bits. The dashed, dotted and dash-dotted lines in the figures are linear regression on the timing data with compression ratios between 10^{-3} and 0.3. Within this range, the logical operation time is very nearly a linear function of the compression ratio. When the operands of a logical operation are nearly incompressible, i.e., compression ratios > 0.3, the actual logical operation time is lower than the regression line. If the compression ratios are extremely small, say $< 10^{-4}$, the IO overhead again dominates the total time. Overall, both WAH and WBC use less than one-tenth the time needed by BBC in performing the same logical operation. This confirms the performance advantages observed on the STAR data. Between the two word-aligned schemes, this test shows that WAH is usually faster than WBC.

5. REFERENCES

[1] G. Antoshenkov and M. Ziauddin. Query processing and optimization in ORACLE RDB. *The VLDB Journal*, 5:229–237, 1996.

[2] T. Johnson. Performance measurements of compressed bitmap indices. In *VLDB'99*, 1999. Pages 278–289.

[3] A. Shoshani, L. M. Bernardo, H. Nordberg, D. Rotem, and A. Sim. Multidimensional indexing and query coordination for tertiary storage management. In *SSDBM'99*. Pages 214–225. 1999.

[4] K. Wu, E. J. Otoo, and A. Shoshani. Word-aligned compressed bitmaps. Technical Report LBNL-47807, Lawrence Berkeley National Laboratory, Berkeley, CA, 2001.

[5] K. Wu, E. J. Otoo, A. Shoshani, and H. Nordberg. Notes on design and implementation of compressed bit vectors. Technical Report LBNL/PUB-3161, Lawrence Berkeley National Laboratory, Berkeley, CA, 2001.