Data Compression for Bitmap Indexes

Y. Chen

Abstract

Compression Ratio (CR) and Logical Operation Time (LOT) are two major measures of the efficiency of bitmap indexing. Previous works by [5, 9, 10, 11] compare the performance of bitmap compression schemes conducted separately on logical operation time and compression ratio. This paper will describe these works and recommend for consideration a new matrix – overall efficiency indicator. The overall efficiency indicator is an integrative approach to evaluating the efficiency of bitmap compression schemes, taking both the compression ratio and the logical operation time into account. A further investigation is conducted to examine the effect of bitmap index density upon an overall efficiency indicator, thus shedding light on the selection of compression schemes depending on the bitmap index density. The finding shows that WAH is the most efficient compression scheme among gzip, BBC and WAH within a certain range of bitmap index density (0.0001 to 0.5). This finding is consistent with the proposal in previous studies of bitmap index compression in [9, 10, 11].

1.0 Introduction

Bitmap indexing appears to be an effective and efficient structure to help with processing complex queries like those in data warehousing applications, and have been implemented in several commercial DBMSs. A major advantage of bitmap indexing is that the main operations on the bitmaps during query processing are bitwise logical operations (AND, OR, NOT, XOR, etc.), which are very efficiently supported by hardware. Conventionally, bitmap indexes are space-efficient only for attributes with low cardinality. However, the size of bitmap indexes may become very large for high cardinality data attributes, thus resulting in heavy storage requirements. To address this problem, a number of bitmap compression schemes have been proposed.

Most of the data compression schemes, such as gzip, Expol, etc., are designed primarily to achieve good compression, without considering the CPU processing cost (time) associated with logical operations on compressed data. The result is much slower query processing time than for uncompressed bitmap indexes. Compression schemes were therefore viewed as sacrificing the advantages of bitmap indexing in query processing – namely, the low-cost bitwise operations and the capability of multiple indexes scans. For this reason, most commercial DBMSs do not use these compression schemes to compress bitmaps when implementing bitmap indexes. A number of schemes specifically designed to compress bitmap indexes have been proposed. One of the best-known schemes is Byte-aligned Bitmap Code (BBC) [2], which is used in ORACLE. BBC can perform logical operations very efficiently compared to the generic compression schemes while offering good compressions. However, in some cases, BBC is still much slower than the uncompressed bitmap indexes during query processing. A recently proposed compression scheme [5, 9, 10, 11], referred to as Word-aligned hybrid code (WAH), is receiving new attention because of its significant performance advantages over its predecessors. WAH claims a CPU-friendly scheme, and, in most cases, is faster than an uncompressed bitmap index during query processing while still using less space.

This paper analyzes and compares performance of the three compression schemes, with a focus on the efficiency of the bitmap indexing, depending on the density of bitmap indexes. The organization of the paper is as follows: In Section 2, a brief review of simple bitmap indexing is given as an introduction. Section 3 reviews three bitmap compression schemes selected as representatives by identifying their key features and their performance characteristics. We introduce the new overall efficiency indicator and discuss the relative performance of the three bitmap compression schemes using the overall efficiency indicator in Section 4. A short summary and discussion of future work are given in Section 5.

2.0 Simple Bitmap Indexing (LIT)

Simple bitmap indexing, referred to as literal (LIT) bit vector [12] or verbatim [5], is the first member of the bitmap indexing family, developed for database use in the Model 204 product from Computer Corporation of America [7]. It is a straightforward way of representing a bitmap. A bitmap index on an attribute is the collection of bitmap vectors. The basic idea behind simple bitmap indexing is to use a string of 1’s or 0’s bits to indicate whether an attribute in a tuple is equal to a specific value or not [7]. We can use the classic example, GENDER, to illustrate this. A simple bitmap index on an attribute GENDER, with domain {F, M}, consists of two bit vectors, one for F and the other for M. The position of a bit in the bit string denotes the position of a tuple in the table, and the length of the bit vectors is equal to cardinality of the table. The bits of the bit vector for M are set to 1 if the tuples of the corresponding bit-positions have the attribute GENDER = M, otherwise the bits are set to 0. The same principle applies to bit vector for F. Figure 1 shows a bitmap index for the attribute GENDER and STATE (STATE can have one of three values, NY, CT, CA). One advantage of bitmap indexes is that complex selection predicates can be computed very quickly, by performing bit-wise AND, OR, and NOT operations on the bitmap indexes. To process the example query “GENDER=F AND STATE=NY”, a bitmap index on attribute GENDER and a bitmap index on attribute STATE are used separately to generate two bitmaps representing objects satisfying the conditions on GENDER and STATE. The final answer is the result of a bitwise logical AND operation on these two bitmaps.

The low-cost of bitwise operations in bitmap index processing has led to considerable interest in their use in Decision Support Systems (DSS). However, within high cardinality domains, both the time and space complexity of building and maintaining the simple bitmap index will rapidly become higher, and the sparse of the bitmap vectors increases with high cardinality attributes, resulting in
poor space utilization and high processing cost. Bitmap compression techniques can make bitmap indexes more compact representations of sparse and high cardinality data attributes.

3.0 Review of Bitmap Index Compression Algorithms

A variety of techniques for compressing bitmap indexes can be found in a search of literature. In this paper, three major bitmap compression schemes are selected as representatives, from a number of schemes studied previously [2, 5, 9]: Gzip, Byte-aligned Bitmap Code (BBC), and word-aligned hybrid run-length code (WAH). This section is a brief review of the schemes that identifies their key features and their performance characteristics.

3.1 Gzip

Gzip is one of the generic compression schemes [4]. It is an implementation of Lempel-Ziv (LZ) encoding, which is designed primarily for data file compression rather than for bitmap compression. LZ encoding compresses repeated sequences of symbols and replaces them with short compression codes.

The Gzip scheme supports only the Basic Boolean Operation Algorithm [5]. In the Basic Boolean Operation Algorithm, the bitmap is first uncompressed, and the bitwise operations between the two bitmaps are performed. The time to decompress the bitmaps occupies the most time during query processing. We chose the gzip as representative of general purpose compression schemes because it is reported to be usually faster than others in decompressing the data files.

3.2 Byte-aligned Bitmap Code (BBC)

The second type of compression scheme is specialized bitmap compression schemes. Most of these compression schemes are byte-based, that is, they access computer memory one byte at a time. The Byte-aligned Bitmap Code (BBC) is one of the well-known schemes of this type. It was first proposed by Antoshenkov in [2, 3]. The claimed advantage of the BBC encoding algorithm is efficient bitwise logical operations, since all operations occur locally on full bytes. In addition, it achieves compression almost as good as gzip. Antoshenkov proposed logical operations on compressed bitmaps, which can be significantly faster than operating on uncompressed bitmaps. The BBC scheme can use the Direct Boolean Operation Algorithm, performing logical operations directly on compressed bitmaps [5].

Most specialized bitmap compression schemes, including the BBC scheme, generally try to store consecutive identical bits in compact forms. Run-length encoding (RLE) is among the simplest of these strategies. Run-length encoding represents the successive identical bits (also called a fill or a gap) by their bit value and their length. The bit value of a fill is called the fill bit. The gap can be zero-filled only for one-sided BBC codes, and two-sided BBC codes can have zero-filled or one-filled.

There are two properties of BBC encoding that are crucial to the efficiency of the BBC scheme. The first property is the byte alignment. This property limits a fill length to be an integer multiple of bytes, and a tail byte is never broken into bits during bitwise logical operation. On most CPUs, working on a whole byte is much more efficient than working on individual bits. Removing the byte alignment might lead to better compression, but the bitwise logical operations are much slower.

3.3 Word-aligned Hybrid Scheme (WAH)

The Word-aligned Hybrid Scheme is the newest scheme. On modern computers, accessing one byte usually takes as much time as accessing one word [8]. Word-based compression schemes can take advantage of this fact. The most efficient word-based compression scheme is proposed by Kesheng and his colleagues [9, 10, 11, 12], and is called the word-aligned hybrid (WAH) code. The WAH scheme achieves performance in setting that all logical operations are performed on entire words, rather than on bytes or bits. Thus, they can perform bitwise logical operations faster. This section contains brief descriptions of this compression scheme.

The WAH code is based on hybrid run-length code (HRL), which represents long fills using run-length encoding and represents short fills literally. In addition to representing all fills and literal bits in whole words, WAH imposes another requirement on the fills. The word alignment requirement demands that all fill lengths be integer multiples of words. This is similar to Byte-alignment requirement in BBC code, except that BBC stores

<table>
<thead>
<tr>
<th>...</th>
<th>RID</th>
<th>GENDER</th>
<th>STATE</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>F</td>
<td>NY</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>M</td>
<td>NY</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>M</td>
<td>CA</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>F</td>
<td>CT</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>F</td>
<td>CA</td>
<td></td>
</tr>
</tbody>
</table>

Table 1. Simple Bitmap Indexes for GENDER and STATE

<table>
<thead>
<tr>
<th>Bp</th>
<th>BM</th>
<th>Bny</th>
<th>Bca</th>
<th>Bct</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 1. Simple Bitmap Indexes for GENDER and STATE
compressed data in bytes and WAH stores compressed data in words.

There are two types of words in WAH codes: literal words and fill words. A literal word represents a bitmap literally and a fill word represents a fill. The fill represents the fill length as a number of words. The logical operation can be performed on the compressed bitmaps and the time needed for such an operation on two bitmaps is related to the sizes of the compressed bitmaps.

Kesheng and his colleagues came to the conclusion that the BBC scheme achieves better compression than the WAH scheme, but bitwise logical operations on BBC are slower than on WAH codes \[11, 12\]. This is mainly due to the following three reasons.

1. The types of runs: In WAH codes, there are only two types of words, while there are four different types of bytes in BBC codes. It takes more time to determine the type of runs in BBC. After run type is decided, the run may still need to be fully decoded to determine the fill length of the tail value, since BBC codes use a clever packing to minimize space use.

2. WAH is designed to obey word-alignment requirements, while BBC is designed to obey byte-alignment requirements. The requirements ensure that WAH always accesses whole words and BBC always accesses whole bytes during logical operations. It takes more time for BBC to load the data from the main memory to CPU registers than for WAH.

3. The way to encode the short fills is different for the WAH and BBC schemes. The BBC scheme encodes shorter fills more compactly than WAH. WAH typically stores the short fill as a literal word, while BBC starts a new run when it encounters a short fill. It is faster to operate on a WAH literal word than on a BBC run, even though WAH cannot compress as well as BBC.

4.0 Performance Analysis

Previous study has investigated the relationship between compression ratio and bitmap index density and the relationship between logical operation time and bitmap index density separately. We will describe these works and recommend for consideration a new matrix – overall efficiency indicator. The overall efficiency indicator is an integrative approach to evaluating the efficiency of bitmap compression schemes, taking both the compression ratio and the logical operation time into account. A further investigation is conducted to examine the effect of bitmap index density upon an overall efficiency indicator, thus shedding light on the selection of compression schemes depending on the bitmap index density.

4.1 Performance Comparison Based on Compression Ratio \[5, 12\]

Figure 2 illustrates the relationship between compression ratio and bitmap index density. Compression ratio is defined as the ratio of the compressed size divided by the uncompressed (original) size. Across the three compression schemes (gzip, BBC and WAH) as the bitmap index density increases, the compression ratio increases steadily. When the bitmap index density approaches 0.5, the compression ratios of the three compression schemes clusters around 1, which is the uncompressible state.

4.2 Performance Comparison Based on Logical Operation Time

Some studies have used the OR query to investigate the relationship between logical operation time and bitmap index density \[5, 12\]. Figure 3 illustrates an interesting trend. As bitmap index density increases, the logical operation time for gzip, BBC and WAH increases until the bitmap index density reaches around 0.2. Then, the logical operation time starts to decline. When density is between 0.0001 and 0.5, WAH takes the least logical operation time. When density is between 0.0001 and 0.01, BBC uses less logical operation time than gzip, yet between 0.01 and 0.5, gzip uses less logical operation time than BBC.

4.3 The Relationship between Compression Ratio and Logical Operation Time

In Kesheng’s study \[12\], plots of the relationship between data compression ratio and logical operation time imply a linear relationship within a certain range of bitmap index density. In general, the higher the compression ratio, the longer the logical operation time it takes to process the operation. Therefore, it is possible that the logical operation
time becomes a function of compression ratio. The illustration of the linear relationship between the logical operation time and compression ratio at a certain level of bitmap index density can be found in [12].

### 4.4 Overall Efficiency Indicator

Although logical operation time and compression ratio are two related measures, using them separately to evaluate compression schemes does not result in a rationale for evaluating overall performance. To overcome this problem, we can use the slope ratio (LOT/CR) as an indicator of the overall efficiency for each compression scheme, assuming a linear relationship between logical operation time and compression ratio. At a certain level of bitmap index density, one unit increase in compression ratio results in different degrees of change in logical operation time for different compression schemes. The shorter the increase of logical operation time, the shorter time it takes to complete the operation, the more efficient is the compression scheme.

The slope ratio (LOT/CR) takes both the logical operation time and compression ratio into account and provides a reasonable way to evaluate the overall efficiency of a compression scheme. At a certain level of bitmap index density, it is possible to compute a slope ratio (LOT/CR) between logical operation time and compression ratio for a compression scheme. It is possible that over a certain range of bitmap index density, the slope ratios of a compression scheme can be plotted. The smaller the ratio, the more efficient is the compression scheme. Thus, it becomes relatively easy to compare the efficiencies of the several compression schemes at different bitmap index densities. The rationale of choosing an optimal scheme becomes clear. We generate the following formula for overall efficiency indicator:

\[
\text{Overall indicator} = \frac{\text{Logical Operation Time}}{\text{Compression Ratio}}
\]

The overall indicator for WAH is roughly around 1 between bitmap index density of 0.0001 and 0.01 and begins to become more efficient as the bitmap index density increases from 0.01. The overall indicator for BBC is relatively stable across the bitmap index density range of 0.0001 and 0.5. The overall indicator for gzip shows the least efficiency at low bitmap index density. As the bitmap index density increases, the overall indicator for gzip surpasses the one for BBC and becomes a better choice after the bitmap index density of 0.01.

As the results indicate, the choice of the compression scheme depends on the density of the bitmap to be compressed and operated on. It also depends on what is the most important to a specific application, logical operation time, or space requirement, or the overall efficiency. Table 1 lists the results of the performance comparison of the three compression schemes.

<table>
<thead>
<tr>
<th>Bit Density</th>
<th>Compression Ratio</th>
<th>Logical Operation Time</th>
<th>Overall Indicator</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0001 - 0.2</td>
<td>BBC, gzip</td>
<td>WAH</td>
<td>WAH</td>
</tr>
<tr>
<td>0.2 - 0.5</td>
<td>WAH, BBC</td>
<td>WAH</td>
<td>WAH</td>
</tr>
</tbody>
</table>

Table 1. Performance comparison for three compression schemes

### 4.5 Limitation

The above analytic approach integrates both logical operation time and compression ratio in evaluating the overall efficiency of compression schemes. Although logical operation time and compression ratio seem to be two of the major factors, other factors might also play significant roles in the overall efficiency, such as CPU capacity, IO speed and other query types. An attempt without fully addressing the effects of these factors is still in a preliminary stage, although it provides an integrative approach in analyzing compression scheme.

Within a certain range of bitmap index density, the relationship between compression ratio and logical operation time is assumed to be linear, which provides the possibility of using the slope ratio as a justification indicator to evaluate performance. Yet for a full range of bitmap index density, the relationship between compression ratio and logical operation time is relatively unexplored. Therefore, the overall indicator only applies to a certain range of bitmap index density.

### 5.0 Summary & Future Work

Bitmap index compression has received new attention recently because of the emergence of a new compression scheme, WAH. WAH claims significant performance advantages over its predecessors. In this paper, a review of three representative bitmap compression schemes is given. Then we create an overall efficiency indicator by integrating both compression ratio and logical operation time. Based on the overall efficiency indicator, we evaluate the comparative performance of the three compression schemes in various levels of bitmap index density. The result shows that WAH
Indeed is the most efficient among the three compression schemes within a certain range of bitmap index density.

Bitmap compression can reduce space usage and possibly Boolean operation time. However, bitmap compression introduces new complications to bitmap indexing design. Much research work needs to be done in bitmap index design, taking account into compression. For example:

Taking into account the overall efficiency indicator in selection of bitmap index compression scheme, depending on the bitmap density level

A linear regression relationship exists between compression ratio and logical operation time in a specific range of bitmap index density. It is also likely that a non-linear relationship might occur outside that range. Further investigation is called for to examine the full relationship.

6.0 References


