Information Systems 62 (2016) 155–168

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/infosys

SBH: Super byte-aligned hybrid bitmap compression

Sangchul Kim, Junhee Lee, Srinivasa Rao Satti *, Bongki Moon *

Department of Computer Science and Engineering, Seoul National University, Seoul 08826, Republic of Korea

ARTICLE INFO

Article history:
Received 9 February 2016
Received in revised form
24 June 2016
Accepted 8 July 2016
Recommended by: D. Shasha
Available online 16 July 2016

Keywords:
Database Indexing
Bitmap Index
Bitmap compression
Byte-based Bitmap Code
Word-Aligned Hybrid

ABSTRACT

Bitmap indexes are commonly used in data warehousing applications such as on-line analytic processing (OLAP). Storing the bitmaps in compressed form has been shown to be effective not only for low cardinality attributes, as conventional wisdom would suggest, but also for high cardinality attributes. Compressed bitmap indexes, such as *Byte-aligned Bitmap Compression* (BBC), *Word-Aligned Hybrid* (WAH) and several of their variants have been shown to be efficient in terms of both time and space, compared to traditional database indexes. In this paper, we propose a new technique for compressed bitmap indexing, called *Super Byte-aligned Hybrid* (SBH) bitmap compression, which improves upon the current state-of-the-art compression schemes. In our empirical evaluation, the query processing time of SBH was about five times faster than that of WAH, while the size of its compressed bitmap indexes was retained nearly close to that of BBC.

1. Introduction

Many enterprise applications generate a massive volume of data through logging transactions or collecting sensed measurements, which brings up the need for effective indexing methods for efficient storage and retrieval of data. To support some of the retrieval queries efficiently, database management systems make use of indexes such as B-trees and bitmap indexes [14,18]. The index schemes based on B-tree are efficient for a wide variety of data, since they support both searches and updates with nearly the same complexity. However, for most data warehousing applications, search operations are more frequent than updates. For such applications, bitmap indexes may improve the overall performance. Besides, when the number of unique values of an attribute is small (e.g., gender), one can achieve much better performance than B-trees by using bitmap indexes.

A bitmap index is a collection of bit vectors created for each distinct value in the indexed column. The number of bit vectors in a bitmap index for a given indexed column, also referred to as *cardinality*, is equal to the number of distinct values that appear in the indexed column. For a column with cardinality m, the ith bit vector corresponds to the ith distinct value (in some order) and the jth bit in the ith bit vector is set to 1 if and only if the value of the jth element in the column is equal to i. Thus, if an indexed column of n elements has cardinality m, then its bitmap index contains m bit vectors of length n each, and hence uses a total of mn bits.

Fig. 1 shows a set of bitmaps for an attribute `City`. The cardinality of `City` is four. Each bitmap represents whether the value of `City` is one of the four available values. Suppose we process an SQL query below.

```
SELECT*
FROM T
WHERE City = Seoul OR City = London
```

The main operation performed in the bitmaps is a bitwise operation $B_{Seoul} \vee B_{London}$, where B_c is a bitmap corresponding to city c. Standard selection queries on bitmap indexes can be

* Corresponding authors.
E-mail addresses: stdio@snu.ac.kr (S. Kim), jvl@tcs.snu.ac.kr (J. Lee), ssrao@snu.ac.kr (S.R. Satti), bkmoon@snu.ac.kr (B. Moon).

http://dx.doi.org/10.1016/j.is.2016.07.004
0306-4379/

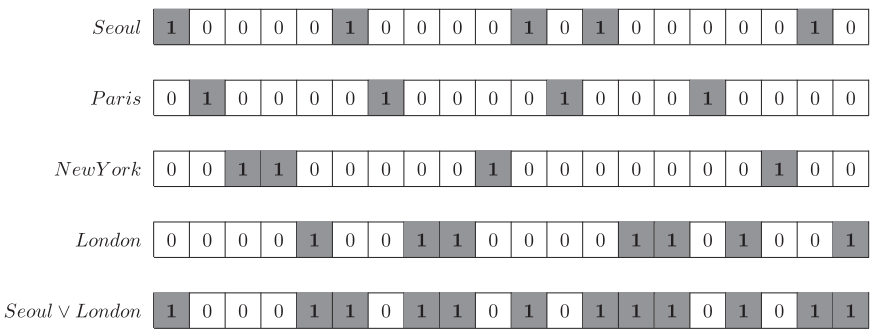

Fig. 1. A set of bitmaps.

supported by performing bitwise operations on the bit vectors. Since modern computers are optimized to perform bitwise operations, dealing with bitmap indexes is easy and efficient [4,16].

One of the main drawbacks of bitmap indexes is that their space usage is high when compared to the raw data or a B-tree index, especially when the cardinality is high. To overcome this, compressed bitmap indexes have been proposed. The simplest way to compress a bitmap index is to compress each individual bit vector using a standard text compression algorithm, such as LZ77 [27]. This improves the space usage significantly, but performing logical operations on LZ77-compressed bit strings is usually much slower than uncompressed bit strings, because compressed bitmaps have to be fully decompressed.

As a result, compression schemes that apply some form of Run-Length Encoding (RLE) are suggested to be used for compressing bitmaps. RLE is a fundamentally lossless compression scheme that encodes a string by splitting it into sequences of runs, and then encoding the runs efficiently. Since a typical bitmap index used in database applications is assumed to have sparsely distributed set bits (i.e., most bits are set to zero), forming long sequences of zeros, RLE-based schemes are expected to achieve good compression and are commonly used to compress bitmaps.

Two of the most popular compression schemes, based on run-length encoding, are Byte-aligned Bitmap Code (BBC) [1,2] and Word-Aligned Hybrid (WAH) bitmap compression [5]. Both the schemes divide the original bit vectors to be compressed into blocks of a specific unit size. The main difference between these two schemes is that BBC uses a unit size of 8 bits, while WAH uses 32 bits as unit size. In terms of performance, BBC typically consumes less space, while WAH supports faster query processing.

In this paper, we propose an improved version of BBC, called Super Byte-aligned Hybrid (SBH) bitmap compression. It uses a new feature that enhances time performance in processing logical operations so that its byte-aligned encoding can lead to better compressibility in comparison with a word-aligned scheme. Specifically, the superiority of our scheme in compressibility becomes more pronounced when the cardinality of an indexed column grows larger than 50. The query processing time of SBH was about five times faster than that of WAH, while the size of compressed bitmap indexes was retained nearly close to that of BBC. Thus, our scheme improves upon both these schemes.

The rest of the paper is organized as follows. We first describe other bitmap compression schemes that were proposed in literature and how they were implemented in Section 2. In Section 3, we describe some algorithmic and implementation details of our new scheme. Section 4 discusses experimental results and comparison with the existing schemes. Finally, Section 5 concludes the paper.

2. Related work

Several compression schemes based on run-length encoding have been proposed in the literature. The main merit of these schemes is that logical operations could be done without decompressing the whole bitmaps. One of the earliest such schemes that have been successful is the Byte-aligned Bitmap Code [2], or BBC for short. BBC is very effective in compressing bit sequences, and in particular for representing bitmap indexes. To improve runtime performance of BBC, a word-based bitmap compression scheme, called the Word-Aligned Hybrid (WAH) scheme has been introduced [24], which takes advantage of the word-level bitwise operations. Subsequently, many other compressed bitmap indexing schemes have been proposed, such as EWAH [15], PLWAH [9], PWAH [22], COMPAX [10], VAL-WAH [12], RLH [20], UCB [3], PLWAH+ [6], and SECOMPAX [23]. Some of the schemes are discussed in Chen et al. [7]. Major schemes have been mathematically analyzed by Guzun and Canahuate [11] and Wu et al. [26]. Most of these schemes are variants of WAH, and perform better than WAH in terms of either time or space or sometimes both. They also typically achieve better time performance than BBC, but not in terms of space.

2.1. Byte-aligned bitmap code (BBC)

Byte-aligned Bitmap Code (BBC) [2] is a byte-based compression scheme that encodes (compresses) a sequence of bytes in the uncompressed bit vector by a sequence of one or more bytes. More specifically, a *run* of bits (i.e., sequence of bits having the same value) followed by a small number of bytes are “compressed” into a *header byte*, optionally followed by one or more *counter bytes* followed by zero or more *literal bytes*, as explained below.