CODIS: A New Compression Scheme for Bitmap Indexes

Wenxun Zheng  
Department of Automation  
Tsinghua University  
Beijing, China  
zhengwx15@mails.tsinghua.edu.cn

Yin Liu  
Department of Automation  
Tsinghua University  
Beijing, China  
liuyin14@mails.tsinghua.edu.cn

Zhen Chen  
Fundamental Training Center for Industry  
Tsinghua University  
Beijing, China  
zhchen@tsinghua.edu.cn

Junwei Cao  
Research Institute of Information Technology  
Tsinghua University  
Beijing, China  
jcao@tsinghua.edu.cn

ABSTRACT
Bitmap indexing is a promising approach for indexing. However, the huge space consumption hinders the wide adoption of bitmap indexing, especially in memory-critical area such as packet classification. To this end, a variety of compression scheme are proposed to reduce the space consumption and simultaneously maintain the fast calculation which is afoecused feature of bitmap indexing. In this paper, a novel compression scheme, named Compressing Dirty Snippet (CODIS), is proposed. It is based on the Word-Aligned Hybrid (WAH) algorithm. The basic idea is to trade some payload for flexibilidady so that the probability of space reduction is raised, which achieves better compression. CODIS is verified by experiments on part of the network traffic data from CAIDA 2016. The results show that, comparing to WAH, CODIS increases the time for intersection at a rate of about 7% while reduces 39% of the space consumption. Comparing to COMPAX, it takes 11% more space but reduces 19% of time for intersection. Comparing to PLWAH, it is better in both space consumption and inter-bitmap operations. CODIS takes the least time to decode bitmap indexes into integers.

CCS Concepts
• Information systems→Data Structures→Data Compression  
• Information Retrieval→Search engine architectures and scalability→Search engine indexing

Keywords
Bitmap index; network traffic; data compression; indexing

1. INTRODUCTION
Bitmap indexing is becoming more and more popular in databases or data stores that support large scale data or real-time stream data, such as Druid [16], FASTBIT [15] and Apache Spark. Varies compression algorithms contribute to current development of bitmap index technology. The main concerns of bitmap index compression algorithm are space consumption and online-operable features that allows executing inter-bitmap operations without decompression.

1.1 Bitmap Indexes
Bitmap indexing could be regarded as a method for the storage of posting list in inverted index, while it is also generalized beyond inverted index. For convenience, we will introduce bitmap indexing in the context of inverted index.

Posting list is a list of integers indicating the locations of records which match the rule of current entry. For example, there could be posting list {0, 7} indicating that the 0th and 7th records are related to http request.

Verbatim bitmap index stores this list as bit string ‘1000 0001’, that is, each ‘1’ bit represents a record. Practical operations such as intersection and join are done by bit-wise AND/OR operation between bitmap indexes. However verbatim bitmap index usually takes up enormous space and demands compression.

2. Background
The first use of verbatim bitmap indexing in commercial database is by P. O’Neil [1] with his database system Model 204. Since then, varies of compression schemes are proposed to improve the performance of bitmap indexing.

Byte-aligned Bitmap Compression (BBC) [2] and Word-aligned Hybrid are based on run-length encoding [3]. The occupancy rate of free bit in WAH is not satisfying, so that Position List WAH (PLWAH) [4] and Compressed Adaptive index (COMPAX) [5] are proposed, which both apply further compression based on the result of WAH. Improvements on the path of WAH include Compressed ‘n’ Composable Integer Set (CONCISE) [6], Scope-Extended COMPAX (SECOMPAX) [7], PLWAH algorithm for sorted data (SPLWAH) [8], Combining Binary And Ternary encoding (COMBAT) [9]. Non-WAH-based algorithms include Maximized Stride with Carrier (MASC) [10], Draggled Fills WAH (DFWAH) [11] and Enhanced WAH (EWAH) [12].

Some researchers build up a framework that takes use of different algorithms to optimize the performances. Such as Variable Length Compression (VLC) [13] and Roaring Bitmap [14].

WAH algorithm is implemented in FASTBIT [15]. CONCISE algorithm is applied in Druid. Roaring Bitmap is applied in the Apache Spark project.

3. CODIS
Compression in CODIS is considered as providing encoding scheme to represent bit string in bitmap index with less bits while keep efficiency of inter-bitmap operations. The result of CODIS consists of 32-bit code words. Types of code word are identified by several most significant bits in code word. All types are listed in Fig. 1. Note that ‘32 bits’ are just for clari ty and alignment.

A literal word is headed with ‘1’. It is decoded into 31 bits in verbatim bitmap. Literal word does not compress the data but increases the size by adding a tag ahead.

Other words are headed with ‘0’ and another four bits. One of the

<table>
<thead>
<tr>
<th>32 bits</th>
<th>literal</th>
<th>zero-filled</th>
<th>one-filled</th>
<th>carried</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000 0001</td>
<td></td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
</tr>
</tbody>
</table>

Figure 1. Header of Different Code Words

four is called tag and the other three are used for recording an integer, length.

If length is 0, this word is a filled word. The remaining 27 bits are considered an integer, counter. A filled word is decoded into...
The second part of decoding result is the decoding result of this word bits. Then the groups are scanned sequentially and encoded during the encoding process starts from grouping bitmap indexes by 31. It is obvious that the decoding result is aligned to 31 bits. Hence in greedy paradigm.

Filled word carries word, first take filled word into consideration. The decoding result is aligned to 31 bits. Then the groups are scanned sequentially and encodes them in greedy paradigm.

4. Results
We implemented WAH, PLWAH, COMAX and CODIS algorithm for experiments. The code is written in C++ compiled in Release mode by Visual Studio 2012 under Windows 8. The dataset is network traffic from CAIDA 2016, containing about 13 million headers of IP packets. Main concerns here are the size of bitmap indexes and time for intersection/union operation between bitmap indexes.

Only source and destination IP addresses are used. IP addresses are divided into 8 bytes and indexed separately. Each byte generates 256 bitmap indexes, which results in 2048 bitmap indexes totally. The total size, encoding time, decoding time of all bitmap indexes is shown in Tab. 1.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>WAH</th>
<th>PLWAH</th>
<th>COMAX</th>
<th>CODIS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>88.7MB</td>
<td>61.2MB</td>
<td>48.2MB</td>
<td>53.4MB</td>
</tr>
<tr>
<td>Encoding</td>
<td>6.9s</td>
<td>7.2s</td>
<td>8.7s</td>
<td>8.3s</td>
</tr>
<tr>
<td>Decoding</td>
<td>1.9s</td>
<td>2.0s</td>
<td>2.1s</td>
<td>1.7s</td>
</tr>
</tbody>
</table>

When doing intersection/union, two bitmap indexes are randomly picked out for further intersection/union operation. Time for single operation is averaged over ten thousand of operations, which are shown in Fig. 2.

4. References