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
zhenchen@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 a focused 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 flexibility 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 and Roaring Bitmap are applied in Druid and the Apache Spark respectively.

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 clarity 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

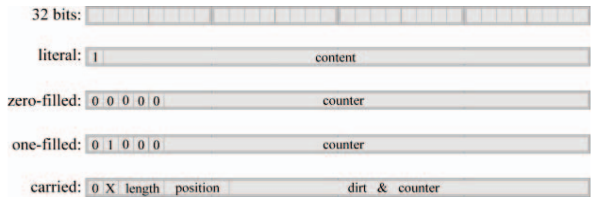

Figure 1. Header of Different Code Words

four is called *flag* 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

*31*counter* ‘0’ bits. However if the flag in *filled word* is ‘1’, the result of decoding should be flipped.

If length is larger than 0, this word is a *carried word*. Another four bits are divided out as integer, *position*. In *carried word,* the number of bits in *dirt* area is equal to *3*length*. To decode a *carried word*, first take *dirt* out and put it into a bit string consists of 31 ‘0’ where the number of ‘0’ at the right of *dirt* should be equal to *2*position*. Such 31-bit string is the first part of the decoding result. Then the *position, length* and *dirt* in the *carried word* are all set to all ‘0’, which turns *carried word* into *filled word.* The second part of decoding result is the decoding result of this *filled word*.

It is obvious that the decoding result is aligned to 31 bits. Hence the encoding process starts from grouping bitmap indexes by 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 1. Performance of encoding and decoding

Algorithm	WAH	PLWAH	COMPAX	CODIS
Size	88.7MB	61.2MB	48.2MB	53.4MB
Encoding	6.9s	7.2s	8.7s	8.3s
Decoding	1.9s	2.0s	2.1s	1.7s

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

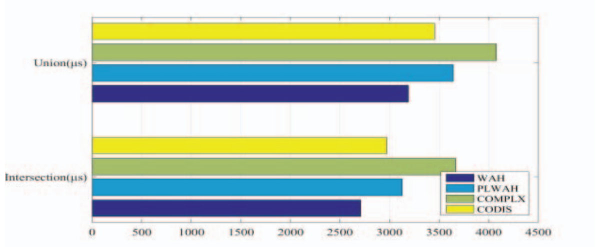

Figure 2. Time for Intersection/Union Operation

Results show that comparing to COMAPX, CODIS sacrifices 11% more space consumption but reduces the inter-bitmap operation time by 19%. Comparing to WAH, CODIS reduces 39% of space consumption while increases 7% time for inter-bitmap operations.

CODIS and COMPAX are both compression based on WAH. CODIS tries to combine two adjacent words into one (*carried word*), but COMPAX tries to combine three, which is more complex. This difference explains why CODIS is faster than COMPAX when conducting inter-bitmap operations and decoding, and also why CODIS takes more space than COMPAX. However, the flexibility of CODIS, that is, the use of *length* and *position*, helps to reduce the space consumption of CODIS.

PLWAH also tries to combine two adjacent words. Its strategy is absorbing *literal* with only several set bits. PLWAH stores the position of individual bit rather than the verbatim content of *literal*, which need additional conversion (e.g. from {0, 7} to ‘10000000’) when conducting inter-bitmap operations and leads to slower operation.

5. REFERENCES

[1] O'Neil, Patrick E. "Model 204 architecture and performance." High Performance Transaction Systems. Springer Berlin Heidelberg, 1989. 39-59.

[2] Antoshenkov, G. "Byte-aligned bitmap compression". Data Compression Conference, 1995.

[3] K. Wu, et al. "Optimizing bitmap indices with efficient compression". ACM Transactions on Database Systems, 31(1):1–38, 2006.

[4] Deli, et al. "Position list word aligned hybrid: optimizing space and performance for compressed bitmaps". In EDBT 2010, Lausanne, Switzerland, March 22-26, 2010, Proceedings, pages 228–239, 2010.

[5] F. Fusco, et al. "Net-fli: On-the-fly compression, archiving and indexing of streaming network traffic". Proceedings of the VLDB Endowment, 3(2):1382–1393, 2010.

[6] A. Colantonio and R. D. Pietro. "CONCISE: Compresssed ’n’ composable integer set". Inform.process.lett, (16):644–650, 2010.

[7] Y. Wen, et al. "SECOMPAX: A bitmap index compression algorithm". In International Conference on Computer Communication and Networks, pages 1–7, 2014.

[8] J. Chang, et al. "SPLWAH: A bitmap index compression scheme for searching in archival internet traffic". In IEEE International Conference on Communications, 2015.

[9] Y. Wu, et al. "Combat: A new bitmap index coding algorithm for big data". Journal of Tsinghua University(Science and Technology), 21(2):136–145, 2016.

[10] Y. Wen, et al. "MASC: A bitmap index coding algorithm for internet traffic retrieval". In IEEE International Conference on Communications, 2016.

[11] Schmidt, et al. "DFWAH: A Proposal of a New Compression Scheme of Medium-Sparse Bitmaps." DBKDA 2011.

[12] D. Lemire, et al. "Sorting improves word-aligned bitmap indexes". Data and Knowledge Engineering, 69(1):3–28, 2009.

[13] F. Corrales, et al. "Variable Length Compression for Bitmap Indices". Springer Berlin Heidelberg, 2011.

[14] S. Chambi, et al. "Optimizing druid with roaring bitmaps". In International Database Engineering and Applications Symposium, 2016.

[15] K. Wu, et al. "Fastbit: interactively searching massive data". In Journal of Physics Conference Series, page 012053, 2009.

[16] Yang, Fangjin, et al. "Druid: a real-time analytical data store." Acm Sigmod International Conference on Management of Data 2014