TSINGHUA SCIENCE AND TECHNOLOGY
ISSN 1007-0214 0?/?? pp???-???
Volume 18, Number 3, June 2013

COMBAT: A new bitmap index coding algorithm for Big Data

Yinjun Wu, Zhen Chen ∗, Yuhao Wen, Wenxun Zheng, Junwei Cao

Abstract: Bitmap indexing has been widely used in various applications due to its speed in bitwise operations. However, it can consume large amounts of memory. To solve this problem, various bitmap coding algorithms have been proposed. In this paper, we present COMBAT *(COMbining Binary And Ternary encoding)*, a new bitmap index coding algorithm. Typical algorithms derived from WAH are COMPAX *(COMPressed Adaptive indeX)* and CONCISE *(Compressed 'n' Composable Integer Set)*, which can combine either two or three continuous words after WAH encoding. COMBAT combines both mechanisms and results in more compact bitmap indexes. Moreover, querying time of COMBAT can be faster than that of COMPAX and CONCISE, since bitmap indexes are smaller and it would take less time to be loaded into memory. To prove the advantages of COMBAT, we extend a theoretical analysis model proposed by our group, which is composed of the analysis of various possible bitmap indexes. Some experimental results based on real data are also provided, which show COMBAT's storage and speed superiority. Our results demonstrate the advantages of COMBAT and codeword statistics are provided to solidify the proof.

Key words: bitmap index; Big Data; COMBAT; CONCISE; COMPAX; index encoding; performance evaluation

1 Introduction

The boom in streaming data, such as IoT (Internet of Things) sensing data, network traffic, and machine operational logs, requires powerful data processing systems, necessitating more efficient data structures and algorithms than before. Many applications are experiencing challenges in querying and searching such Big Data. For example, as the biggest 3G wireless networks operator, China Unicom is experiencing difficulties in processing queries in their CDR (call detail records) billing data and returning real-time results to mobile users. Although advancements in CPUs and other hardware devices have relieved the pressure of real-time query demands to some degree, this problem is far from being fully resolved. Moreover, CDR data from 3G and 4G wireless networks are increasing dramatically [1] and as a result, it would take several days for users to access their usage information. So many telecom operators have already paid much attention to accelerating traffic data queries[2-3].

Bitmap indexing [4-8] has been widely used to solve problem of quick response to queries for traffic data,

• Yinjun Wu is studying in the Department of Automation and Tsinghua National Laboratory for Information Science and Technology (TNList), Tsinghua University, Beijing 100084, China.
E-mail: wu-yy12@mails.tsinghua.edu.cn
• Zhen Chen is working in iCenter of Tsinghua University, Beijing 100084, China.
E-mail: zhenchen@tsinghua.edu.cn
• Yuhao Wen is studying in the Department of Computer Science, Duke University,NC 27708 USA.
E-mail: filippo_wen@126.com
• Wenxun Zheng is studying in the Department of Automation and Tsinghua National Laboratory for Information Science and Technology (TNList), Tsinghua University, Beijing 100084, China.
E-mail: 292386890@qq.com
• Junwei Cao are working in the Research Institute of Information Technology and Tsinghua National Laboratory for Information Science and Technology (TNList), Tsinghua University, Beijing 100084, China.
E-mail: jcao@tsinghua.edu.cn
∗ To whom correspondence should be addressed.
Manuscript received: year-month-day; revised: year-month-day; accepted: year-month-day

such as netflow data and CDR data. A bitmap index [9] is a bit sequence that represents a specified property and indicates which data items in the dataset satisfy this property. The bit sequence has a 1 in position i if the i-th data item satisfies the property, and 0 otherwise. Queries are executed using fast bitwise logical operations and binary vectors supported by hardware. Bitmap indexing has more flexible encoding schemes and takes less time to answer the query than other indexing schemes[10]. However, since bitmap indexing can consume a large amount of memory and disk space, powerful query systems and corresponding algorithms are indispensable when processing.

Many kinds of systems have been devised and implemented to deal with such challenges. For example, Druid [11], an emerging real-time data analytics system, has satisfying performance in storing and querying real-time streaming data. Moreover, to deal with the storage requirements, a series of bitmap index coding algorithms have been proposed, such as BBC[13], CONCISE [12], WAH [14-15], UCB [16], RLH [17], PLWAH [18], EWAH [19], PWAH [20] , COMPAX [21], GPU-WAH [22-23], GPU-PLWAH [24], VLC [11], SECOMPAX [25], PLWAH+[26], DFWAH [27] and Roaring bitmap[28]. A detailed survey of these diverse coding schemes is given in [29]. We deemed CONCISE(*Compressed 'n' Composable Integer Set*) and COMPAX (COMPressed Adaptive indeX) better than others for our purposes because they have relatively simple coding schemes.

In the following sections, a new coding algorithm called COMBAT *(COMbining Binary And Ternary encoding)* is introduced, which includes more coding schemes than COMPAX and CONCISE. In the following sections, details about COMBAT encoding are shown and a mathematical proof is provided to demonstrate its advantage. This proof includes memory consumption analyses for two kinds of bitmap indexes-sparse bitmaps and dense bitmaps in uniform distribution. Besides theoretical analysis, experiments were also conducted to compare COMBAT with CONCISE and COMPAX. Two real datasets were used for representing two kinds of bitmap indexes separately. The results show that COMBAT performs better in terms of storage demands and querying time than COMPAX and CONCISE.

2 THE STATE OF THE ART

2.1 Basic Definitions

A bitmap index is composed of a large number of bit sequences. An operation unit in bitmap index can be a byte, a word, a Dword, or a qword, to suit different CPUs. In this paper, all operations are word-based.

Here are some definitions. In a bit sequence, a bit set to zero or one is called a *unset bit* or *set bit*. A group of 31 continuous bits is defined as a *chunk* and a raw bit sequence will be divided by *chunk*. If the bits in a *chunk* are different, then this *chunk* will be defined as *literal*. If 31 bits in a *chunk* are all *set bit*s or *unset bit*s, then it is called *fill*. A *fill* is called a 0-*fill* or a 1-*fill* according to whether all its bits are *unset bit*s or *unset bit*s.

2.2 WAH

WAH(*Word Aligned Hybrid*) is a classic bitmap index coding algorithm, which uses a word to contain a *chunk* in memory by adding a most significant bit (MSB). In WAH, this bit is used to distinguish *fill* and *literal*. If a *chunk* is *literal*, then the MSB is a *set bit*. An unset-bit MSB is used to denote a *fill*. The type of *fill* is denoted in the bit next to the MSB, and the remaining bits are used to store the number of consecutive *fill*s of the same type. The merged word is still called a *fill* in WAH.

2.3 CONCISE

CONCISE [12] is used to represent a series of integers in a set; it is used as further compression after WAH encoding. It is proposed for handling sparse bitmaps. In order to simplify our analysis in the following sections, some concepts in CONCISE are redefined here. CONCISE introduces a new type of codeword based on *fill*; it includes information in each encoded word in addition to the number of continuous *fill*s. After WAH encoding, if there exists a *chunk* with only one *set bit*, and this *chunk* is just before a *fill*, then it is defined as an *N-fill*. It is combined with the following *fill*, and the position of the sole *set bit* is recorded in the newly created word by using five bits to represent the *set bit*'s position. This codeword is denoted as [NL-F]. Other *literal*s remain unchanged in CONCISE. Fig. 1 shows an example of CONCISE.

2.4 COMPAX

COMPAX (*COMPressed Adaptive indeX*) provides a different strategy from CONCISE. While CONCISE focuses on merging two contiguous words, COMPAX can deal with three contiguous words after WAH encoding. Thus it consumes less memory to store

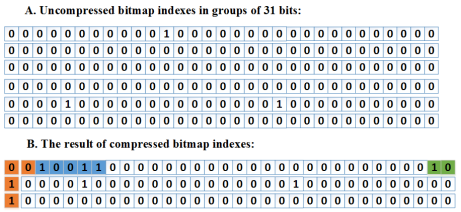


Fig. 1 An examples of CONCISE

bitmap indexes. In COMPAX, besides *fill* and *literal* (borrowed from WAH), another two encoding schemes are introduced.

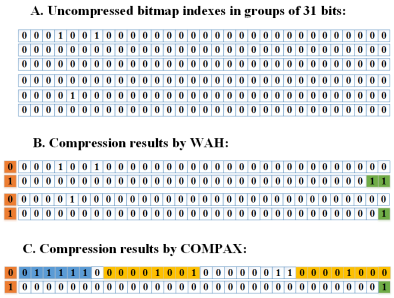


Fig. 2 An examples of WAH and COMPAX

In a *literal*, if only one byte contains both *set bit*s and *unset bit*s, while other bits in this word are composed of only *set bit*s or *unset bit*s, this byte is called a *dirty byte* and this *literal* is called *L* for short. In an *L*, if the three bytes other than the *dirty byte* are all composed of only *unset bit*s, then this *dirty byte* is classified as 0-*dirty* and this *L* is defined as 0-*L*. 1-*dirty* and 1-*L* are similarly named.

There are also [LFL] and [FLF] codewords in COMPAX. An [LFL] codeword combines three consecutive WAH words (*L* + *fill* + *L*), and an [FLF] codeword combines three consecutive WAH words (*fill* + *L* + *fill*). In COMPAX, F only represents 0-*fill* and *L* only represents 0-*L*. Fig. 2 shows how COMPAX works and achieves better compression than WAH.

2.5 COMBAT

COMBAT (*COMbining Binary And Ternary encoding*) is similar to CONCISE and COMPAX in that two or three contiguous words can be combined into a single word. Its specific coding schemes are introduced below.

In COMBAT, the definitions of [FLF] and [LFL] codewords are extended from those of COMPAX. But unlike COMPAX terminology, F in COMBAT can denote both 0-*fill* and 1-*fill*, and *L* can also denote both 0-*L* and 1-*L*. The composition of the [FLF] and [LFL] codewords is shown in Fig. 3 and Fig. 4.

COMBAT also shares some characteristics with CONCISE in coding schemes. If there already exist two contiguous words, *L* and *fill* (but without another *L* following), they can be combined in COMBAT. This codeword is called [LF]; and it is shown in Fig. 5.

Another definition, *NI2-L*, is introduced in COMBAT, leading to a new kind of codeword. *NI2-L* refers to a *literal* containing only two *dirty byte*s (of the same type). If in an *NI2-L*, the two *dirty byte*s that are not the *dirty byte*s are composed of only *unset bit*s, this *NI2-L* is called a 0-*NI2-L*. We define 1-*NI2-L* similarly. So if an *NI2-L* is just before a *fill* after WAH encoding, then these two words can be merged into a new one, called an [NI2-LF]. This kind of codeword is shown in Fig. 6.

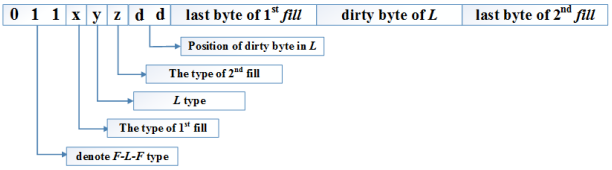


Fig. 3 [FLF] codeword in COMBAT

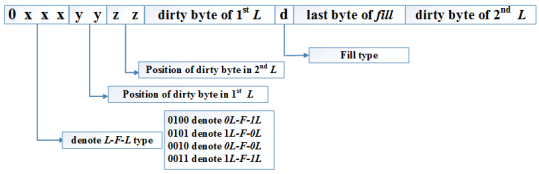


Fig. 4 [LFL] codeword in COMBAT

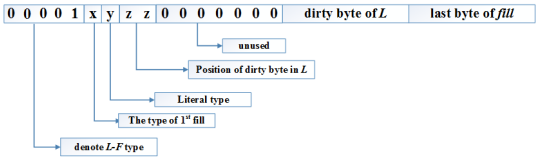


Fig. 5 [LF] codeword in COMBAT

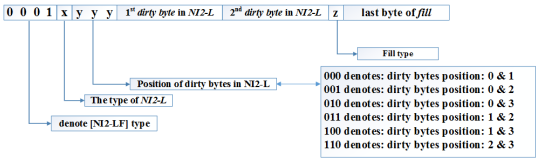


Fig. 6 [NI2-LF] codeword in COMBAT

3 THEORETICAL ANALYSES

In this section, we discuss an ideal distribution in bitmap index-uniform distribution, which is a part of the theoretical analysis found in [30]. Similar to [30], We

list some assumptions below that simplify our analysis. The following analysis is composed of three different possible cases, i.e. sparse bitmaps, dense bitmaps, and bitmaps following a Zipf distribution.

i. The positions and density of *set bit*s are independent from each other.

ii. According to the encoding schemes of COMBAT, COMPAX and CONCISE, only three *chunk*s occur in a raw bitmap index.

3.1 Sparse Bitmap

In the case of sparse bitmaps, we assume that the density of *set bit*s (denoted by d) is very small, which can facilitate Taylor expansion. Only terms of the first and second degree are retained in the following analysis.

According to [30], the theoretical compression results of CONCISE and COMBAT (denoted by $\overline{L_{CONCISE}}$ and $\overline{L_{COMBAT}}$) are shown below:

$$\overline{L_{CONCISE}} = 1 + 62d + 1922d^2; \quad (1)$$

$$\overline{L_{COMBAT}} = 1 + 31d + 496d^2; \quad (2)$$

Although [30] does not provide the theoretical compression results of COMPAX, they should be the same as the compression results of SECOMPAX in [30], because the two algorithms share the same compression schemes, in the case of sparse bitmaps. So the corresponding result of COMPAX is:

$$\overline{L_{COMPAX}} = 1 + 62d - 210d^2; \quad (3)$$

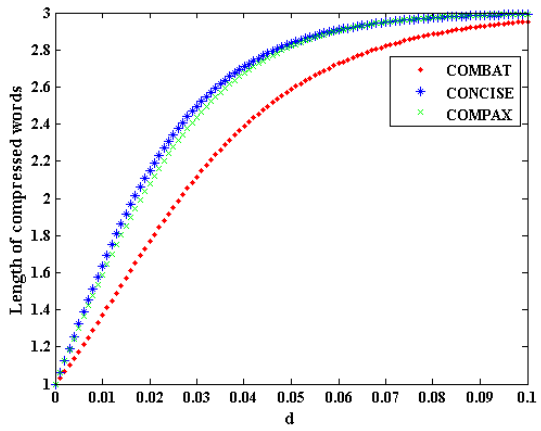


Fig. 7 The simulation results in sparse bitmaps

We can now compare spatial performance in COMBAT with COMPAX and CONCISE according to the equations above. When the value of d is very close to 0, more space can be expected to be saved after COMBAT encoding because the coefficient of the monomial term in $\overline{L_{COMBAT}}$ is smaller than those in the other two.

In order to show the differences between COMBAT and other algorithms, some simulation results are provided in Fig. 7, where the value of d ranges from 0 to 0.1. Since in most cases the density of real datasets does not exceed 10%, the simulation results can be a reflection of reality.

From Fig. 7, it is obvious that COMBAT can beat CONCISE and COMPAX in the given density interval, and its savings are 10% on average. This result is expected because COMBAT provides more compression schemes than COMPAX or CONCISE in this case. Although the savings are not enormous in terms of percentage, in the context of Big Data when the data can reach ZB levels, the actual savings are still considerable.

3.2 Dense Bitmap

In practice, sparse bitmaps represent most cases. However, we cannot omit a special case when the bitmap is very dense. For example, in the field of network intrusion detection, when explosive traffic appears, clustered *set bit*s can be expected. Or within local area networks, the number of possible IP addresses is limited and thus continuous *set bit*s can be expected. In the following analysis, the value of d is assumed to be very large, approaching one. Taylor expansion is also applicable in this subsection, but in order to simplify it, another variable r is introduced and assigned the value of 1 - d . Thus r approaches zero here. This is similar to what is seen in subsection 3.1- only the first and second degree terms are kept in the Taylor expansion.

As in [30], the *Basic probabilities*, i.e. the probabilities of *fill*, *L*, *NI2-L*, *literal*, and *N-fill* in an uncompressed bit sequence can be calculated. But here, we denote these values using r, which is realized by replacing d in the probability value from [30] with *1 - r*. These new probability values and their simplified values are presented in Table 1.

Probabilities of all kinds of compressible three-word combinations with corresponding compressed lengths from COMBAT, COMPAX and CONCISE can be computed by replacing d with 1 - r in corresponding values from [30]. All these values are presented in Table 2.

Table 1 Value of *Basic probabilities*

chunk type	symbol	Value	Simplified value
0-*fill*	p_1	r^{31}	0
1-*fill*	p_2	$(1-r)^{31}$	$1-31r+465r^2$
0-*L*	p_3	$(1-r^7)r^{24}+3(1-r^8)r^{23}$	0
1-*L*	p_4	$(1-(1-r)^7)(1-r)^{24}+3(1-(1-r)^8)(1-r)^{23}$	$31r-825r^2$
0-NI2-*L*	p_5	$3(1-r^7)(1-r^8)r^{16}+3(1-r^8)^2r^{15}$	0
1-NI2-*L*	p_6	$3(1-(1-r)^8)^2(1-r)^{15}+3(1-(1-r)^7)(1-(1-r)^8)(1-r)^{16}$	$360r^2$
literal	p_7	$1-(1-r)^{31}-r^{31}$	$31r-465r^2$
N-fill	p_8	$C_{31}^1 r(1-r)^{30}+C_{31}^1(1-r)r^{30}$	$31r-930r^2$

Table 2 Probability value (COMBAT)

word combination and corresponding algorithms	Compressed length	Calculated value
0-*fill*+0-*fill*+0-*fill* (all)	1	0
1-*fill*+1-*fill*+1-*fill* (all)	1	$1-93r+4278r^2$
0-*fill*+0-*fill*+1-*fill* (all)	2	0
1-*fill*+0-*fill*+0-*fill* (all)	2	0
1-*fill*+1-*fill*+0-*fill* (all)	2	0
0-*fill*+1-*fill*+1-*fill* (all)	2	0
0-*fill* + 0-*L* + 0-*fill*(COMPAX)	1	0
fill + *L* + *fill*(COMBAT)	1	$31r-2747r^2$
0-*L* + 0-*fill* + 0-*L*(COMPAX)	1	0
L + *fill* + *L*(COMBAT)	1	$961r^2$
fill + *fill* + *literal* (all)	2	$31r-2387r^2$
literal + *fill* + *fill* (the same type of *fill*,COMPAX)	2	$31r-2387r^2$
literal(not *N-fill*) + *fill* + *fill* (the same type of *fill*, CONCISE)	2	0
literal + *L* + *fill*([LF], COMBAT)	2	$961r^2$
L + *fill* + *literal*(not *L*,[LF], COMBAT)	2	0
literal (not *L*) + *fill* + *fill* (COMBAT)	2	$360r^2$
L+ *fill* + *fill*(COMBAT)	1	$31r-2747r^2$
NI2-L + *fill* +*literal* ([NI2-LF], COMBAT)	2	$360r^2$
NI2-L + *fill* + *fill* ([NI2-LF], COMBAT)	1	$360r^2$
Any type + *NI2-L* + *fill* ([NI2-LF], COMBAT)	2	$360r^2$
N-fill+0-*fill*+ 0-*fill*(CONCISE)	1	0
N-fill+1-*fill*+1-*fill* (CONCISE)	2	0
Any type of word + *N-fill* + 0-*fill* (CONCISE)	2	0

Based on the probabilities above, the mathematical expectation of the compressed length after COMBAT, COMPAX, CONCISE encoding(denoted by $\overline{L_{COMBAT}}$,$\overline{L_{COMPAX}}$ and $\overline{L_{CONCISE}}$) are shown as follows:

$$\overline{L_{COMBAT}} \approx 1+31r+856r^2 \quad (4)$$

$$\overline{L_{COMPAX}} \approx 1+124r-3782r^2 \quad (5)$$

$$\overline{L_{CONCISE}} \approx 1+124r-3782r^2 \quad (6)$$

Likewise, the monomial coefficient in $\overline{L_{COMBAT}}$ is smaller than those of $\overline{L_{COMPAX}}$ and $\overline{L_{CONCISE}}$. That means that when the value of r is approaching zero, i.e., the value of d is approaching one, COMBAT has better compression performance than the other two algorithms.

In fact, in the case of dense bitmaps, the codewords [FLF], [LFL] in COMPAX and [N-LF] in CONCISE have nearly no influence on the compression, thus degenerating into WAH.

Similarly, simulation results in this case are presented in Fig. 8 and the superiority of COMBAT can be shown more explicitly. As shown in Fig. 8, when the value of d ranges from 0.99 to 1, COMBAT has much better spatial performance than the other two algorithms while COMPAX has nearly the same performance as CONCISE in this case.

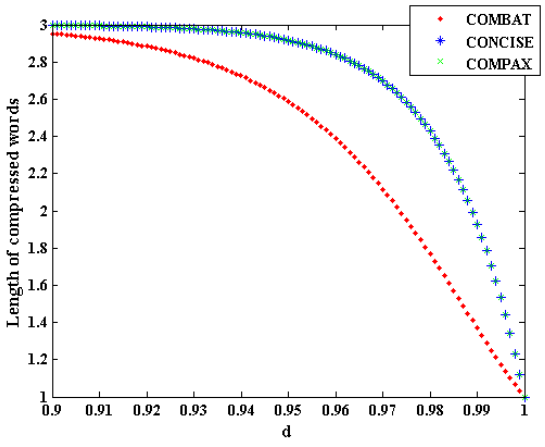


Fig. 8 The simulation results in dense bitmaps

3.3 Bitmaps Following Zipf Law

The dataset in reality follows a zipf distribution, which contains N possible values. Based on the Zipf law, the key values v_1, v_2...v_N belonging to one column of a dataset rank 1^{st}, 2^{nd}...N^{th} separately. The probability of the i^{th} common key value would be $p(i)=\frac{c}{i^{\alpha}}$

For every single bitmap index, which represents one single key value, we assume that it follows a uniform distribution. But the density of these bitmap indexes will not approach zero or one. Similar to the previous analysis, the value of each of its *Basic probability* is derived and listed in Table 3.

Since all the compressible three-word combinations in COMBAT, COMPAX and CONCISE are known to us, their values with respect to newly derived

Table 3 Value of *Basic probabilities* for the i^{th} common key value

chunk type	symbol	Original value
0-*fill*	$p_1(i)$	$(1-p(i))^{31}$
1-*fill*	$p_2(i)$	$p(i)^{31}$
0-*L*	$p_3(i)$	$(1-(1-p(i))^7)(1-p(i))^{24}+3(1-(1-p(i))^8)(1-p(i))^{23}$
1-*L*	$p_4(i)$	$(1-p(i)^7)p(i)^{24}+3(1-p(i)^8)p(i)^{23}$
0-*NI2-L*	$p_5(i)$	$3(1-(1-p(i))^7)(1-(1-p(i))^8)(1-p(i))^{16}+3(1-(1-p(i))^8)^2(1-p(i))^{15}$
1-*NI2-L*	$p_6(i)$	$3(1-p(i)^8)^2p(i)^{15}+3(1-p(i)^7)(1-p(i)^8)p(i)^{16}$
literal	$p_7(i)$	$1-p(i)^{31}-(1-p(i))^{31}$
N-fill	$p_8(i)$	$C_{31}^1(1-p(i))p(i)^{30}+C_{31}^1p(i)(1-p(i))^{30}$

Basic probabilities can be calculated, which is the same for the following compressed length (denoted by $\overline{L_{COMBAT}(i)}$, $\overline{L_{COMPAX}(i)}$, $\overline{L_{CONCISE}(i)}$ for the i^{th} common key value). The total compressed length of COMBAT, COMPAX and CONCISE is:

$$\overline{L_{COMBAT}}=\sum_{i=1}^{n}\overline{L_{COMBAT}(i)} \tag{7}$$

$$\overline{L_{COMPAX}}=\sum_{i=1}^{n}\overline{L_{COMPAX}(i)} \tag{8}$$

$$\overline{L_{CONCISE}}=\sum_{i=1}^{n}\overline{L_{CONCISE}(i)} \tag{9}$$

Since the calculation of $\overline{L_{COMBAT}}$, $\overline{L_{COMPAX}}$ and $\overline{L_{CONCISE}}$ is complex, some simulation results from Matlab are provided in Fig. 9, Fig. 10 and Fig. 11. Since the Zipf law is restricted to two factors, i.e. α and c, the simulated compressed length changes with the two variables in the three figures.

From Fig. 9, Fig. 10 and Fig. 11, it is obvious that the length of compressed words in COMBAT is less than those in COMPAX and CONCISE with the change of α and c. The savings range from 5% to 10%, which demonstrates that COMBAT consume less memory and storage in theory than COMPAX and CONCISE.

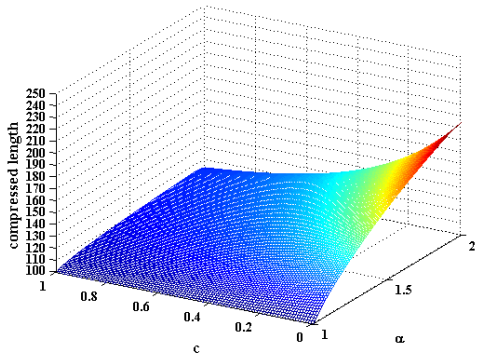


Fig. 9 The simulation results of COMBAT with the change of α and *c*

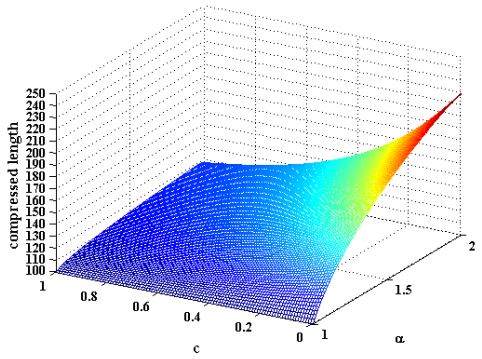


Fig. 10 The simulation results of COMPAX with the change of α and *c*

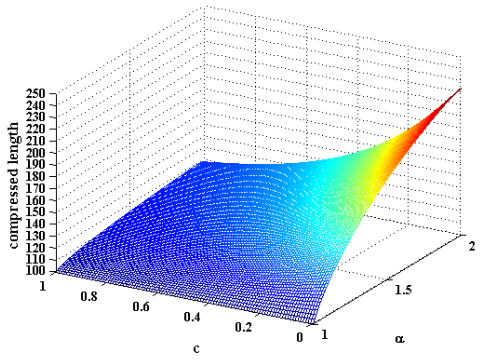


Fig. 11 The simulation results of CONCISE with the change of α and *c*

4 EXPERIMENT ANALYSIS

4.1 Datasets and Experimental Setting

Two real datasets are used in our experiments. One is netflow data from CAIDA 2013, which is composed of up to 13 million records including source IP, destination IP, source port, destination port and protocol type. The other dataset is CDR (call detail records) billing data from China Unicom. One file contains all the communication logs from one day. Two files were selected for the following experiments. Up to 900,000 communication records are contained in one file, each of which is composed of a time stamp denoting the starting time of a call, sending end number, receiving

end number, etc.

The codes for the following experiments originate from parts of codes in Druid. And all the experiments are executed in the same JVM (Java virtual machine) in a 64-bit Ubuntu Server with a Intel Core i7 CPU with 18GB RAM. In order to avoid mutual interference, only one program runs in the machine at one time.

4.2 Experiment Results

4.2.1 CAIDA data

Fig. 12 to 14 give the experimental results from CAIDA 2013 data. Four features are selected for the experiments-source IP address, destination IP address, source port and destination port. Four separate bytes (0-255) comprise a single IP address, which facilitates the creation of a bitmap index for each byte. A port number occupies two bytes in memory, which corresponds two separate bitmaps in the following experiments.

Fig. 12 is the storage comparison after compressing with CONCISE, COMPAX, COMBAT and WAH respectively. It contains the total size of index files, as well as the corresponding storage ratio to that of COMBAT. From the figure, it is obvious that COMBAT beats the other three algorithms. The savings are 7%, 8%,4% and 4% in source IP, destination IP, source port and destination port when compared to COMPAX, which consumes the least storage among the other three algorithms. Although the improvement is not enormous at first glance, the savings can be still considerable in practice when considering the huge amount of data.

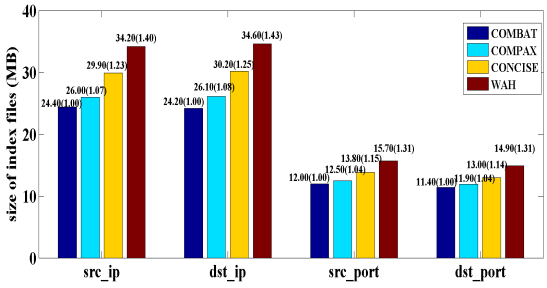


Fig. 12 Storage comparison with CAIDA data

Fig. 13 shows a comparison of querying time, including time consumed by the loading process. In order to avoid fortuity, up to 500 diverse queries are created. For each query, an IP address or a port number is retrieved from the index files which are created randomly in the experiments.

The results from Fig. 13 show that querying with COMBAT is faster than that with the other three algorithms in most cases. Admittedly, it is slightly slower when it comes to destination port number. However, the speed-up is still about 4%, 3% and 1% in other three cases. The speedup in COMBAT is traceable to its smaller index files, resulting in quicker loading. To verify this point, the loading time is also recorded in Fig. 14.

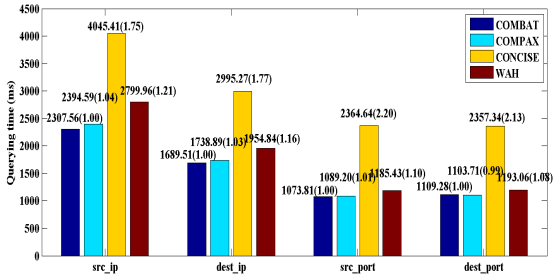


Fig. 13 Querying time comparison with CAIDA data

In Fig. 14, all the index files are loaded sequentially, showing that the loading process of COMBAT is faster than those of its competitors, and the speed-up can reach 7%, which is proportional to the savings in index files. The faster loading process is the main reason for the speed-up in the querying process. This apparently minor improvement matters a lot when taking various practical factors into account. For example, in distributed systems, data are uploaded and downloaded frequently, and thus IO speed is a major issue in system performance, which can be largely determined by index file loading time.

The improvement in querying can be also accounted for by the coding schemes themselves. COMBAT is an extended version of compression scheme of COMPAX, which differs from CONCISE. According to the coding schemes of CONCISE, in the query process, Boolean operations are conducted between two indexes; they conclude by retrieving the position of the combined *set bit*s in a compressed word. The retrieval processes slow down the Boolean operations because the position calculations are complex. In contrast, because fast shifting operations can be performed directly by a CPU, and thus all the *dirty byte*s inside words encoded by COMBAT or COMPAX at the top possible speed. The Boolean operations that follow are also performed very efficiently.

In order to verify the workability of the coding schemes, the codeword statistics of CAIDA data are presented in Fig. 15. Obviously, all kinds of COMBAT codewords are used in CAIDA data. The numbers

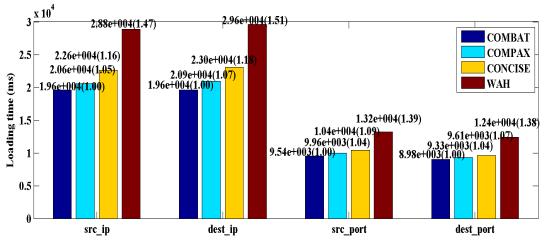


Fig. 14 Loading time with CAIDA data

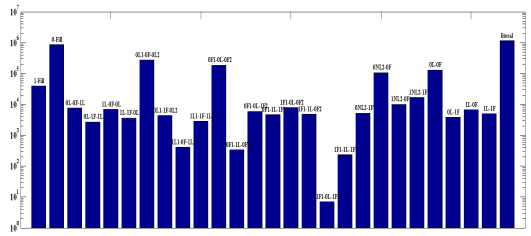


Fig. 15 Codeword statistics with CAIDA data

of codewords other than *fill* and *literal* are nearly the same order of magnitude, which differs somewhat from the theoretical analysis. The reason is that all data in CAIDA 2013 are reordered, which leads to more concentrated distribution of *set bit*s and *unset bit*s and thus to a higher probability of various codewords after COMBAT encoding.

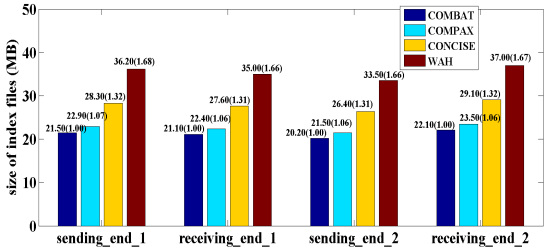


Fig. 16 Storage comparison with CDR data

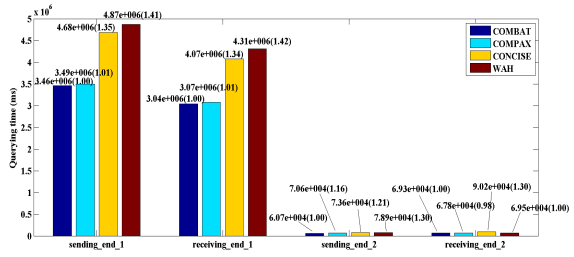


Fig. 17 Querying time comparison with CDR data

4.2.2 CDR Data

As noted earlier, two CDR data files were used in our experiments. Within each file, various features compose a single record; here we are only concerned about phone

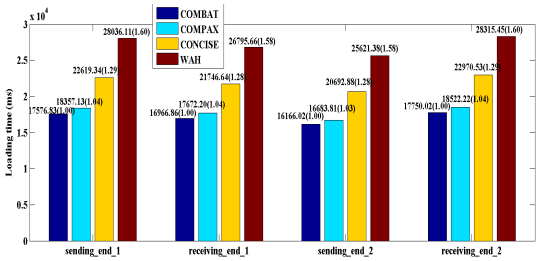


Fig. 18 Loading time comparison with CDR data

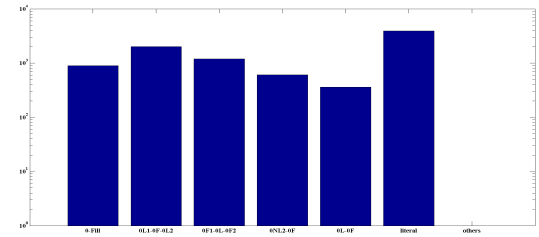


Fig. 19 codeword statistics with CDR data

numbers (both the sending end and the receiving end) because only this feature is non-numerical, and thus corresponding bitmap indexes can be built.

Similar to what we present in section 4.2.1, Fig 16 to 18 compare the performance of the different algorithms on CDR data. Spatial performance with CDR data under the various algorithms is shown in Fig. 16. COMBAT has 6% better spatial performance than COMPAX.

Query performance on CDR data is provided in Fig. 17, which has nearly the same trends as CAIDA data results. However, when it comes to CDR data, the querying speed-up is as much as 16%. This demonstrates that the advantage of COMBAT is more obvious with specifically distributed data. In order to figure out where the savings originate, loading time is also presented in the following figure.

According to Fig. 18, COMBAT consumes the least loading time, and the improvement is nearly the same as that in CDR data. However, it is not convincing enough to become the top factor for the savings in querying time because the speed-up in the second sending-end dataset varies from that in the other datasets.

In order to resolve this contradiction, all kinds of codewords of COMBAT with CDR data are counted in Fig. 19. From Fig. 19, we can see that the statistical results differ greatly from those of CAIDA data. Not all the codewords exist and the number of codewords, i.e., 0-*fill*, 0L1-0F-0L2, 0F1-0L-0F2, 0NL2-0F, 0L-0F and *literal*, are nearly the same. Among the existing

codewords, 0NL2-0F and 0L-0F do not appear in COMPAX- which ensures the superiority of COMBAT in storage. So the smaller indexes can reduce the overhead in loop operations in the process of Boolean operations processes. Moreover, since many other codewords disappear in this case, the coding schemes become relatively succinct, which can leave out a lot of unnecessary conditional judgments.

5 CONCLUSION AND FUTURE WORK

In this paper, a new bitmap index coding algorithm named COMBAT is proposed, and its superiority is both theoretically and practically demonstrated in comparison with COMPAX and CONCISE, two well-known bitmap index coding algorithms. The theoretical analysis extends the mathematical model in [30] for analysis of bitmap index performance, which contains more possible cases, including sparse bitmaps, dense bitmaps, and bitmaps following Zipf's law. According to the analysis, COMBAT beats COMPAX and CONCISE in terms of spatial performance because COMBAT can provide more coding schemes, and can provide compression in more cases.

Experiments based on real data sets from CAIDA 2013 and CDR data also prove that COMBAT has a strong advantage in both storage and querying time. The savings are up to 7% in storage and 16% in querying. Although they are not enormous, the improvement can play an important role in improving the existing systems in both spatial and temporal performance. The statistics of codewords in COMBAT also demonstrates that COMBAT is suitable for a variety of datasets. When more codewords in COMBAT are used, better compression effects can be expected; and when the opposite conditions obtain, smaller indexes can give rise to less overhead in Boolean operations in the querying process.

In the future, in order to demonstrate the effectiveness of our algorithm in practice, more experiments in COMBAT will be conducted, including experiments on GPUs and real Big Data platforms. Then COMBAT will be integrated into real data management systems and make a contribution to solving the real-time querying problem in Big Data.

Acknowledgements

This work was supported in part by the Ministry of Science and Technology of China under National 973 Basic Research Program (No.2013CB228206 and No.2012CB315801), National Natural Science Foundation of China (grant No. 61233016), and China NSFC A3 Program (No.61140320). This is also supported by National Training program of Innovation and Entrepreneurship for Undergraduates with Project No.201410003033 and No.201410003031.

References

[1] W. Huang, Z. Chen, W. Dong, H. Li, B. Cao, and J. Cao, Mobile Internet Big Data Platform in China Unicom, Tsinghua Science and Technology, Volume 19, Issue 1, pp.95-101, 2014.

[2] L. A. Barroso, J. Clidaras, and U. H?lzle, The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines, Synthesis Lectures on Computer Architecture, doi:10.2200/S00516ED2V01Y201306CAC024.

[3] Z. Chen, W. Huang, and J. Cao, Big Data Engineering for Internet Traffic, Beijing: Tsinghua University Press, 2014.

[4] P. Cheng, bitmap index techniques and its research advancement, Science and technologies information, Vol. 026, pp.134-135, 2010.

[5] J. Li, Research in bitmap index in data warehouse, (in Chinese), PhD diss, Shandong University, 2007.

[6] Z. Huang, W. Lv, and J. Huang, Improved BLAST algorithm based on bitmap indexes and B+ tree, Computer Engineering and Applications, 49(11), pp.118-120, 2013.

[7] B. Yang, Y. Qi, Y. Xue, and J. Li, Bitmap data structure: Towards high-performance network algorithms designing, Computer Engineering and Applications, 45(15), 2009.

[8] H. Garcia-Molina, J. D. Ullman, and J. Widom, Database System implementation, Second Edition, Prentice Hall, 2009.

[9] C. Chan, Bitmap Index, in Encyclopedia of Database Systems, Springer, 2009, pp. 244-248.

[10] M. Wu and A.P. Buchmann, Encoded Bitmap Indexing for Data Warehouses, Proc. 14th Intl Conf. Data Eng. (ICDE), pp.220-230, 1998.

[11] F. Corrales, D. Chiu, and J. Sawin, Variable Length Compression for Bitmap Indexes, in DEXA11, Springer-Verlag, pp.381-395, 2011.

[12] A. Colantonio, and R. Di Pietro, Concise: Compressed n composable integer set, In Information Processing Letters, 110(16), 2010, pp.644-650.

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

[14] K. Wu, Ekow J. Otoo , and A. Shoshani, Compressing bitmap indexes for faster search operations. In Scientific and Statistical Database Management, 2002. Proceedings. 14th International Conference on, pp. 99-108. IEEE, 2002.

[15] K. Wu, Ekow J. Otoo , and A. Shoshani, Optimizing bitmap indexes with efficient compression, in ACM Transactions on Database Systems (TODS), 31(1), 2006, pp.1-38.

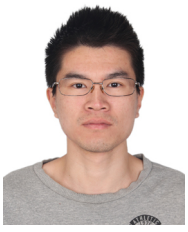
[16] C. Guadalupe, M. Gibas, and H. Ferhatosmanoglu, Update conscious bitmap indexes, 19th IEEE International Conference on Scientific and Statistical Database Management SSBDM07, pp. 15-15, 2007.

[17] M. Stabno, and R. Wrembel. RLH: Bitmap compression technique based on run-length and Huffman encoding, Information Systems 34, no. 4, 2009, pp.400-414.
[18] F. Deli'ege and T. B. Pedersen, Position list word aligned hybrid: optimizing space and performance for compressed bitmaps, In Proceeding of the 13th International Conference on Extending Database Technology, 2010.
[19] D. Lemire, O.Kaser, and K. Aouiche, Sorting improves word-aligned bitmap indexes, Data & Knowledge Engineering, 69(1), pp.3-28, 2010.
[20] S. J. van Schaik and O. de Moor, A memory efficient reachability data structure through bit vector compression, In Proceedings of the 2011 ACM SIGMOD International Conference on Management of data, pp. 913-924. ACM, 2011.
[21] F. Fusco, M. P. Stoecklin, and M.Vlachos, Net-fli: on-the-fly compression, archiving and indexing of streaming network traffic, Proceedings of the VLDB Endowment, 3(1-2), pp.1382-1393, 2010.
[22] W.Andrzejewski, and R.Wrembel, GPU-WAH: Applying GPUs to compressing bitmap indexes with word aligned hybrid, In Database and Expert Systems Applications, Springer Berlin Heidelberg, January, pp. 315-329, 2010.
[23] F. Fusco, M. Vlachos, X. Dimitropoulos, and L. Deri, Indexing million of packets per second using GPUs, In Proceedings of the 2013 conference on Internet measurement conference, pp.327-332. ACM, 2013.
[24] W. Andrzejewski, and R. Wrembel, GPU-PLWAH: GPU-based implementation of the PLWAH algorithm for compressing bitmaps, Control & Cybernetics, 40(3), pp. 627-650, 2011.
[25] Y. Wen, Z. Chen, G. Ma, J. Cao, W. Zheng, G. Peng, and W. L. Huang, SECOMPAX: A bitmap index compression algorithm, In 23rd International Conference on Computer Communication and Networks (ICCCN), IEEE, pp. 1-7, 2014.
[26] J. Chang, Z. Chen, W. Zheng, Y. Wen, J. Cao, and W. L. Huang, PLWAH+: a bitmap index compressing scheme based on PLWAH, In Proceedings of the tenth ACM/IEEE symposium on Architectures for networking and communications systems, ACM, pp. 257-258, 2014.
[27] A. Schmidt, D. Kimmig, and M. Beine, DFWAH: A Proposal of a New Compression Scheme of Medium-Sparse Bitmaps, in the Third International Conference on Advances in Databases, Knowledge, and Data Applications (DBKDA 2011), pp. 192-195.
[28] S. Chambi, D. Lemire, O. Kaser, and R. Godin, Better bitmap performance with Roaring bitmaps, arXiv preprint arXiv:1402.6407 (2014).
[29] Z. Chen, Y. Wen, J. Cao, W. Zheng, J. Chang, Y. Wu, G. Ma, M. Hakmaoui, G. Peng, A Survey of Bitmap Index Compression Algorithms for Big Data, Tsinghua Science and Technology, 20(1), February 2015.

Yinjun Wu is an undergraduate student studying in Department of Automation at Tsinghua University. His research interests include bitmap indexing algorithms.

Yuhao Wen is an Ph.D. student studying in Department of Computer Science at Duke University. His research interests include big data and networks.

Zhen Chen is the group leader of Internet+ lab in iCenter of Tsinghua University now. He has published 109 academic papers, 7 patents and 5 books in computer network area since 2003. His research interests include network architecture and data management. He was awarded excellent mentor for SRT (student research training) program in Tsinghua University in 2015, 2014 and 2013 respectively. He worked as associate professor in Research Institute of Information Technology of Tsinghua University during 2006 to 2015. He once worked as visiting scholar in network group in ICSI of UC Berkeley in 2006. During 2004 to 2006, He was a postdoctoral researcher in Network Institute of Department of Computer Science and Technology in Tsinghua University. He received his B.E. and Ph.D. degrees from Xidian University in 1998 and 2004.



Wenxun Zheng is an master student studying in Department of Automation at Tsinghua University. His research interests is now on bitmap index compression.

Junwei Cao is currently Professor and Deputy Director of Research Institute of Information Technology, Tsinghua University, China. He is also Director of Open Platform and Technology Division, Tsinghua National Laboratory for Information Science and Technology. His research is focused on advanced computing technology and applications. Before joining Tsinghua in 2006, Junwei Cao was a Research Scientist of Massachusetts Institute of Technology, USA. Before that he worked as a research staff member of NEC Europe Ltd., Germany. Junwei Cao got his PhD in computer science from University of Warwick, UK, in 2001. He got his master and bachelor degrees from Tsinghua University in 1998 and 1996, respectively. Junwei Cao has published over 130 academic papers and books, cited by international researchers for over 3000 times. Junwei Cao is a Senior Member of the IEEE Computer Society and a Member of the ACM and CCF.