SPLWAH: a bitmap index compression scheme for searching in archival Internet traffic

Jiahui Chang, Zhen Chen*, Wenxun Zheng,

Junwei Cao, Yuhao Wen, Guodong Peng

Research Institute of Information Technology, Tsinghua University

Tsinghua National Lab for Information Science and Technologies (TNList), Beijing, China

zhenchen@tsinghua.edu.cn

Wen-Liang Huang

China Unicom Groups Labs

China Unicom Groups

Beijing, China

huangwenliang@gmail.com

***Abstract*—Bitmap index is widely used in archiving and searching of Internet traffic, which is an essential step for analyzing network events in the field of network forensics. However, bitmap index requires a large storage space for fast searching in archival data. As current state-of-the-art bitmap index compression techniques, various encoding algorithms have been proposed, e.g. WAH, PLWAH, COMPAX, etc. With the advantages of fast query speed and easy implementation, PLWAH is an outstanding encoding scheme to encode the sparse dirty bits in bitmap index. Unfortunately, for searching Internet traffic, the constructed bitmap index can be quite dense locally according to the statistics. This is because that Internet traffic are usually composed of the flows with the same five tuple *(SrcIP, SrcPort, DstIP, DstPort, proto)*. In this paper, SPLWAH is proposed to adapt to Internet traffic based on PLWAH. In SPLWAH, a new codebook is introduced to fit the characteristics of Internet traffic. We also conduct several performance evaluation experiments based on real network flow data from CAIDA. The results show that SPLWAH reduces the space consumption with a factor of 20% or more without incurring extra encoding and decoding cost. This work also shows that the design space in bitmap index compression is still a fruitful unknown frontier and worth further exploring to adapt to the emerging data spaces.**

Keywords—Bitmap index, Bitmap index compression, Big Data, WAH, PLWAH, Internet traffic, Flow trace, Performance evaluation, Network forensic.

III. INTRODUCTION

A. Bitmap Index

Bitmap indexing [4-6] was proposed by P'O Neil in 1987, and deployed in a commercial database system called Model 204 for the first time [4]. And it uses a bit vector or a sequence of bits to indicate the value of the index whether exists in the indexed data, which can efficiently use bit logical operations (AND/OR/NOT/XOR, etc.) to answer the complex queries. A simple bitmap indexing example is shown in Fig. 1.

Bitmap index is designed for scientific data and databases. The scientific data are usually generated by scientific instruments or mathematical simulation, and it is characterized by an extremely large amount of data without changing. Bitmap index database solves the problem of how to quickly identify a small amount of data in a mass of scientific data, while traditional relational databases are not suitable for this work.

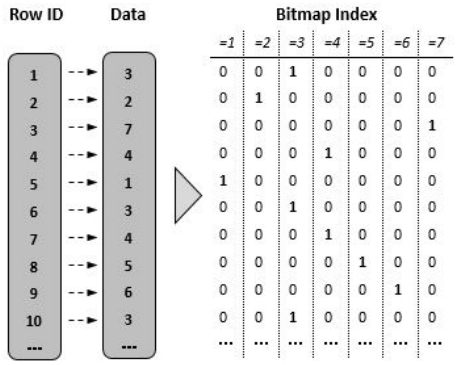


Fig. 1. An example of bitmap index.

B. Bitmap Index Compression

In bitmap index based database, data are usually stored in a columnar way, where every column is stored together and a bitmap index is created correspondently. The core technology used in bitmap indexing includes bitmap index compression and others.

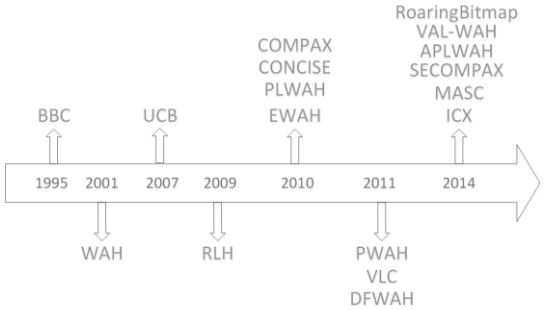


Fig. 2. The advancement of bitmap index compression algorithms.

Currently, the representative bitmap index compression schemes include BBC [7], WAH [8-9], UCB [10], RLH [11],

PLWAH [12], EWAH [13], CONCISE [14], COMPAX [15] and VLC [19], PWAH [20], VAL-WAH [21], DFWAH [22], Roaring Bitmap [23], BREAD [24], etc. GPU implementations of WAH and PLWAH are introduced in [16-18]. A more detailed survey of bitmap index compression algorithms is presented in [25]. Fig. 2 also shows the various bitmap index compression algorithms appearing in chronological order.

PLWAH has some minor improved versions such as APLWAH (PLWAH with adaptive counter), and enhanced variants such as PLWAH+ [26]. COMPAX also has minor improved versions such as COMPAX with oLSH, COMPAX2, and enhanced variants such as SECOMPAX [27].

C. Internet Traffic

With the popularity of Internet applications and mobile wireless networks for large-scale commercialization, huge amounts of information content greatly enriches users. The outbreak of mobile Internet which allows users from anywhere and anytime access to any content of the network, results in generating more traffic data. The entire Internet traffic maintains a rapid growth as a normal Internet company generates and accumulates users. Internet traffic is quite large, which cannot use gigabit (G) or trillion (T) to measure. Cisco's report [1] predicts that Internet traffic data will grow four-fold from 2011 to 2016 and reach 1.3 ZB in 2016. Internet traffic is a typical streaming data and needs to be explored with big data platform based on bitmap indexing [2, 15].

D. Main Contribution

For faster retrieval and better space efficiency, bitmap indexes are usually sorted in practice. However, It is observed that PLWAH does not perform well in the dense bitmap with few dirty bit positions, especially when it comes to Internet traffic and other sorted data.

Based on the observation of the compression results of current state-of-the-art bitmap index compression algorithms, this paper presents a new bitmap index compression technique that outperforms PLWAH in both storage and query performance perspectives. This new bitmap index compression scheme is named as SPLWAH (Position List Word Aligned Hybrid algorithm for Sorted data).

The remainder of this paper is structured as follows. A more detailed encoding scheme is provided in Section 2 while Section 3 presents a FSM model to describe the encoding procedure of SPLWAH. Section 4 shows the experiments conducted in real network flow data from CAIDA. Finally, we conclude and discuss future research directions in Section 5.

IV. SPLWAH ENCODING

A. The Main Idea

SPLWAH keeps the idea of the "Position List" in PLWAH to record "Switch Positions" in Literal Chunk, which is more efficient in the case where the fraction of the set bit is not at a low level. For isolated "1" in the middle position of a Literal Chunk, SPLWAH pays double bits than PLWAH to store its position. However, when the number of continuous "1" is larger than two, SPLWAH uses less bits to store the positions than PLWAH [12].

Furthermore, the new definition of Simple Chunk which can be piggybacked by Fill codeword abandons the concept of the diverse types, which is definitely different from other algorithms.

At last, the definitions of FSF and SFS codewords perform better in cases where the bitmap indexes are sorted well. According to a number of experiments, a conclusion can be naturally made that SPLWAH is more suitable for the dense and clustered bitmap index.

Details of the definitions and compression processes will be discussed in later subsections.

B. Definitions for Chunks

The original bitmap consists of columns which have a large amount of sequences of bits divided into 31-bit-long chunks to ensure they fit into the L1 cache. All processes being carried out are based on the chunks, which is more suitable for modern CPU architecture. Firstly, each chunk is classified into different types as follows:

0-Filled Chunk: If the 31 bits of a chunk are all "0", the chunk is called "0-Filled Chunk".

1-Filled Chunk: If the 31 bits of a chunk are all "1", the chunk is called "1-Filled Chunk".

Generally, 0-Filled Chunk and 1-Filled Chunk are two types of *Filled Chunk*.

Literal Chunk: If a chunk cannot be classified into 0-Filled Chunk or 1-Filled Chunk, it is called "Literal Chunk".

Switch Position: If the bit before the position is different from current one, the bit position is called "Switch Position". Specially, SPLWAH takes the position of first "1" bit in a Literal Chunk as the first Switch Position. Obviously, each Switch Position occupies 5 bits to represent the number from 1 to 31.

Simple Chunk: If a Literal Chunk can be represented by no more than four Switch Positions, it is called "Simple Chunk".

C. Definitions for Codewords

After the categorization of chunks, we begin to encode the bitmap roughly into the codewords as shown below:

0-Fill: If there are some continuous 0-Filled Chunks, SPLWAH replaces them with a 0-Fill codeword which indicates the number of the replaced chunks.

1-Fill: If there are some continuous 1-Filled Chunks, SPLWAH replaces them with a 1-Fill codeword which indicates the number of the replaced chunks.

Obviously, 0-Fill and 1-Fill are two types of *Fill* which is similar to WAH [8].

Furthermore, the 2-tuple codewords are shown as below:

FS: For a continuous 2-tuple in the sequence, if the first element is a Fill and the second element is a Simple Chunk, this 2-tuple is encoded into a FS codeword, including 0-Fill-S and 1-Fill-S.

SF: For a continuous 2-tuple in the sequence, if the first element is a Simple Chunk and the second element is a Fill,

this 2-tuple is encoded into a SF codeword, including S-0-Fill and S-1-Fill.

At last, the 3-tuple codewords are shown as follows:

FSF: For a continuous 3-tuple in the sequence, if the first and the third elements are both Fill codewords and the second element is a Simple Chunk with no more than 2 Switch Positions, this 3-tuple is encoded into a FSF codeword, including 0-Fill-S-0-Fill, 0-Fill-S-1-Fill, 1-Fill-S-0-Fill and 1-Fill-S-1-Fill.

SFS: For a continuous 3-tuple in the sequence, if the first and the third elements are Simple Chunks and the second element is a Fill codeword, this 3-tuple is encoded into a SFS codeword, including S-0-Fill-S and S-1-Fill-S.

Literal: If a Literal Chunk survives after the encoding procedure above, it is called a Literal codeword with a "0" bit added as shown in Fig. 4.

For easy understanding, let symbol [F] be Fill codeword, [L] be Literal codeword.

So far, the whole process of SPLWAH compression has finished. The result of the encoding scheme consists of Fill, Literal, FS, SF, FSF and SFS codewords.

D. Bit-Represented Codebook

In this subsection, the final result of every codeword is represented by 4 bytes. The first four bits of word are used as a header of codeword. The details are shown as follows:

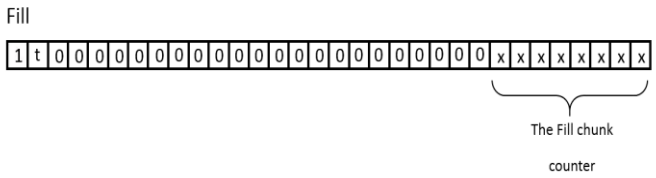


Fig. 3. A Fill codeword.

The first bit of Fill codeword is set to "1" and the second bit ("*t*") is used to encode Fill codeword type ("0" means 0-Fill codeword while "1" means 1-Fill codeword). Only the lowest byte will be used because of the 4,096-bit-long segment in the later experiments as shown in Fig. 3. However, it is easy to find that a Fill codeword actually has 23 bits for storing a counter.

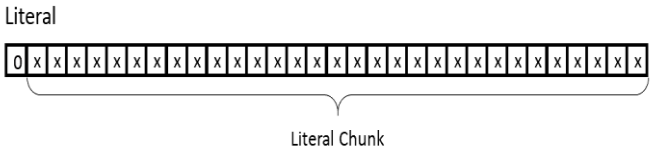
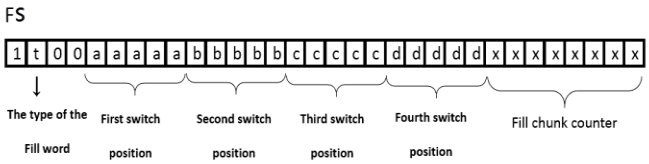


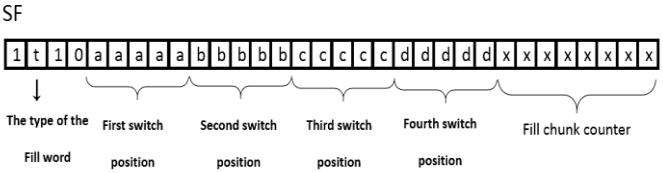
Fig. 4. A Literal codeword.

For Literal, a "0" bit is added before the 31 bits as the flag for identifying as shown in Fig. 4.

For FS, SF, FSF and SFS codewords, the third and fourth bits of the codeword are used to identify the types of the codewords. Especially, the first Switch Position can't be zero so that FS codeword can be distinguished from Fill codeword. However, the second, third and fourth Switch Positions are optional, which is the reason why the number of Switch Positions used to represent Simple Chunk is no more than four.



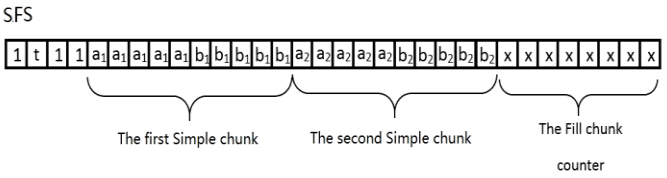
a. FS codeword.



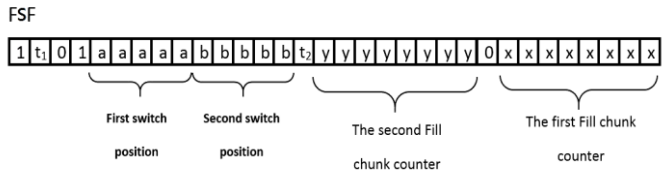
b. SF codeword.

Fig. 5. FS and SF for the 2-tuple codewords.

Fig. 5 shows the difference between FS and SF codeword. The third bit represents the type of 2-tuple codewords ("0" means FS codeword and "1" means SF codeword) with the fourth bit set to "0". The second bit ("*t*") is used to encode the type of the Fill codeword ("1" means 1-Fill codeword and "0" means 0-Fill codeword).



a. SFS codeword.



b. FSF codeword.

Fig. 6. FSF and SFS for 3-tuple codewords.

As shown in Fig. 6, the 3-tuple codewords make full use of bits. Actually, the bits to represent the third and the fourth Switch Positions in 2-tuple codewords are used to carry a new tuple with the fourth bit set to "1" ("11" means a SFS codeword and "01" means a FSF codeword considered with the third bit). For FSF codeword, the 15^{th} bit ("t_2") of the codeword represents the type of the second Fill codeword while the following 8 bits are used as the counter of the second Fill codeword (Fig. 6. b), which is more adaptive to encode Fill codeword.

V. FINITE STATE MACHINE REPESENTATION FOR SPLWAH

SPLWAH encoding can be regarded as a re-encoding based on the result of the WAH encoding scheme. The re-encoding process can also be described as a finite state machine (FSM) in coding theory [28].

Firstly, two types of codewords from WAH are defined as follows:

1) *Fill: 0-Fill or 1-Fill codeword.*
2) *Literal: Literal codeword which contains Simple Chunk in this case.*

The FSM model of the encoding procedure with SPLWAH is shown in Fig. 7, where "Start" is the start state of a bitmap index compression procedure. The meanings of symbols are also defined as follows:

1) *Symbol "F" stands for codeword Fill.*
2) *Symbol "S" stands for Simple Chunk.*
3) *Symbol "L" stands for codeword Literal.*

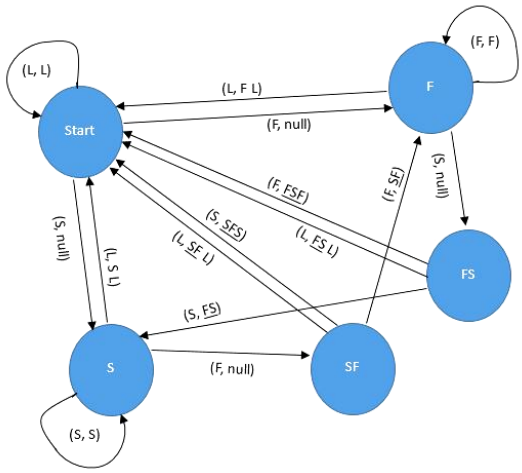


Fig. 7. Finite state machine model of SPLWAH.

The pair (x, y) labels the edge to stand for an action taken at a shift of states, which means when x is the input from WAH encoding scheme, the state moves along the corresponding edge and y is the output to the final result. Particularly, if y equals "null", it outputs nothing. If the output is a 2-tuple codeword or a 3-tuple codeword, e.g. FS codeword and FSF codeword, y is one symbol like "FS" or "FSF" underlined. If the outputs are two codewords, e.g. a Fill codeword and a Literal codeword, y consists of two symbols like "F L".

According to the FSM, a hardware implementation of SPLWAH can be easily designed and implement in a FPGA platform.

VI. EXPERIMENTS AND RESULTS

A. Input / Output Data

In the experiments, the real network flow data from CAIDA is parsed using *libpcap library*. Internet trace is collected from a core router and anonymized by CAIDA, which is widely accepted for the experiments for Internet traffic measurement. The fields of source IP, source port, destination IP, destination port and protocol ID are extracted from the *pcap* archive, and are saved into a plain text file in columnar way.

The record ID of a file is the same as the Row ID in the bitmap as shown in Fig. 1, which means a row is corresponding to a particular record. A row in the file is in the form of *<SRC_IP SRC_PORT DEST_IP DEST_PORT PROTOCOL>*. The total space consumption for a row is 14 bytes (4 bytes for *src_ip*, 2 bytes for *src_port*, 4 bytes for *dest_ip*, 2 bytes for *dest_port*, 4 bytes for *dest_ip*, 2 bytes for *dest_port*, 2 bytes for *protocol*). Generally, the input data is 14 vectors of bytes.

All in all, the output data for one vector will be one bitmap with 256 bit sequences as a column. For 14 vectors, there will be totally 14 bitmaps, or 3584 (14*256) bit sequences.

B. Experiments

In this subsection, the compression process of SPLWAH is presented. The whole process is roughly divided into three steps, all of which adopt bitwise operations with high performance.

1) Step 1. Create a bitmap

Firstly, a vector is created to store the Record ID of each input. There are totally 13,581,810 packets in this trace. And then we reorder packets with the mechanism based on the principle of locality-based hashing used in [15].

After sorting, a sequence of tuples like (*input*, *Record ID*), where the input data listed in ascending order are obtained. And then, the Record ID is converted into a pair (*Chunk ID*, *offset*).

Now the output data can be easily created by the Chunk ID and offset including the 14 uncompressed bitmaps with 3,584 bit sequences which are divided into 31-bit-long chunks, corresponding to the word length of the CPU architecture. As mentioned before, the bitmap is compressed in each column with a fixed block size of 3968 (128*31) rows as a segment in each loop which is also used in [15], generating a series of result files.

After the first step, bit sequences can be compressed in parallel.

2) Step 2. Merging (WAH Encoding)

The merging step merges the adjacent homogenous Filled Chunks into a Fill codeword. And the bit sequence is divided into 1-Fill, 0-Fill, Literal Chunk (contains Simple Chunk) as mentioned in [8-9].

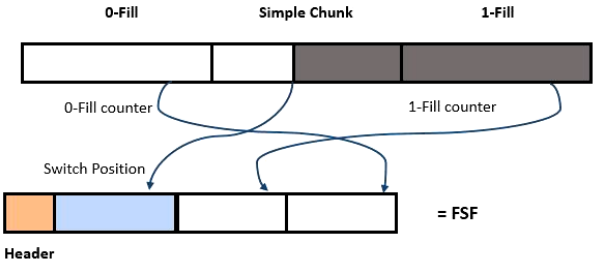


Fig. 8. An example of FSF codeword.

3) Step 3. Combining

After the merging step, the bit sequence has been divided into words. According to SPLWAH, the result of the *step 2* can be compressed with the combination of 2-tuple and 3-tuple

codewords. The rough algorithm is shown in Algorithm 1. And a more detailed example is shown in Fig. 8.

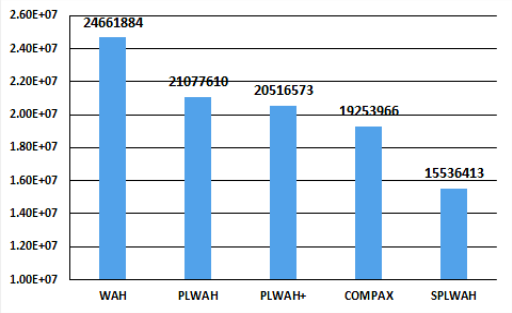


Fig. 9. The space consumption of the algorithms.

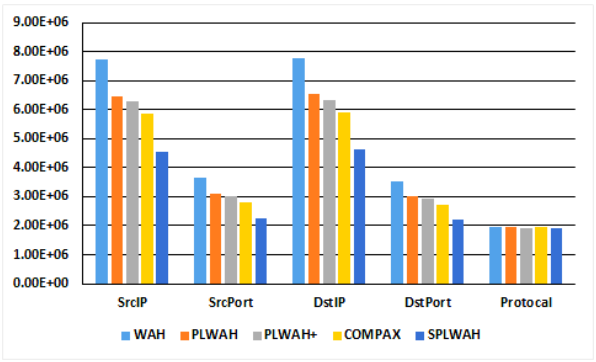


Fig. 10. The compression result of the algorithms.

Algorithm 1: Combining(*B*, *n*)
Require: the result of WAH Encoding *B* and the *length* n
Ensure: Compressed Bitmap *C*

```
1:  length ← 0
2:  for (i ← 0 to n − 2 ){
4:  if( Combinable(B[i], B[i+1], B[i+2]))
3:  // can be combined as a 3-tuple codeword
5:      C[length]=Combined( B[i] , B[i+1] , B[i+2] )
6:      ++i
7:  else if( Combinable(B[i] and B[i+1]))
8:  // can be combined as a 2-tuple codeword
8:      C[length]=Combined( B[i],B[i+1])
9:      ++i
10: else
11:     C[length]=B[i]
12: end if
13:     ++i
14:     ++length
15: end for
```

"*Combinable*" and "*Combined*" referred in the algorithm are both functions which can classify the types of codewords based on WAH Encoding and do Combine procedure.

C. Results

The length in *Dword* (4 bytes) of the compressed result is shown in Fig. 9. There are totally 15,536,413 Dwords in the final compressed files. The original data size is 13,581,810 multiplied by 14 bytes, equaling to 47,536,335 Dwords. Then the compression ratio is roughly 32%.

Besides, Fig. 9 shows that the result of SPLWAH reduces about 24.3% of the storage of PLWAH+ and 37.0% of the storage of the WAH.

It is obvious that COMPAX performs better than PLWAH and PLWAH+ when it handles sorted data as shown in Fig. 10. However, SPLWAH makes up for the shortcomings of the PLWAH+ and reduce the storage about 23.9% compared with COMPAX.

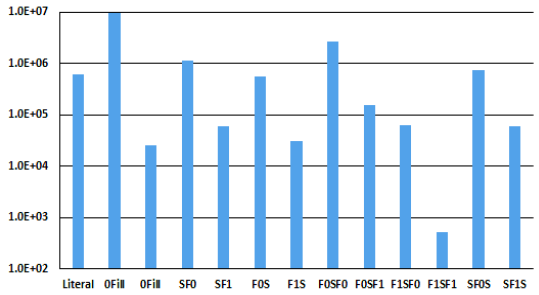


Fig. 11. The distribution of codeword(in logarithmic scale).

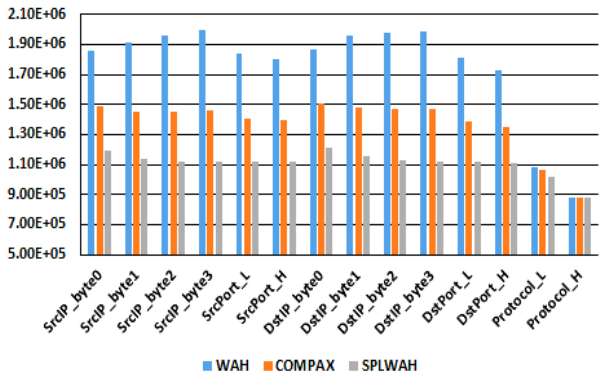


Fig. 12. The compression result of current state-of-the-art algorithms.

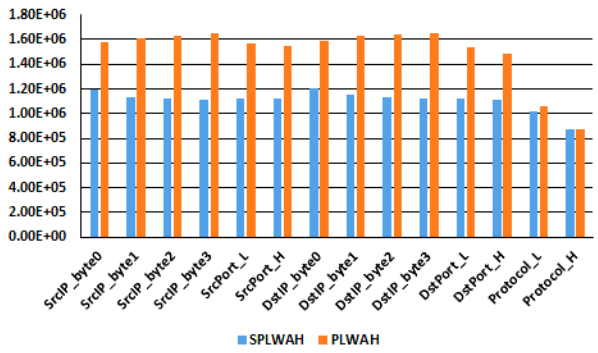


Fig. 13. The compression result of SPLWAH compared with PLWAH.

As shown in Fig. 10, PLWAH, PLWAH+ and COMPAX, which can be regarded as re-encoding based on WAH, all perform well in the bitmap index compression procedure. However, SPLWAH outperforms these algorithms in both storage and performance perspective.

The statistics distribution of each codeword in all compressed bitmaps is given in Fig. 11. From left to right, the labels in the X-axis represents *Literal*, *0-Fill*, *1-Fill*, *S-0-Fill*, *S-1-Fill*, *0-Fill-S*, *1-Fill-S*, *0-Fill-S-0-Fill*, *0-Fill-S-1-Fill*, *1-Fill-S-0-Fill*, *1-Fill-S-1-Fill*, *S-0-Fill-S* and *S-1-Fill-S* codewords.

From Fig. 11, it is obvious that the definitions of the 2-tuple and 3-tuple codewords in SPLWAH encoding scheme are

much more useful in the compression procedure than PLWAH and COMPAX.

D. Insight of Results

As shown in Fig. 12, it is easy to see that SPLWAH is more adaptive to the dense bitmap index, especially for sorted data, and thus improves the compression ratio directly.

As shown in Fig. 13, SPLWAH improves PLWAH in this case with about a 26.3% reduction in storage space.

VII. Conclusion and Future Work

In this paper, we present SPLWAH, a bitmap index compression scheme that compresses bitmaps better than both PLWAH and COMPAX. According to the statistics of Internet traffic data, the clustered effect of ones is quite obvious in generated bitmap indexes because traffic data are composed of flows with the same five-tuple. As an enhanced PLWAH, SPLWAH adapts well to this data characteristic with new designed codebook. Based on real Internet traffic data from CAIDA, we also conduct several performance evaluation experiments. The results show that SPLWAH reduces the space consumption with a factor of 20% or more without incurring extra encoding and decoding cost. SPLWAH can be easily applied into query for Internet traffic data. In general, this work also shows that the design space in bitmap index is still a fruitful unknown frontier and worth to further explore to adapt to the different emerging data spaces, such as machine logs, IoT sensing data and monitoring data etc.

There are still many works need to be done. Future works includes: Optimizing the performance of SPLWAH implementation, applying more parallelism in bitmap index compression and exploiting GPU to accelerate, and giving a more detailed and accurate analytical model of SPLWAH.

Acknowledge

This work is supported in part by Ministry of Science and Technology of China under 973 Program No.2013CB228206 and No.2012CB315801, NSFC No.61233016 and No.61472200 and National Training program of Innovation and Entrepreneurship for Undergraduates with Project No.201410003033 and No.201410003031.

References

[1] Cisco Visual Networking Index Forecast (2011 - 2016).

[2] Wenliang Huang, Zhen Chen, Wenyu Dong, Hang Li, Bin Cao, Junwei Cao. Mobile Internet Big Data Platform in China Unicom. Tsinghua Science and Technology, vol. 19, issues 1, pp. 10-16, 2014.

[3] Bao-hua Yang, Ya-xuan Qi, Yi-bo Xue and Jun Li. Bitmap data structure: Towards high-performance network algorithms designing." Computer Engineering and Applications 45(15), 2009.

[4] Hector Garcia-Molina, Jeffery D. Ullman, Jennifer Widom, Database System implementation, Second Edition, Prentice Hall, 2009.

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

[6] Patrick O'Neil and Dallan Quass. Improved query performance with variant indexes. In ACM Sigmod Record, vol. 26, no. 2, pp. 38-49. ACM, 1997.

[7] Antoshekov G. Byte-aligned bitmap compression. Proc of the Conf on Data Compression. Piscataway, NJ: IEEE, 1994: 363- 098, 1994.

[8] Kesheng Wu, Ekow J. Otoo and Arie 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.

[9] Kesheng Wu, Ekow J. Otoo and Arie Shoshani. Optimizing bitmap indices with efficient compression. ACM Transactions on Database Systems (TODS), 31(1), 1-38, 2006.

[10] Guadalupe Canahuate, Michael Gibas and Hakan Ferhatosmanoglu. Update conscious bitmap indices. 19th IEEE International Conference on Scientific and Statistical Database Management SSBDM'07, 2007.

[11] Michał Stabno, and Robert Wrembel. RLH: Bitmap compression technique based on run-length and Huffman encoding. Information Systems 34, no. 4 (2009): 400-414.

[12] François Deli`ege, Torben Bach 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.

[13] Daniel Lemire, Owen Kaser, Kamel Aouiche. Sorting improves word-aligned bitmap indexes. Data & Knowledge Engineering, 69(1), 3-28, 2010.

[14] Alessandro Colantonio, Roberto Di Pietro. Concise: Compressed 'n' composable integer set. Information Processing Letters, 110(16), 644-650, 2010.

[15] Francesco Fusco, Michail Vlachos, Marc Ph. Stoecklin. Net-fli: on-the-fly compression, archiving and indexing of streaming network traffic. Proceedings of the VLDB Endowment, 3(1-2), 1382-1393, 2010.

[16] Witold Andrzejewski, Robert Wrembel, GPU-WAH: Applying GPUs to compressing bitmap indexes with word aligned hybrid. In Database and Expert Systems Applications, pp. 315-329, Springer, January, 2010.

[17] Witold Andrzejewski, Robert Wrembel, GPU-PLWAH: GPU-based implementation of the PLWAH algorithm for compressing bitmaps. Control & Cybernetics, 40(3), 2011.

[18] Francesco Fusco, Michail Vlachos, Xenofontas Dimitropoulos, and Luca Deri. Indexing million of packets per second using GPUs. In Proceedings of the 2013 conference on Internet measurement conference, pp. 327-332. ACM, 2013.

[19] Fabian Corrales, David Chiu, and Jason Sawin, Variable Length Compression for Bitmap Indices, in DEXA'11, pp. 381-395, Springer-Verlag, 2011.

[20] van Schaik, Sebastiaan J., and Oege 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] Ryan Slechta, Jason Sawin, Ben McCamish, David Chiu and Guadalupe Canahuate, A tunable compression framework for bitmap indices, In Data Engineering (ICDE' 2014), pp. 484-495. IEEE.

[22] 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.

[23] Chambi, Samy, Daniel Lemire, Owen Kaser, and Robert Godin. "Better bitmap performance with Roaring bitmaps." arXiv preprint arXiv:1402.6407 (2014).

[24] Ma, Ge, Zhenhua Guo, Xiu Li, Zhen Chen, Junwei Cao, Yixin Jiang, and Xiaobin Guo. "BreadZip: a combination of network traffic data and bitmap index encoding algorithm." In Systems, Man and Cybernetics (SMC), 2014 IEEE International Conference on, pp. 3235-3240. IEEE, 2014.

[25] Zhen Chen, Yuhao Wen, Junwei Cao, Wenxun Zheng, Jiahui Chang, Yinjun Wu, Ge Ma, Mourad Hakmaoui, Guodong Peng, "A Survey of Bitmap Index Compression Algorithms for Big Data," Tsinghua Science and Technology, 20(1), February 2015.

[26] Jiahui Chang et al., PLWAH+: A Bitmap Index Compressing Scheme based on PLWAH, ACM/IEEE Symposium on Architectures for Networking and Communication System Design, Los Angeles, CA, USA, 2014.

[27] Yuhao Wen, Wen-Liang Huang, Zhen Chen, Ge Ma, Junwei Cao, Wenxun Zheng, Guodong Peng, Shiwei Li, SECOMPAX: a bitmap index compression algorithm, ICCCN HotData'2014, August 2014.

[28] McEliece, Robert. The theory of information and coding. Cambridge University Press, 2002.