2014 IEEE International Conference on Systems, Man, and Cybernetics
October 5-8, 2014, San Diego, CA, USA

BreadZip: a Combination of Network Traffic Data and Bitmap Index Encoding Algorithm

Ge Ma[*‡], Zhenhua Guo[*‡], Xiu Li[*‡], Zhen Chen[§], Junwei Cao[*§], Yixin Jiang[†], Xiaobin Guo[†]
[*]Department of Automation, Tsinghua University, Beijing, China
[§]Research Institute of Information Technology, Tsinghua University, Beijing, China
[‡]Shenzhen Key Laboratory of Information Science and Technology, Tsinghua University, Shenzhen, China
[†]Electric Power Research Institute, CSG
mage69699@gmail.com, zhenhua.guo@sz.tsinghua.edu.cn

***Abstract*—Nowadays, rapid evolution of computers and mobile devices has caused the explosive increase in network traffic. So it becomes more and more necessary to archive network traffic for analyzing network events and a lot of emerging applications. Compression is fundamental for traffic archival solution to save the storage space, and indexing is effective to accelerate search queries for archive of traffic data. In this paper, we propose BreadZip (blocks row-reordering and adaptive index zip), a combination of initial traffic data and index compression. BreadZip has three main advantages. 1) to improve compressing efficiency and reduce memory footprint, traffic data is reordered in sequence and divided into fixed-size blocks; 2) to accelerate queries, an improved bitmap indexes with smaller volume than traditional will be introduced; 3) to save space, both traffic blocks and bitmap indexes are compressed in different simple run-length encoding methods respectively. Finally, our empirical results on network traffic from CAIDA (Cooperative Association for Internet Data Analysis) show that our solution can significantly reduce the volume of traffic data, while simultaneously preserving the ability to perform selectively queries with response times in seconds.**

Keywords—network traffic archives; row-reordering; bitmap indexes; compression; bitmap encoding

I. INTRODUCTION

Since the 1970s when current Internet was initially created, there is a rapid increase on network traffic. Cisco predicts that from 2011 to 2016, the volume of network traffic will quadruple and reach 1.3 ZettaBytes [1]. According to the statistics of China Unicom, mobile traffic increases rapidly with CAGR (compound annual growth rate) of 135%. From the data of China Unicom in 2013, the number of records is more than 2 trillion (2×10^{13}) per month, and the volume of traffic is over 500 TBytes per month.

In many instances the traffic data is an aggregation of records from varieties of sources and collected in a single location. Network traffic is a history of the status of a system. It is extremely useful to "look back in time" and trace the causality of events. Thus, system administrators are reluctant to delete it, opting to archive the traffic in case it is needed in the future. Meanwhile, an increasing number of applications need to efficiently search massive traffic archives. Good network traffic archival solutions are suitable for the applications which need the following conditions:

- record high-speed traffic data efficiently;
- reduce the size of network traffic repositories without severely impacting performance;
- support drill-down queries to find "needles in a haystack";
- the time of search and query should be as short as possible.

In the context of long-term traffic archiving, compression and indexing are essential enabling techniques. Compression substantially reduces the size of traffic archives and indexing directly increases the speed of search and query.

Previous researches show that considerable benefits can be obtained by designing a compression approach based on the data and requirements in specific domains. F. Fusco et al. [3] propose a high-performance approach for high-speed data archiving and retrieval of network traffic flow information, named NET-FLi (NETwork Flow Index). It can be used in networking anomaly investigation, on demand traffic profiling and customized reporting and visualization. TIFAflow [4] (TImemachine FAstbit flow) is used for traffic acquisition and aggregation for forensic analysis, which is software based dection that combines TImemachine [5] with Fastbit [6] indexing to provide flow granularity data storage.

A 10 Gbps network link can reach a maximum of 14.8 million packets per second. It is a big challenge to index these packets in one second. For the majority of network operators, they manage more than one links. Even when they only record the network traffic data, the volume of resulting data repository is rather huge. Thus, for the real-time systems, how to accommodate and index such big data for further analysis remains to be a major challenge.

Our work introduces an integrated method, called blocks row-reordering and adaptive index zip (Breadzip), which is a combination of compressed traffic data and indexes. In particular, the values stored in a traffic archive often share a common prefix (e.g., IP addresses), typically lie within a particular range (e.g., port numbers) and have high repetition level within a relatively short time window. And Breadzip also leverages these characters to optimize the traffic data. Breadzip first row-reorders the traffic blocks and performs a

The research was partly supported by National Natural Science Foundation of China (NSFC, Project No.: 71171121) and National "863" High Technology Research and Development Program of China. (863 Project No.: 2012AA09A408).

978-1-4799-3840-7/14/$31.00 ©2014 IEEE

RLE compression scheme, while builds indexes using an improved bitmap index and compresses the bitmap index sequences into a new format. The main achievements of this work are as follows:

Breadzip is a new design based on row-reordering RLE and bitmap index, as we also improve the traditional bitmap index algorithm to benefit more. Our method shows better flexibility and can be easily implemented. In the end, we compare BreadZip with RasterZip [17] and COMPAX [13] on compression ratio and query response time, which are current state-of-the-art network traffic archival solution and bitmap compressor. And the experimental results show that BreadZip has higher compression ratio and faster query response time.

The rest of this paper is organized as follows. Firstly, related works are listed in Section II. The methodology of BreadZip is described and discussed in Section III. Performance comparison between BreadZip and RasterZip+COMPAX is evaluated in Section IV. Finally, we give a conclusion and future work in Section V.

II. RELATED WORKS

This section summarizes the most notable works related to our reaearch.

A. Columnar Databases

During the development of flow record collection, columnar databases have been identified to be more suitable for compression and indexing [7, 8]. Column-oriented databases, such as MonetDB [9], Big Table [10] and CStore [11], provide several advantages: 1) have more opportunities for compression by storing distinct attributes of traffic records in separate physical columns; 2) reduce the I/O bandwidth for queries that are highly selective at the attribute level, e.g., queries require just a few fields of traffic records to be accessed; 3) allow network operators to be implemented directly over the compressed traffic data. So we also adapt columnar databases, for example, we take IP addresses and port numbers into two columns.

B. Reordering

Some previous works show that reordering database tables before compressing improves the compression rate. D. Lemire and O. Kaser [2] prove that reordering columns by increasing cardinality first often maximizes compression. They stress that selecting the right column order is important as the compressibility can vary substantially. The compression of bitmap indexes also greatly benefits from table sorting. In [12], D. Lemire et al. show that in some situation, the size of bitmap index is reduced by nearly an order of magnitude. In addition, some other algorithms are used to attain sorting objective. In [13], F. Fusco et al. describe a system where bitmap indexes must be compressed on-the-fly to index network traffic. To improve compressibility without sacrificing performance, they cluster the rows using locality sensitive hashing (LSH), so that their system can accommodate the insertion of more than a million records per second.

C. Bitmap Index

A bitmap index is a structure that can accelerates search queries for archival data. Its format is shown in Tab. 1.

Table 1. An example of Bitmap Index.

RowID	Column value	Bitmap Index			
		=1	=2	=3	=4
0	1	1	0	0	0
1	3	0	0	1	0
2	4	0	0	0	1
3	4	0	0	0	1
4	2	0	1	0	0
5	3	0	0	1	0
…	…	…	…	…	…

Tab. 1 shows that a bitmap index is a binary array that indicates the row positions. Since bitmap index uses atomic operations, such as ADD, AND, BITWISE, etc, the retrieval efficiency is much higher. However, bitmap index sacrifices space to save time. And with the number of data rows increasing, bitmap index becomes larger and larger. For this reason, compressed bitmap indexes emerge at the right moment. Compared to tree-based index, compressed bitmap indexes: 1) are more simple and compact; 2) offer high-speed retrieval; 3) are optimized for read-only data.

Research from compressed bitmap index encoding algorithms has been raised and some are widely used. WAH is proposed by K. Wu et al [14]. It introduces the method of dividing 31-bit chunks into fill chunks (all of 31 bits are 0) and literal chunks (the rest), then encodes all fill chunks into a single fill word. And it performs well when there are huge amount of consecutive 0 in the bit sequence. F. Deli`ege et al. propose PLWAH [15]. PLWAH tries to encode the fill word and its next literal word together if the literal word is nearly-identical to a 0-fill word. As a result, PLWAH has higher compression ratio than WAH in general. F. Fusco et al. propose a novel algorithm, named COMPAX [13]. It also tries to combine literal words and fill words after dividing them. However, COMPAX introduces a codebook which enables the algorithm to encode original bit sequence in more paths. Actually, COMPAX divides bit sequence into F (fill), L (literal), FLF (fill-literal-fill), and LFL (literal-fill-literal) types and it is current state-of-art compressed bitmap index encoding. Thus, by experiment we will compare our approach with COMPAX in Sec. IV.

III. METHODOLOGY

In this section, we focus on introducing the main data structure, encoding and compression methods used in BreadZip. In particular, we summary the BreadZip compressing and decompressing procedure in Sec. III-D.

A. Improved Bitmap Index

The traditional bitmap index needs large storage space, which makes bitmap index have plenty of room to be compressed. And there are really several algorithms which have good compression ratio, like PLWAH and COMPAX. But we should notice that every column of bitmap needs to be compressed. For example, if a traffic archive is IP address, its bitmap index has 256 column. We should repeat 256 times

compressing process. Considering the whole traffic data, the workload of compressing bitmap index is non-ignorable. Then, we propose an improved bitmap to relieve this problem substantially, which is described as follows.

The main idea of our method is that using two small bitmaps replaces the traditional large bitmap. The original bitmap is treated as an $m \times n$ matrix of bits. We assume that the size of two small bitmaps is $m \times n_1$ and $m \times n_2$ separately. And we can calculate the values of $n_i (i = 1, 2)$ through Equation 1 and 2.

$$(n_2 - 1)^2 < n \le n_2^2 \quad (1)$$

$$(n_1 - 1) \times n_2 < n \le n_1 \times n_2 \quad (2)$$

If in traditional bitmap the position of m_1 row is n^{m_1} and in small bitmaps the position is $n_1^{m_1}$ and $n_2^{m_1}$, the relationship of them is represented in Equation 3.

$$n^{m_1} = n_1^{m_1} \times n_2 + n_2^{m_1} \quad (3)$$

What's more, Tab. 2 gives us some examples of comparison between traditional bitmap and improved bitmap. And Tab. 3 illustrates a specific example of improved bitmap that $n = 7$.

Table 2. Some examples of comparison.

n	n_1	n_2	Volume Reducing $= \frac{n-(n_1+n_2)}{n}$
4	2	2	0
6	2	3	16.7%
128	11	12	82%
4096	64	64	96.9%

Table 3. An example of improved Bitmap Index.

RowID	Column value	Bitmap Index A			Bitmap Index B		
		=0	=1	=2	=0	=1	=2
0	6	0	0	1	1	0	0
1	5	0	1	0	0	0	1
2	4	0	1	0	0	1	0
3	3	0	1	0	1	0	0
4	2	1	0	0	0	0	1
5	1	1	0	0	0	1	0
6	0	1	0	0	1	0	0
…	…	…	…	…	…	…	…

From the examples in Tab. 2, we easily find that when *n* is small, the volume reduces little, but with *n* increasing, the result is quite considerable, e.g., when *n*=4096, we can save about 96.9% space. In summary, our improved bitmap has the following advantages over traditional bitmap:

- the volume of bitmap is reducing substantially and the more number of column is, the better result of volume reducing is;
- the repeat times are reduced and the workload of bitmap compression becomes less heavy;

Because our improved bitmap replaces large bitmap with two small bitmaps, the operation of retrieval will become double. But the bitmap uses atomic operations, which are pretty fast, and taking the significant trade-off between time and space into consideration, the improved bitmap is extremely deserved to use.

B. Row-reordering

It has been discussed in Sec. II-B that reordering results in significantly better compression ratios not only for traffic data, but also for the data index. And we adopt columnar databases that row-reordering is necessary. Row-reordering could lead to data blocks with lower entropy, reduce bitmap index and archive sizes and accelerate query response time.

What we should notice is that the reordered data is difficult to recover to the initial state without other information. For instance, we can leverage the timestamp attribution to obtain the initial data, but some archived data may have no similar attributions. So we introduce a sorted table to record the original positions, which also use our improved bitmap. And we only reorder the data in each block, instead of the whole traffic data. It is unordered among blocks. Compared to the whole data reordering, data in blocks reordering has some advantages:

- the size of sorted table is not large, so that it is convenient to be compressed;
- the complexity of time and computation is low, so it is realizable to use this approach in real-time systems.

In BreadZip, row-reordering is essential operation and a flowgraph is present in Fig. 1.

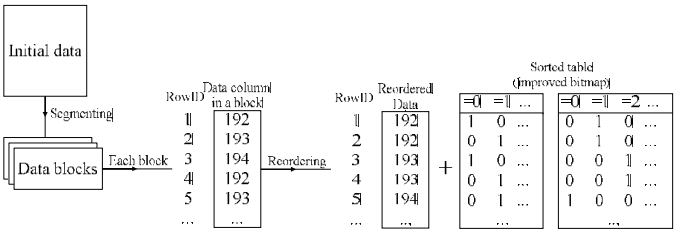


Fig. 1. The flowgraph of row-reordering.

C. Run-length Encoding

We compress each reordered data and bitmap index in a column manner using run-length encoding (RLE). There are three variants of RLE: RLE1 for improved bitmap index, RLE2 for sorted table and RLE3 for reordered data. RLE1 offers a one-sided variant that compresses sequences of both zeros and ones, whereas the two-sided RLE2 focuses on sequences that the number of nonzero bits is fixed and small. RLE3 is the most popular RLE. They offers different codebooks and more details are provided below.

RLE1 is used to compress the improved bitmap. Its codebook has two structures, the difference between which is just the length. Fig. 2 depicts the codebook of RLE1.

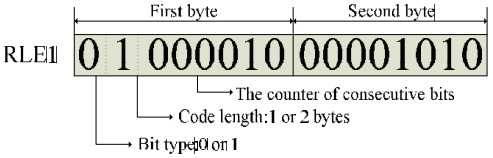


Fig. 2. RLE1 codebook.

From Fig. 2, we design RLE1 codebook that the first bit encodes the current input stream type (0 means that bit type is 0, and 1 means that bit type is 1), the second bit encodes current code length (0 means the length is 1 byte and 1 means the length is 2 bytes) and the remaining 6 or 14 bits are used as counter, counting the number of consecutive zero or nonzero bits. In short, the first bit determines the bit type of counter counting, and the second bit determines the length of counter (when the length is less than 64 bits, the counter uses 6 bits to count; when the length is more than 64bits the counter uses 14 bits to count). So the code in Fig. 2 represents 522 bits consecutive nonzero. And Fig. 3 gives some examples that how an input stream is encoded.

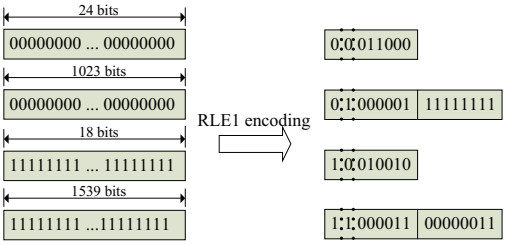


Fig. 3. The examples of RLE1 coding.

Since we set the block size to be 4096 (4K) rows, the number of consecutive zeros or ones is less than 4096 (2^{14}). So the RLE1 encoding doesn't overflow in our blocks.

RLE2 is used to compress the sorted table. The difference between sorted table and bitmap is that each column in bitmap could have uncertain quantity nonzero bits, while in sorted table each column must have fixed small nonzero bits. Namely, the nonzero bits in sorted table are more sparse.

Firstly, we define two formats of word:

- 0-fill-word: when in consecutive 7 bits all the 7 bits are zero, this sequence is called 0-fill-word, e.g., 0000000 ;
- dirty-word: when in consecutive 7 bits there is a nonzero bit or more, this sequence is called dirty-word, e.g., 0001000 .

Then RLE2 also has two structures, one counts the amount of 0-fill-word before the position of a dirty-word and the other records the dirty-word. And we set their length to be a byte (8 bits) and Fig. 4 depicts the codebook of RLE2.

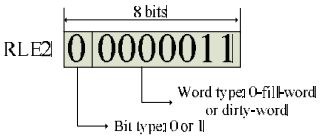


Fig. 4. RLE2 codebook.

In RLE2 codebook, 1st bit encodes the sequence type (0 means that this type as a counter encodes the number of 0-fill-word, and 1 means that this type records a dirty-word) and the rest of bits stands for the word type. Fig. 5 gives some examples how RLE2 runs.

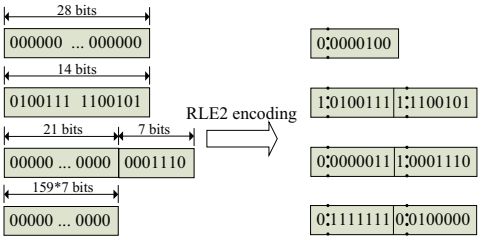


Fig. 5. The examples of RLE2 coding.

In the RLE2 encoding, it may occur that the last bits is less than seven. On this occasion, we fill up them with zero bit until reaching 7 bits. And in the process of decompression, we only choose the top 4096 bits as the initial data.

RLE3 is used to compress reordered data. It is the most common RLE without special parts that when there are consecutive identical data value, they are encoded as a single data value and count respectively, rather than as the original data. In addition, we set the sum length of value and count to be 2 bytes. Fig. 6 give examples of RLE3.

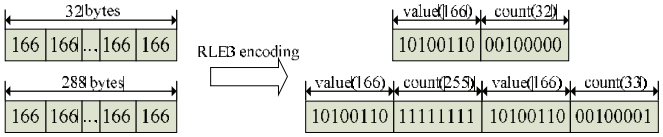


Fig. 6. The examples of RLE3 coding.

Since we set the sum length of value and count to be 2 bytes, it is possible that the RLE3 encoding overflow in our blocks and the second example in Fig. 6 describes how to deal with such situation.

D. BreadZip compressing and decompressing process

An overview of our solution is depicted in Fig. 7. Our solution compressing comprises the following procedures:

Step 1. The traffic archival data is separated into different columnar databases. This step makes the same attribution data into the same place, which increases the opportunities for compression and reduces the I/O bandwidth for queries.

Step 2. In columnar databases, traffic archival data is divided into equal blocks of 4K rows.

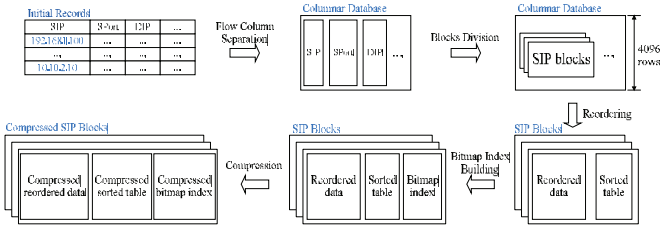
Step 3. Each column is row-reordered in order to achieve better locality and boost compression. And we get an additional sorted table, using our improved bitmap index, to assist us to recover the initial data.

Step 4. Improved bitmap index is built for the reordered data to accelerate search queries response time.

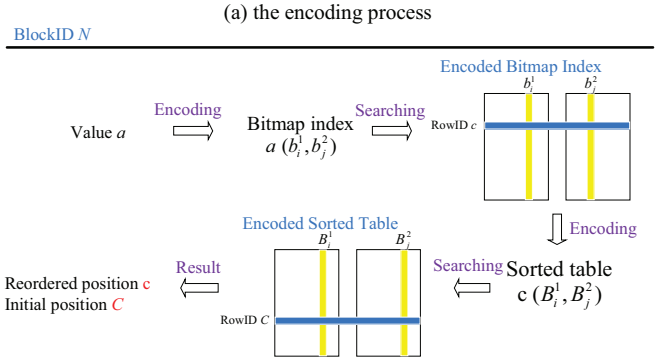
Step 5. For each block, the reordered data, sorted table and bitmap index are compressed into corresponding format by using RLE3, RLE2 and RLE1 respectively.

From above steps, we find that our solution can be easily executed on the fly and in parallel, which improves the performance substantially.

In addition, the decompressing process or the query process is always regarded as the inverse process of compression. For instance, if we want to find out the value *a* in a block (assume blockID is *N*), we should first encode the value *a* corresponding to the bitmap index (anssume b_i^1 and b_j^2). Then we only decompress the b_i^1 and b_j^2 columns of bitmap index instead of the whole columns. And we use bit operations to find out the reordered position (assume the rowID is *c*) of value *a*. Next we encode the rowID *c* corresponding to the sorted table (assume B_i^1 and B_j^2) and we also only decompress the B_i^1 and B_j^2 columns of sorted table. Finally, we can find out the initial position (assume the rowID is *C*) of value *a* through bit operations. The above procedures are depicted in Fig 7 (b). Then through the information of initial position *C* and value *a*, we can build connection of different columnar databases as the data in different columnar databases with the same blockID *N* belongs to the same record, when these data shares a common initial position *C*.



(a) the encoding process



(b) the decoding process

Fig. 7. An overview of Breadzip.

IV. NUMERICAL RESULTS

In this section, we will evaluate the implementation of our approach. We investigate critical performance metrics of the archival and retrieval process. In addition, we compare our approach with RasterZip and COMPAX, which are current state-of-the-art network traffic archival solution and bitmap compressor respectively.

We use a real Internet traffic trace from CAIDA [16] in the evaluation. This Internet traffic trace is anonymized and captured from a core router of backbone network by CAIDA in 2013. There are more than 10 million IPv4 packets in this data set. And each packets is stored as six tuple <SIP, sport, DIP, dport, proto, others>. In our experiments, we pick out source IPs as an example.

In our experiments, we extract the source IPs from 10 million IPv4 packets, meanwhile generate the reordered data, sorted tables and bitmap indexes (we pick out each byte of IP address, four bytes in all, and reorder and convert each byte into bitmap respectively for further encoding). Then we encode them by BreadZip and RasterZip + COMPAX respectively to evaluate the encoding schemes and make comparison between them, i.e., compression ratio and retrieval efficiency. For portability reasons, we determine not to exploit any specific instruction set.

A. Compression Ratio

In order to be compared with BreadZip, we use RasterZip to compress the reordered data, and use COMPAX to encode the sorted tables and indexes using improved bitmap. In Fig. 8, we illustrate simple and clear comparison results with BreadZip and RasterZip+COMPAX. And detailed information is reported in Tab. 4.

Table 4. An example of Bitmap Index.

Field		Initial	RasterZip + COMPAX	BreadZip	Compression Ratio of BreadZip
SIP (CAIDA) /bit	Byte1	80,000,000	4,125,840	3,794,488	4.75%
	Byte2	80,000,000	9,701,600	9,106,168	11.39%
	Byte3	80,000,000	10,572,152	9,639,344	12.05%
	Byte4	80,000,000	11,095,288	9,952,624	12.45%
	sum	320,000,000	35,494,880	32,492,624	10.16%
Sorted Table/bit		5,120,000,000	38,645,565	24,434,312	0.48%
Bitmap Index /bit	Byte1	320,000,000	20,917,126	7,534,242	2.36%
	Byte2	320,000,000	34,009,816	15,734,224	4.92%
	Byte3	320,000,000	38,376,640	16,667,032	5.21%
	Byte4	320,000,000	42,767,560	17,199,200	5.38%
	sum	1,280,000,000	136,071,142	57,134,698	4.47%

Table 5. Disk consumption compared with RASTERZIP+COMPAX.

	SIP	Sorted Table	Bitmap Index
BreadZip	-8.45%	-36.77%	-58.01%

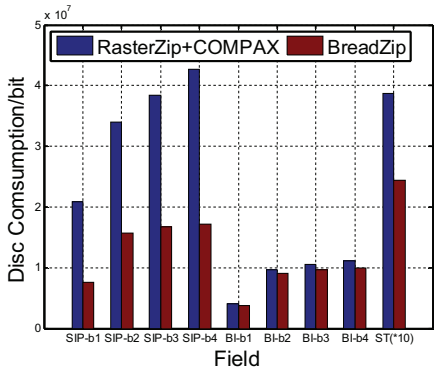


Fig. 8. The examples of RLE3 coding.

From Fig. 8, it clearly shows that BreadZip has a better compression ratio and smaller disk consumption than RasterZip+COMPAX in all three fields.

From Tab. 4 and Tab. 5, in the field of SIP, we note that BreadZip is slightly better than RasterZip, but in the field of sorted table and bitmap index, BreadZip is much better than COMPAX. Then we learn that BreadZip substantially reduces the disk consumption. In particular, the size of BreadZip-encoded sorted tables are just 64% of COMPAX-encoded's and the size of BreadZip-encoded indexes are just 42% of that

of COMPAX-encoded. It is because that BreadZip has a simpler and more compact encoding mechanism than RasterZip+COMPAX.

B. Retrieval Efficiency

Next, we measure the performance during query time, which corresponds to evaluate the retrieval efficiency of both index and archive (have been encoded). For the sake of comparison, we record the cumulative index and archive time. Then we do the following experiments in the dataset CAIDA.

Experiment 1 measures the index time. Each time we randomly choose a block and judge whether this block contains some specific SIPs that we select 1,000 random source IP addresses in advance. And this procedure is repeated 10,000 times. The result is recorded in Tab. 6.

Experiment 2 evaluates the archive time. We choose 10,000 random source IP addresses, search each IP in the whole index and retrieve the initial data from the archive respectively. The result is recorded in Tab. 6.

Table 6. Comparison of retrieval efficiency.

	Index Time	Archive Time
$\frac{\sum t_{BreadZip}}{\sum t_{RasterZip+COMPAX}}$	0.83	0.77

From the Tab. 6, we find that BreadZip provides better retrieval efficiency of not only index time, but also archive time. We summarize the following two reasons: 1) the size of BreadZip-encoding index is smaller; 2) the codebook of BreadZip is more concise. Thus, Breadzip improves the decompressing efficiency and accelerates the speed of search and retrieval.

V. Conclusion and Future Work

In this paper, we propose BreadZip, a combination of initial traffic data archive and index compression, which has the following advantages over traditional methods:

1. BreadZip introduces partial row-reordering. Relative to the overall ordering, it can easily recover the initial data without bringing in severe performance damage. And we introduce the sorted table so that we can finish the recovering process not depending on other attributions, like timestamp. In addition, it is not necessary when the traffic archive shares large amounts of redundant prefix, such as IP addresses in a LAN.

2. BreadZip introduces an improved bitmap, which substantially reduces the volume of bitmap. Meanwhile, the workload of bitmap compression is also reduced.

3. BreadZip has a simple and compact codebook, so it is suitable for compression and decompression, the speed of which is also improved.

4. Compared to the current state-of-the-art RasterZip and COMPAX, BreadZip has better compression ratio and higher retrieval efficiency.

5. BreadZip can be applied to real-time system and other systems, such as library information retrieval system, etc.

We realize that the traffic archiving and retrieving is a very sophisticated system, techniques described in this paper are just the skeleton and some of them need to be further explored. In the future, we will conduct more experiments to make all-round evaluation on the new algorithm. What's more, we will realize BreadZip encoding in GPU to achieve higher performance.

References

[1] Cisco, I. "Cisco visual networking index: Forecast and methodology, 2011--2016." *CISCO White paper* (2012): 2011-2016.

[2] Lemire, Daniel, and Owen Kaser. "Reordering columns for smaller indexes."*Information Sciences* 181.12 (2011): 2550-2570.

[3] Fusco, Francesco, Marc Ph Stoecklin, and Michail Vlachos. "Net-fli: on-the-fly compression, archiving and indexing of streaming network traffic."*Proceedings of the VLDB Endowment* 3.1-2 (2010): 1382-1393.

[4] Chen, Zhen, et al. "TIFAflow: Enhancing traffic archiving system with flow granularity for forensic analysis in network security." *Tsinghua Science and Technology* 18.4 (2013).

[5] Maier, Gregor, et al. "Enriching network security analysis with time travel."*ACM SIGCOMM Computer Communication Review*. Vol. 38. No. 4. ACM, 2008.

[6] Wu, Kesheng, et al. "FastBit: interactively searching massive data." *Journal of Physics: Conference Series*. Vol. 180. No. 1. IOP Publishing, 2009.

[7] Deri, Luca, Valeria Lorenzetti, and Steve Mortimer. "Collection and exploration of large data monitoring sets using bitmap databases." *Traffic Monitoring and Analysis*. Springer Berlin Heidelberg, 2010. 73-86.

[8] Abadi, Daniel J., Samuel R. Madden, and Nabil Hachem. "Column-stores vs. row-stores: how different are they really?." *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. ACM, 2008.

[9] Boncz, Peter A., Martin L. Kersten, and Stefan Manegold. "Breaking the memory wall in MonetDB." *Communications of the ACM* 51.12 (2008): 77-85.

[10] Chang, Fay, et al. "Bigtable: A distributed storage system for structured data." *ACM Transactions on Computer Systems (TOCS)* 26.2 (2008): 4.

[11] Stonebraker, Mike, et al. "C-store: a column-oriented DBMS." *Proceedings of the 31st international conference on Very large data bases*. VLDB Endowment, 2005.

[12] Lemire, Daniel, Owen Kaser, and Kamel Aouiche. "Sorting improves word-aligned bitmap indexes." *Data & Knowledge Engineering* 69.1 (2010): 3-28.

[13] Fusco, Francesco, Michail Vlachos, and Marc Ph Stoecklin. "Real-time creation of bitmap indexes on streaming network data." *The VLDB Journal—The International Journal on Very Large Data Bases* 21.3 (2012): 287-307.

[14] Wu, Kesheng, Ekow J. Otoo, and Arie Shoshani. "Optimizing bitmap indices with efficient compression." *ACM Transactions on Database Systems (TODS)* 31.1 (2006): 1-38.

[15] Deliège, François, and Torben Bach Pedersen. "Position list word aligned hybrid: optimizing space and performance for compressed bitmaps."*Proceedings of the 13th International Conference on Extending Database Technology*. ACM, 2010.

[16] CAIDA. Archipelago measurement infrastructure. http://www.caida.org/

[17] Fusco, Francesco, Michail Vlachos, and Xenofontas Dimitropoulos. "RasterZip: compressing network monitoring data with support for partial decompression." *Proceedings of the 2012 ACM conference on Internet measurement conference*. ACM, 2012.