TSINGHUA SCIENCE AND TECHNOLOGY
ISSN 1007-0214 11/11 pp100-115
Volume 20, Number 1, February 2015

A Survey of Bitmap Index Compression Algorithms for Big Data

Zhen Chen*, Yuhao Wen, Junwei Cao, Wenxun Zheng, Jiahui Chang, Yinjun Wu, Ge Ma, Mourad Hakmaoui, and Guodong Peng

Abstract: With the growing popularity of Internet applications and the widespread use of mobile Internet, Internet traffic has maintained rapid growth over the past two decades. Internet Traffic Archival Systems (ITAS) for packets or flow records have become more and more widely used in network monitoring, network troubleshooting, and user behavior and experience analysis. Among the three key technologies in ITAS, we focus on bitmap index compression algorithm and give a detailed survey in this paper. The current state-of-the-art bitmap index encoding schemes include: BBC, WAH, PLWAH, EWAH, PWAH, CONCISE, COMPAX, VLC, DF-WAH, and VAL-WAH. Based on differences in segmentation, chunking, merge compress, and Near Identical (NI) features, we provide a thorough categorization of the state-of-the-art bitmap index compression algorithms. We also propose some new bitmap index encoding algorithms, such as SECOMPAX, ICX, MASC, and PLWAH+, and present the state diagrams for their encoding algorithms. We then evaluate their CPU and GPU implementations with a real Internet trace from CAIDA. Finally, we summarize and discuss the future direction of bitmap index compression algorithms. Beyond the application in network security and network forensic, bitmap index compression with faster bitwise-logical operations and reduced search space is widely used in analysis in genome data, geographical information system, graph databases, image retrieval, Internet of things, etc. It is expected that bitmap index compression will thrive and be prosperous again in Big Data era since 1980s.

Key words: Internet traffic; big data; traffic archival; network security; bitmap index; bitmap compression algorithm

• Zhen Chen and Junwei Cao are with the Research Institute of Information Technology, Tsinghua University, Beijing 100084, China. E-mail: {zhenchen, jcao}@tsinghua.edu.cn.
• Yuhao Wen is with Department of Electronic Engineering, Tsinghua University, Beijing 100084, China.
• Wenxun Zheng is with Department of Physics, Tsinghua University, Beijing 100084, China.
• Jiahui Chang is with Department of Aerospace Engineering, Tsinghua University, Beijing 100084, China.
• Guodong Peng is with Department of Mechanical Engineering, Tsinghua University, Beijing 100084, China.
• Yinjun Wu and Ge Ma are with Department of Automation, Tsinghua University, Beijing 100084, China.
• Mourad Hakmaoui is with Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China.
• Zhen Chen, Yuhao Wen, Wenxun Zheng, Jiahui Chang, Guodong Peng, Yinjun Wu, Ge Ma, Mourad Hakmaoui, and Junwei Cao are also with Tsinghua National Laboratory for Information Science and Technology (TNList), Tsinghua University, Beijing 100084, China.
∗ To whom correspondence should be addressed.
Manuscript received: 2014-12-05; accepted: 2014-12-28

1 Introduction

1.1 Big data

The Internet has brought with its access to enormous quantities of rapidly changing data in all fields. As the leading search engine, Google provides powerfully customizable search capabilities to individuals[1]. Google records each users' search behavior, including their Web access path and the access time for each page. The system is able to process more than 34 000 requests per second. Important scientific events can generate staggering quantities of data, at enormous rates of growth. For example, the experiments of the European Large Hadron Collider (LHC) produced more than 15 petabytes of data at up to 1.5 gigabytes per second[2].

Network monitors, communication services, sensor networks, and financial services produce unlimited continuously streaming data, growing in real

time. Streaming data is characterized by its growing data volume. Traditional relational database has been unable to meet the requirements of the storage, index, query and analytics of the growing streaming data, which is critical to make timely decisions.

1.2 Internet traffic data

With the commercial popularity of Internet applications and mobile wireless networks, Internet traffic is growing at an accelerating pace. A report from Cisco[3] predicts that Internet traffic will grow four-fold from 2011 to 2016, and reach 1.3 zettabytes (a zettabyte is one trillion gigabytes) in 2016. On the Internet, information is transmitted in packets, and transmitted along different paths in one or more networks, and reassembled at the destination. Internet data transmission is based on TCP/IP protocol, which divides the network packets into IP packets, TCP/UDP packets, and application packets due to different information they contain.

Figure 1 shows the format of original data acquired from a physical link to the Internet. Pcap (Packet capture) is an internet "packet sniffing" API, which

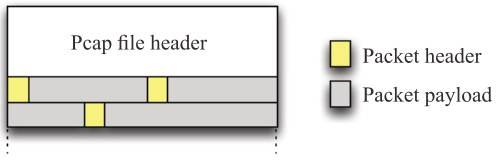


Fig. 1 Pcap file format.

stores captured data as Pcap files. Raw network packets are stored sequentially in Pcap files. The Pcap file header describes the properties of Pcap files (e.g., file size). Packet header contains the description information (e.g., timestamps and size). The packet payload includes the contents of the complete TCP/IP network packets, see Fig. 2. Figure 2a shows an IP packet format, including packet header and payload. It has the following fields: version (4 bits), header length (4 bits), type of services (8 bits), total length (16 bits), identification (16 bits), flags (3 bits), fragment offset (13 bits), Time To Live (TTL) (8 bits), protocol (8 bits), header checksum (16 bits), source IP address (32 bits), and destination IP address (32 bits).

Figure 2b shows the format of a TCP header formats, including the packet header and payload. It has the following fields: source port number (16 bits),

0	4	8	12	16	20	24	28	32
4-bit version	4-bit header len	8-bit type of service		16-bit total length (Bytes)				
16-bit identification				3-bit flags	13-bit fragment offset			
8-bit time to live (TTL)		8-bit protocol		16-bit header checksum				
32-bit source IP address								
32-bit destination IP address								
Options								
Data								

(a) IP packet format

0	15	16	31
16-bit source port number		16-bit destination port number	
32-bit sequence number			
32-bit acknowledgment number			
Data offset	Reserved	URG ACK PSH RST SYN FIN	16-bit window size
16-bit checksum			16-bit urgent pointer
Option			Padding
Data			

(b) TCP packet format

Fig. 2 TCP/IP packet format.

destination port number (16 bits), sequence number (32 bits), acknowledgement number (32 bits), data offset (4 bits), reserved (6 bits), emergency bit URG, confirm bit ACK, reset bit RST, synchronization bit SYN, termination bit FIN, window size (16 bits), checksum (16 bits), urgent pointer (16 bits), option field, and padding field. Network flow records describe network message and transmission characteristics. The record can describe a User Datagram Protocol (UDP) connection or a Transmission Control Protocol (TCP) connection. The flow usually refers to a quintuple of the source IP address, source port, destination IP address, destination port, and protocol.

A network flow record structure obtained from an Internet router is shown in Table 1. It has the following fields: source Autonomous System (AS) number, destination AS number, start and end time, and number of network packets. This is the Netflow V5 flow record format, developed by CISCO; it is the most popular flow record format. Netflow determines which flow packets belong to based on seven fields: source IP address, destination IP address, source port number, destination port number, third-layer protocol type, TOS byte (DSCP), network equipment input (or output), and logical network port (ifIndex). CISCO's Netflow V9, also known as IPFIX (IP Flow Information Export), became an IETF standard (The Internet Engineering Task Force develops and promotes voluntary Internet standards.). It defines how IP flow information is to be formatted and transferred from an exporter to a collector.

Mobile Internet operators obtain Call Detail Records (CDR) traffic information, their format shown in Table 2. It includes terminal attributes, location area code, CI number, service gateway, start time, end time, TCP/IP information, HTTP application information, etc.

Table 1 Network flow record structure (Router).

Field	Remark
SrcIP	Source IP address
SrcPort	Source port
DstIP	Destination IP
DstPort	Destination port
Proto	Layer 3 protocol (e.g., TCP UDP)
TCPflags	Cumulative TCP flags (e.g., SYN ACK FIN)
SrcAS	Source autonomous system number
DestAS	Destination autonomous system number
Octs	# Bytes exchanged in the flow
Pkts	# Packets exchanged in the flow
Start	Flow start time
Duration	Flow duration
others	

Table 2 Flow record format (wireless base station).

Field	Remark
Cell phone number	Without the prefix such as +86,0086,86
Location area code	LAC
CI number	Select the first CI when a network switches
Terminal	IMEI/ IMSI
Start time	YYYY-MM-DD HH:MM:SS.1234567
Duration	In seconds
RAT type	1 represents 3G, 2 represents 2G
Traffic (Byte)	Upstream/ Downstream/total
Terminal IP	Traffic (in bytes)
Port	Source port/ Destination port
APN	3gwap,3gnet,uniwap,uninet,cmwap,cmnet
SGSN/ GGSN IP	First access IP
Http protocol	User Agent/Content-type/URL/Status Code
Others	

1.3 Traffic archiving and retrieval

Network monitoring has been the core function of network management, network fault diagnosis, and network security. In addition to real-time firewalls and intrusion detection systems, traffic-archiving systems are important to network forensic. An Internet Traffic-Archiving System (ITAS) captures packet or flow records for subsequent analysis and processing. Such systems have many important applications.

In addition, statistics from China Unicom[4] shows that current mobile Internet users daily generate more than 30 billion records and 8.4 TB of data in the telecommunications business, resulting in trillions of records and requiring petabyte storage capacity. With mobile Internet users doubling about every six months, the number of Internet records they generate will further increase[5]. Network security has become a major challenge, and technologies such as intrusion detection, signature detection, and security scanning technology have arisen to prevent network attacks[6]. But the volume of Internet traffic data has exceeded current real-time detection and analysis capabilities. It has therefore become necessary to capture Internet traffic for forensic analysis[7–9].

This paper is organized as follows: The structure and function of traffic archive systems is introduced in Section 2. Section 3 describes the key technologies of packet capture, compression, and bitmap index

encoding in network flow archiving systems, and gives a detailed survey on bitmap index compression technology. Section 4 gives an outlook on the wide use of bitmap index technology. We conclude the paper and discuss future work in Section 5.

2 ITAS

2.1 ITAS structures and functions

ITAS, typically includes traffic data acquisition, index storage, and query processing. There are two categories of flow data-acquisition, corresponding to two types of data: (1) packet-level network data and (2) flow-level network data. A typical traffic-archiving system structure is shown in Fig. 3.

Figure 3 describes a typical traffic archiving system. After being captured, network data can be routed to an index module in real time, or stored to disk for subsequent processing. The index module updates the index with the new data, typically compressing it to reduce storage requirement. The indices are stored by rows or columns after indices compression. In order to reduce storage overhead, we adopt the method of storage compression in the process of storage, such as zip and LZO. When a query arrives, the query processor looks up the current index and returns the corresponding result.

2.2 Key technologies in ITAS

2.2.1 High-performance packet/flow capture technologies

Internet traffic's volume is very large, and still growing. There are tens of millions of 64-byte packets needed to deal with each second and similar orders of magnitude of flow records from routers. In a 10 Gbps link, for example, the network will reach 14 million packets per second with 64 bytes per packet. Even after aggregation, there are still millions per second. Though only dealing with flow data, a backbone router's link of telecom operators would generate millions of records per second which will reach 30 billion per day[10–13]. Therefore, how to capture and store arriving packets and flows in real time is a major challenge.

2.2.2 High-performance packet/flow storage technologies

In ITAS, packets and flow records are stored in relational databases by rows, and this consumes large storage. Current distributed database systems have begun to store the data in columns, compressing it to reduce storage overhead. A popular compression method is LZO (Lempel-Ziv-Oberhumer), known for its focus on compression ratio. Other new methods such as RasterZip and BreadZip[14] are proposed for compression ratio and query performance.

2.2.3 High-performance packet/flow indexing technologies

The accepted approach to indexing large volume of data is to employ an inverted index, which is widely used, for example, in search engines. However, a more efficient approach is using bitmap index which is a special case of inverted index also widely used by search engines. Bitmap Index is very efficient in dealing with the network flow queries, especially forensic analysis. The "index space" for an enormous datastore will explode; consequently, companies such as Google have paid much attention to index compression[15, 16]. In order to create an efficient index, the index space should be fragmented (i.e., shard or segmentation), and the index file should be compressed at the same time. Bitmap compression algorithm is an effective method for compressing the bitmap index, to solve the

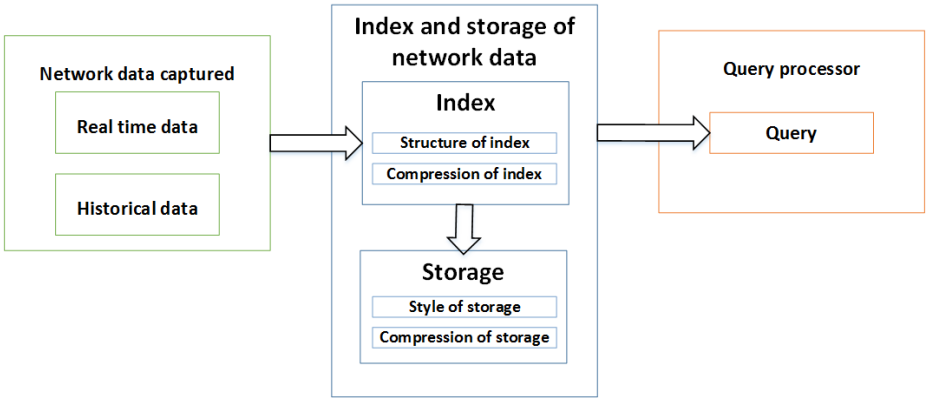


Fig. 3 The structure of an Internet traffic-archiving system.

index-space explosion issue.

2.3 Categories of ITAS

We compare and summarize the features of Internet traffic-archiving systems in Table 3. Raw data can be stored in a database by rows or by columns. In bitmap indexes, the usual choice is "by columns". For example, NET-FLi, an efficient on-the-fly compression, archiving and indexing system for streaming network traffic, stores data in columns, and uses the LZO compression algorithm to reduce the storage size. Traffic archiving entails packet capture, storage, indexing, and querying. Different traffic-archiving systems focus on different aspects. This paper takes the bitmap index compression algorithm as a starting point for understanding traffic-archiving systems.

3 Key Technologies of Bitmap Index Compression Coding

Efficient indexing of network packets or flow is central to traffic archiving systems. Indexing of traffic data has the following characteristic: (1) Large volume of data: The number of index messages is massive, even for brief periods. (2) High rates of incoming data: To keep up with the rate of packet influx, systems must be highly efficient. (3) Fixed data structure: The index information for each network packet has a fixed format with a fixed length. (4) Appending without modification: Network packet index information will increase only. Once the information is generated, it can't be changed. (5) High redundancy: Data items of network data are frequently repeated.

Due to the characteristics of network flow data, relational databases are not appropriate for this task. Traditional relational databases are optimized for handling frequent changes. They use B or B+ trees, which are not particularly suitable for indexing traffic data.

Another common technology of large-data retrieval systems is the inverted index. The core data structure of the inverted index is the posting list. Sequence of integers of a KEY is stored in inverted lists, such as a timestamp and offset, and the most typical KEY is the position list, which shows the position where a word appears. Due to the characteristics of network flow data, net flow archiving systems use bitmap indexing rather than inverted indexes.

3.1 Introduction to bitmap indexing

The concept of bitmap index was first introduced by Spiegler and Maayan in 1985[29]. The first commercial database product was published to implement a bitmap index in 1987[30–35]. It uses a sequence of bits to indicate the the presence or absence of an item in the indexed data. With bitmap indexing, logical operations, such as AND, OR, NOT, and XOR, can be used to answer complex queries. Bitmap index once was designed for scientific data and database, which is usually generated by scientific instruments or scientific simulation. Scientific data are extremely large and without further modification change. Bitmap index databases solve the problem on how to quickly identify a small amount of selected data in a mass of scientific data, while the traditional relational database is not suitable for this work. The technologies used in bitmap index databases are bitmap indexing, bitmap compression, and classification. In bitmap index database, the data are stored in columnar and a corresponding bitmap index is therein. A bitmap index example is shown in Table 4.

Table 3 A comparison of different traffic-archiving systems.

System	Data format		Indexing			Storage	
	Packet	Flow record	Bitmap	Hash	Tree	Column based	Row based
TelegraphCQ[17]	No	Yes		PostgreSQL(DBS)		No	No
Gigascope[18]	No	Yes	No	No	Yes	No	No
MIND[19]	No	Yes	No	Yes	No	No	No
Hyperion[20]	Yes	No	BloomFilter	Multi-level indexing	No	No	No
FastBit[21]+TelegraphCQ	No	Yes		WAH, PostgreSQL(DBS)		Yes	No
TimeMachine[22]	Yes	No	No	Yes	No	No	Yes
NET-FLi	No	Yes	COMPAX	No	No	LZO	No
NetStore[23]	No	Yes	No	No	No	Segmentation	No
RASTERZip[24]	No	Yes	Yes	No	No	Yes	No
PcapIndex[25]	Yes	No	COPMAX	No	No	Yes	No
TifaFlow[26–28]	Yes	No	Yes	No	No	No	No

Table 4 An example of a bitmap index which consists of six tuples (rows). The column as a attribute can have one of four values, 0, 1, 2, and 3.

Row	Column	Bitmap index			
		=1	=2	=3	=4
1	1	1	0	0	0
2	2	0	1	0	0
3	4	0	0	0	1
4	3	0	0	1	0
5	2	0	1	0	0
6	4	0	0	0	1

Currently, classical bitmap index compression algorithms are Byte-aligned Bitmap Compression (BBC)[36], Word-Aligned Hybrid (WAH)[37, 38], Position List Word-Aligned Hybrid (PLWAH)[39], Enhanced Word-Aligned Hybrid (EWAH)[40], COmpressed N Composable Integer SEt (CONCISE)[41], Partitioned Word-Aligned Hybrid (PWAH)[42], COMPressed Adaptive indeX (COMPAX)[43], UCB[44], VLC[45], VAL-WAH[46], RoaringBitmap[47], RLH[48], and DFWAH[49]. There are improved versions of PLWAH (PLWAH with adaptive counter), such as PLWAH+[50] and APLWAH. An improved version of COMPAX (COMPAX + oLSH and COMPAX2) has also presented. Figure 4 shows the various bitmap index compression algorithms in chronological order.

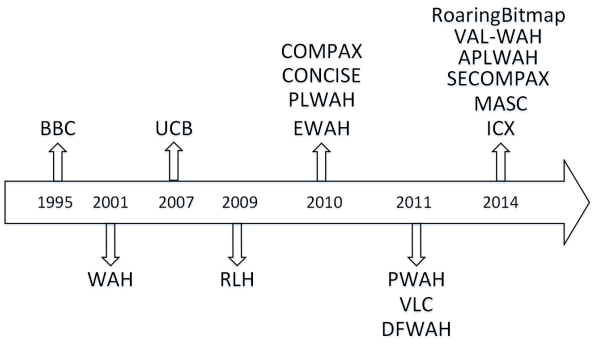


Fig. 4 The advancement of bitmap index compression algorithm.

3.2 Categories of bitmap index compression algorithms

To summarize the characteristics of bitmap index compression algorithms, we compare the bitmap index compression algorithms based on different dimensions, i.e., Segmentation, Chunking, Merge Compress, and Near Identical (NI), as shown in Table 5.

3.3 Bitmap index compression algorithms

3.3.1 BBC

Antoshenkov[36] proposed the byte-aligned bitmap compression method in 1995, the bitmap bytes are classified as gaps and maps. Gaps containing only 0's or only 1's and maps containing a mixture of both. Continuous gaps are encoded by their byte length and a fill bit is used to differ 0 and 1 gaps. Map bytes are directly compiled after the control byte without encoding. A pair (gap, map) is encoded into a single atom which is composed as "control byte + gap length + map". Control byte can be divided into four categories:

(1) The bitmap bytes which contain 0-3 gaps and 0-15 maps can be encoded as follows:

1 [fill bit] [fill length (2 bits)] [tail length (4 bits)].

A fill bit is used to represent all 0's or all 1's; fill length is used to represent the length of gap; and tail length is used to indicate the length of a map.

(2) Bitmap bytes that contain 0-3 gaps, with 1 bit different between the gap and the map, can be encoded as follows:

01 [fill bit] [fill length (2 bits)] [odd bit position (3 bits)].

"Odd-bit position" is the different 1 bit (dirty bit) position between the map and the gap.

(3) Bitmap bytes that contain more than 3 gaps and

Table 5 Bitmap index encoding algorithms.

Methods	Segmentation	Chunking (bit)	Merge Compress		Near Identical			
			Piggyback	LFL/FLF	Dirty bits		Dirty bytes	
					NI-0	NI-1	NI-0	NI-1
BBC	No	8	Yes	No	Yes	Yes	No	No
WAH	No	31	No	No	No	No	No	No
EWAH	No	31	No	No	No	No	No	No
PWAH	No	63	Yes	No	No	No	No	No
PLWAH	No	Yes	Yes	No	Yes	No	No	No
CONCISE	No	Yes	Yes	No	Yes	No	No	No
COMPAX	Yes	31	No	Yes	No	No	No	Yes
SECOMPAX	Yes	31	Yes	Yes	No	No	Yes	Yes
ICX	Yes	31	Yes	Yes	No	No	Yes	Yes
MASC	Yes	No	No	Yes	No	No	Yes	Yes
PLWAH+	Yes	No	Yes	Yes	Yes	Yes	No	No

0-15 maps can be encoded as follows:

001 [fill bit] [tail length (4 bits)].

Gap lengths follow the control byte.

(4) Bitmap bytes that contain more than 3 gaps with 1 bit different between the gap and the map can be encoded as follows:

0001 [fill bit] [odd bit position (3 bits)].

3.3.2 WAH

WAH is the default bitmap index compression algorithm in Fastbit database. It makes the bit sequences into 31 bits (63 bits for WAH 64) as a group. There are two types of group: original Literal, which contains 0 and 1 in 31 bits; and original Fill, which contains all 0's or all 1's in 31 bits. Literal group: The type flag is 0; the remaining 31 bits are the original Literal group. Fill group: Divided into 1-Fill and 0-Fill. The type flag is 1, and the second bit is the sign of Fill type (0-Fill is 0, 1-Fill is 1). The remaining 30 bits are a counter, which indicates the number of consecutive 0-Fill (or 1-Fill) groups.

3.3.3 PLWAH

PLWAH groups the bit sequence into 31-bit units. There are Literal and Fill groups, but there are some changes in compression. In WAH, there are an extra flag to distinct Fill group and Literal group, and each group codes as 32 bits. For Fill groups, the lower 25 bits are a counter. By introducing the definitions of "nearly identical" and "position list", PLWAH further enriches the codebook, increasing the encoding types and improving compression efficiency. A slight improvement over PLWAH is achieved by the method of "PLWAH + adaptive counter". When there is a large number of consecutive 0's or 1's, so that a word can't fully accommodate a counter, this method uses the same type of two consecutive Fill-word, making two counters into a single large counter. The first Fill-word records the 25 least-significant bits, and the second Fill-word records the 25 most-significant bits. Meanwhile, the position list of the first Fill-word is empty, and the position list of the second Fill-word is calculated as usual.

3.3.4 EWAH

EWAH defines a 32-bit field that contains consecutive 0's or 1's as a "clean" segment and defines a 32-bit field that contains mixed 0's and 1's as a "dirty" segment. EWAH algorithm also uses two types of words, where the first type is a 32-bit verbatim word; the second type of word is a marker word, whose first bit indicates which clean word will follow, and 16 bits are used to store the number of clean words. The remaining 15 bits are used to store the number of dirty words following the clean words.

Because EWAH uses only 16 bits to store the number of clean words, it may be less efficient than WAH when there are many consecutive sequences of clean words. Three ways are proposed to alleviate this compression overhead over clean words. First, more than half of the bits can be allocated to encode the runs of clean words. Second, when a marker word indicates a run of 216 clean words, the next word will be used to indicate the number of remaining clean words. Finally, this compression penalty is less relevant when using 64-bit words instead of 32-bit words.

When there are few dirty words in the bitmaps, WAH might be preferable. Even considering EWAH and WAH indexes of similar sizes, each EWAH marker word needs to be accessed three times to determine the running bits and two running lengths, while no word needs to be accessed more than twice with WAH. When there are lots of long runs of dirty words in the bitmaps, EWAH might be preferable. EWAH will access each dirty word at most once, while WAH needs to check the first bit of each dirty word to ascertain it is a dirty word. The EWAH decoder can skip a sequence of dirty words, while the WAH decoder must access them all.

3.3.5 CONCISE

CONCISE is based on WAH. In a compressed 32-bit segment, when the leftmost bit is 1, the following 31 bits are uncompressed; when the leftmost bit is 0, it means Fill, and the second bit indicates the type of Fill. The following 5 bits are the position of a "flipped" bit within the first 31-bit block of the Fill. The remaining 25 bits count the number of 31-blocks that compose the fill minus one. When position bits equal 0 (binary '00000'), the word is a "pure" fill. Otherwise, position bits indicate the bit to switch within the first 31-bit block of the sequence represented by the Fill word. That means that 1 (binary '00001') indicates that it has to flip the rightmost bit, while 31 (binary '11111') indicates that it has to flip the leftmost bit.

3.3.6 PWAH

Based on WAH, PWAH extends word length to 64 bits, and makes a word into P pieces, starting with a P-bit header, which is used to indicate the type of Fill or Literal. The length of Literal can be indicated flexibly to save space.

There are three kinds of PWAH algorithm, PWAH-2, PWAH-4, and PWAH-8. PWAH-2 uses WAH compression coding. In the experiments, PWAH-8 is used more frequently than the others. Combined with the Nuutila algorithm[51], PWAH-8 can be effective in solving the accessibility query problem of large-scale graphs.

3.3.7 COMPAX

There are more codewords in COMPAX codebook. The definitions of Literal and Fill are same as that of WAH and PLWAH. Every 31 bits are a chunk, and the chunks are divided into four groups according to the following features: (1) Literal-Fill-Literal (LFL); (2) Fill-Literal-Fill (FLF); (3) Fill (F); and (4) Literal (L). The COMPAX+oLSH (online-Locality-Sensitive-Hashing) compression method is the same as that of COMPAX, but this compression method needs the reordering of input data stream in advance, which improves compression rate considerably.

3.3.8 SECOMPAX/ICX and MASC/PLWAH+

3.3.8.1 New ideas in bitmap index encoding

SECOMPAX/ICX and MASC/PLWAH+ are proposed based on COMPAX/PLWAH. SECOMPAX/ICX is based on COMPAX2, introduces the concept of NI-1 Literal, and extends the number of dirty bytes to 2; while MASC/PLWAH+ is based on PLWAH and introduces new features from run-length encoding. In terms of "piggyback" (the combination of two symbols) and "FLF/LFL" (the combination of three symbols), the current bitmap indexing encoding algorithm can be classified as two types. According to these categories, the state-of-the-art bitmap index compression algorithms are shown in Fig. 5.

(1) **Scope Extended COMPAX (SECOMPAX)** SECOMPAX[52] is based on COMPAX2, an enhanced version of COMPAX. The differences are as follows: (a) Nearly Identical Literal (NI-L) SECOMPAX extends the "dirty byte" concept to consist of 0-NI-L and 1-NI-L, i.e., a Literal word can be nearly identical to a 0-Fill word or a 1-Fill word. This maintains symmetry in the codebook. (b) Extended LFL: In addition to the 0-NI-L + F + 0-NI-L type, there are three others: 0-NI-L + F + 1-NI-L, 1-NI-L + F + 0-NI-L, and 1-NI-L + F + 1-NI-L. Extended FLF: In addition to the 0F + 0-NI-L + 0F1F+0-NI-L+1F type, there are six others: 0F + 1-NI-L + 0F, 1F + 1-NI-L + 1F; 1F + 1-NI-L + 0F, 0F + 1-NI-L + 1F; 1F + 0-NI-L + 0F, and 0F + 0-NI-L + 1F.

By the inclusion of new codewords, some kinds of origin sequences that COMPAX cannot further compress can be dealt with by SECOMPAX:

(a) In the form of FLF, if two Fill words are of different types, or a Literal word is nearly identical to Fill word, COMPAX does not try to encode them as FLF. COMPAX encodes them as an F, an L, and an F, while SECOMPAX can encode them as FLF. Obviously, in this circumstance, the result encoded by COMPAX is twice as long as those encoded by SECOMPAX.

(b) In the form of LFL, COMPAX can only deal with that both of the two Literal-words are nearly identical to 0-Fill words. If one or both of the Literal words

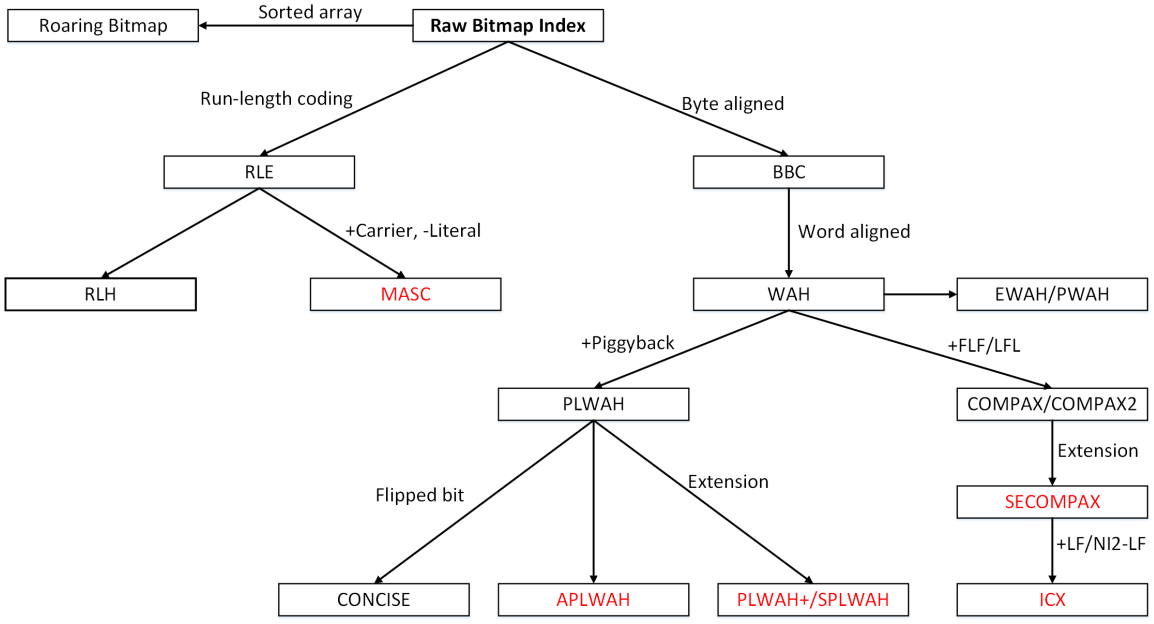


Fig. 5 New ideas in bitmap index encoding.

are identical to a 1-Fill word, COMPAX encodes them as an L-word, an F-word, and an L-word (L+F+L, 3 words in total), not as LFL (1 word in total). However, SECOMPAX can encode all of those circumstances as LFL, which saves a great deal of room compared with COMPAX.

(2) **Improved COMPAX (ICX)** ICX is proposed to further compress the bitmap by considering the possible two dirty-bytes cases, which are not considered in the COMPAX. And we extend PLWAH nearly identically concept to consider one or two dirty bytes case, and represent these cases with Nearly Identical bits in the new codebook. By this new codebook, more possible LF combinations can be encoded, and they can be easily compressed into one word. ICX can achieve a better compression ratio than COMPAX. In the cases where the number of 1's is comparable to the number of 0's, ICX performs especially better than both WAH and COMPAX, since they haven't taken those cases into consideration.

(3) **MAximized Stride with Carrier (MASC)** MASC is proposed to further improve the compression performance without impairing query performance. MASC uses as long a stride size as possible—not limited to 31 bits as in PLWAH and COMPAX—to encode the consecutive zero bits and nonzero bits in a more compact way. MASC records origin bitmap index sequences into a new format.

MASC is a new extended version of PLWAH. The concept "carrier" in MASC and "piggybacked" in PLWAH are similar. However, the carrier can carry at most 30 nonzero bits, while PLWAH can piggyback only a single nonzero bit. In addition, we generalize the concept of Literal word and eventually obsolete it. As a consequence, several (no more than 30) nonzero bits can be carried by the former 0-Fill word and output a 1-carried 0-Fill word, while PLWAH has to encode them in a Literal word, or two Literal words in the worst case, when consecutive nonzero bits are located in two adjacent chunks. Considering zero bits' and nonzero bits' distribution in real data sets, they usually appear in batches—especially after being sorted by the hash value of each record. Thus MASC can perform better than PLWAH.

(4) **Position List Word-Aligned Hybrid Plus (PLWAH+)** We propose the PLWAH+ bitmap compression scheme based on PLWAH. First, we add the definition of an LF word that can piggyback more NI words, which is not considered in PLWAH. According to the experiment, with about a 20% reduction in the amount of Literal words, it can save 3% or more of the storage, compared to PLWAH. And the result shows that the PLWAH+ is more suitable for streaming network traffic. Furthermore, the definition of the NI Chunk is enlarged, which performs better in some cases, where the ratio of set bits is not at a low level. PLWAH+ further classifies NI into two kinds: Nearly Identical 0 Fill word (NI-0 word) and Nearly Identical 1 Fill word (NI-1 word). According to a number of tests, a prediction can be made that PLWAH+ is more suitable for complex databases than PLWAH.

3.3.8.2 Encoding algorithm state diagrams

The state diagrams of SECOMPAX, ICX, MASC, and PLWAH+ are shown in Figs. 6–9. Figure 6 shows the state diagram of the SECOMPAX encoding algorithm. The states of the encoder represent the different word types that have been stored.

Figure 7 introduces the encoding state diagram for ICX. As ICX handles new cases when the number of "dirty bytes" is 2, there are more edges than in SECOMPAX.

Figure 8 shows the state diagram of the MASC encoding algorithm. As MASC encodes the maximum length of consecutive 0's or 1's, the counter is not in multiples of 31, as in WAH.

The encoding state diagram for the PLWAH+ is shown in Fig. 9. The pair (x, y) labels the edge to stand for an action taken at a shift of states, which means

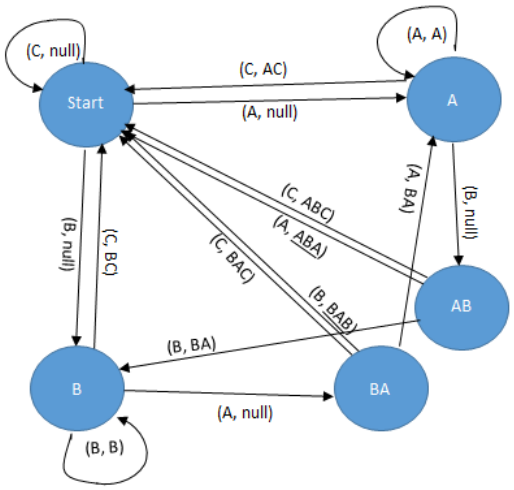
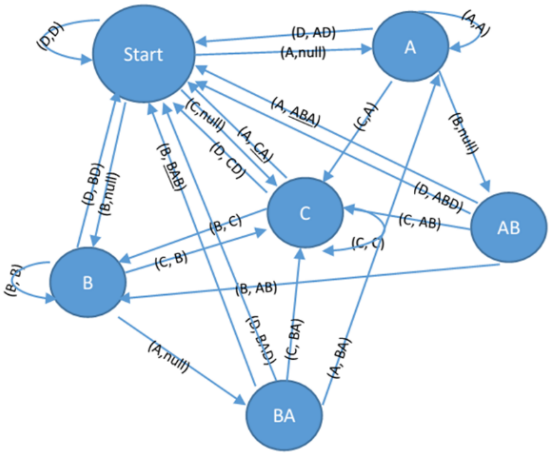


Fig. 6 Encoding algorithm of SECOMPAX. (x, y) means the next character is x, and the output is y except if y=null, output nothing, and if y is underlined, the output is a codeword as a whole.



A, Fill; B, NI-Fill; C, Literal

Fig. 7 Encoding automaton in ICX.

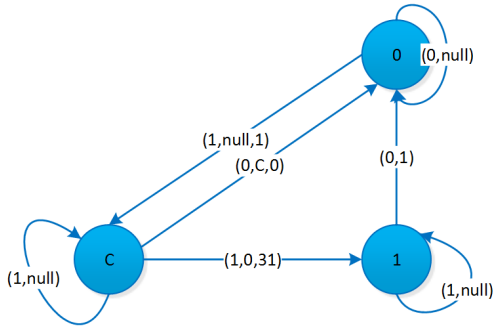


Fig. 8 Encoding automaton in MASC. States 0: 0-Fill; 1: 1-Fill; and C: 1-carried-0-Fill (0-Fill word carried 1's). The meaning of transfer edge (x, y, z) is as follows: x, input bit (0 or 1); y, output bit; z, the counter. If label (x, y) does not occur, the counter will increase by 1, by default.

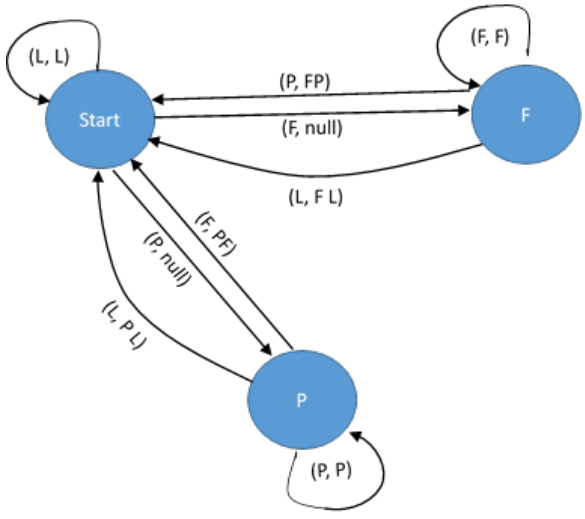


Fig. 9 Encoding automaton in PLWAH+. "Start" is the start state of a compression procedure, symbol "F" stands for codeword Fill word, symbol "P" stands for codeword NI word, and symbol "L" stands for codeword Literal word.

when x is the input from a WAH result, the state moves along the corresponding edge and y is the output to the final result. In particular, if y equals "null", it outputs nothing. If y has one symbol like "FP", it outputs a combination of the two codewords, i.e., FL. If y has two symbols, such as "F" and "L", it outputs two codewords, "F" and "L".

3.3.8.3 Implementation and evaluation

We evaluate SECOMPAX, ICX, MASC, and PLWAH+ with a real Internet traffic trace from CAIDA[53]. This Internet traffic trace was anonymized and captured from a core router by CAIDA at the end of 2013. There are 13 581 181 packets in this trace. First, we reorder packets with the mechanism based on the principle of locality-based hashing used in Ref. [54]. Then we convert five-tuples ⟨SIP, sport, DIP, dport, proto⟩ to bitmaps, and compress the bitmaps in each column with a fixed block size of 4 Kb, which is also used in Ref. [54].

In these experiments, source IP (4 bytes), destination IP (4 bytes), source port (2 bytes), and destination port (2 bytes) are compressed with bitmap index compression algorithms. For SrcIP, there are four bytes, and each byte expands to 256 columnar files. There are 1024 columnar files, which contain 13 581 181 bits (the number of packets in the trace). This is similar for the DstIP and Ports fields.

Figure 10 shows the average size of a compressed columnar file with PLWAH, COMPAX2, and SECOMPAX, ICX, and MASC, where each original columnar file has 13 581 181 bits (the number of packets in the trace). Compared with PLWAH, COMPAX2 and SECOMPAX reduce the size of the index for a source IP address by 6.74%, and for a destination

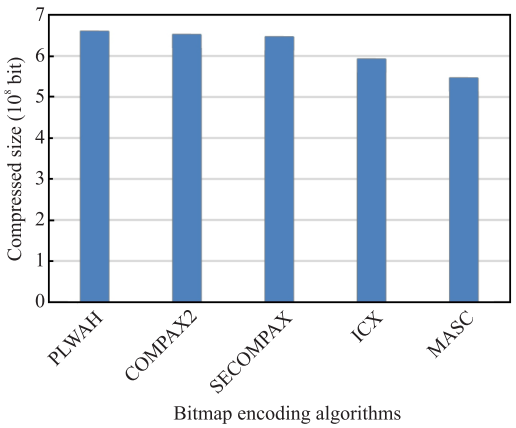


Fig. 10 Bitmap index encoding with PLWAH/COMPAX2/SECOMPAX/ICX/MASC.

IP by 6.05%. While compared with COMPAX2, SECOMPAX reduces the size of the source IP by 4.01%, and destination IP by 3.97%. ICX and MASC show further reduction in compressing the bitmap files. A detailed comparison is illustrated for source IP (4 bytes), destination IP (4 bytes), source port (2 bytes), and destination port (2 bytes) in Figs. 11–14. From Figs. 11–14, it is clear that SECOMPAX/ICX/MASC have a better compression ratio and smaller space consumption than COMPAX2 and PLWAH, especially in Byte 0 in SrcIP and DstIP. Compared with PLWAH, SECOMPAX can reduce the size of the index for SrcIP Byte 0 by 7.62% and DstIP by 8.32%. Among these, MASC has the best performance, as the improvement it shows reaches about 16%–18%.

3.3.8.4. GPU implementation

Usually, the encoding and decoding operation in the process of compressing runs in a CPU. To further accelerate the compression, we propose a GPU-based solution, to offload the indexing and parallelize the encoding operations. As the CPU is also responsible

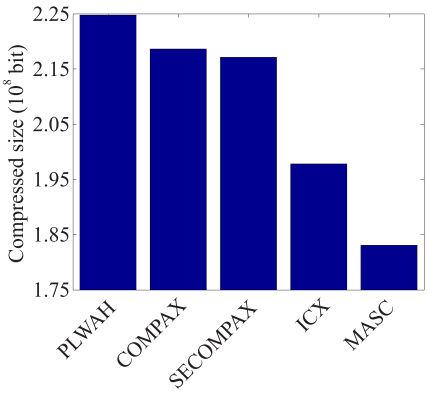


Fig. 11 Size after compression using five encoding schemes in DstIP.

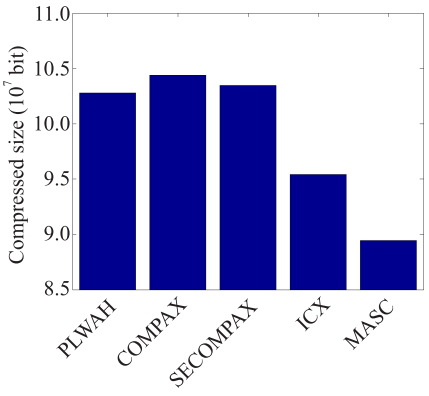


Fig. 12 Size after compression using five encoding schemes in DstPort.

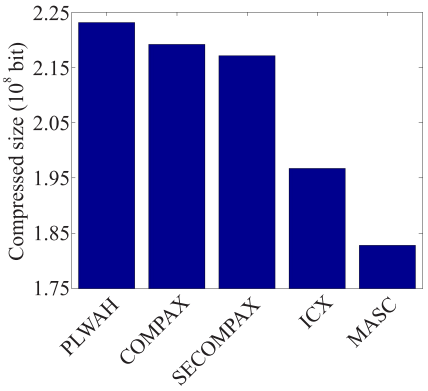


Fig. 13 Size after compression using five encoding schemes in SrcIP.

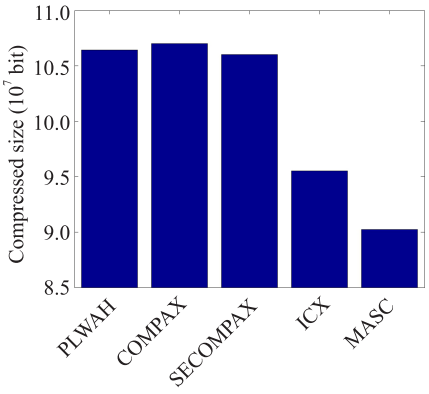


Fig. 14 Size after compression using five encoding schemes in srcPort.

for overall system operation, in consideration of total performance issues, it is better to offload the bitmap indexing operation to a GPU. Andrzejewski and Wrembel[55, 56] introduced GPU-based WAH and PLWAH. However, those implementations could not avoid extending the original data into bitmap form before processing, which increased memory consumption and decreased performance. A new way to compress bitmaps without extending the original data was introduced in Ref. [40], which will be also used in this paper. Especially in Ref. [57], Fusco et al. evaluated the GPU-based WAH and PLWAH with a sequence of random integers to mimic the five-tuples of Internet trace, and proved the potential of a GPU to achieve the speed of indexing a million of packets per second.

GPU-based SECOMPAX, ICX, and MASC algorithms are implemented with Thrust, a C++ library provided with the NVIDIA SDK, designed to enhance code productivity and, more importantly, performance and portability across NVIDIA GPUs. To evaluate the performance of our implementation, we

used a CPU and a GPU with almost the same price: a 3.4-GHz Intel i7-2600K processor with 8 MB of cache and an NVIDIA GTX-760 GPU, fitted in a PCI-e Gen 2.0 slot. The input data we used is an anonymous Internet trace data set from CAIDA of 13 581 810 packets too. We extract their five tuples (source IP, destination IP, source port, destination port, and protocol number). For simulating circumstances in practice, we cut those packets by 3968 (128×31), and created bitmap indexes for 14 (bytes) × 3968 one at a time.

We construct and compress a bitmap index using GPU-MASC, and compare its result with the encoding result of MASC. The memory consumption is the same, and is shown in Fig. 15. However, GPU-MASC can build bitmap indexes for 128×31 packets in 8.057 ms, while CPU-MASC takes 157.3 ms, because MASC cannot encode in parallel on a CPU. Thus, GPU-MASC improves encoding speed by 19.5 times.

Based on Fig. 16, the throughput of GPU-MASC is 492 491 packets per second. However, GPU-MASC constructs and encodes bitmap indexes for all 14 bytes in the five tuples for Internet trace packets, while other algorithms on GPU only construct two bytes of the five tuples, one at a time. So the equivalent throughput for GPU-MASC is 3 447 437 packets per second.

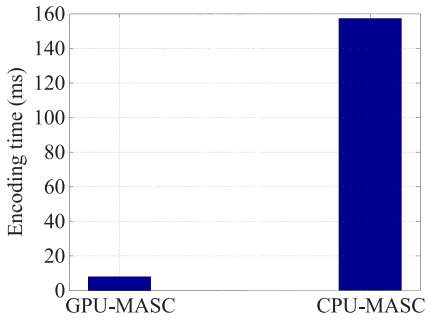


Fig. 15 Encoding-speed comparison between GPU-MASC and CPU-MASC.

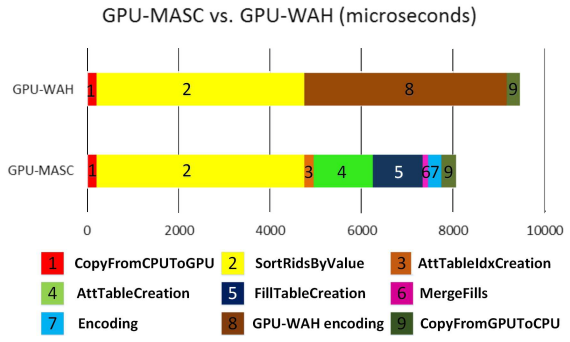


Fig. 16 Encoding time of GPU-WAH and GPU-MASC.

4 Outlooks

Bitmap indexing is a powerful technique to accelerate ad-hoc query in Big Data. With potentially higher compression rates, faster bitwise-logical operations, and reduced search space, bitmap index compression is a booming research area since 1980s and thrives again in Big Data era. Beyond scientific data management[58–60], novel bitmap data representation and compacting of index has a wide usage in many other area, e.g., biological network[61], gene context analysis[62], RFID-based item tracking[63], relational XML twig query processing[64], geographical data warehouses[65], Geographical Information System (GIS)[66], graph databases[67], Content-Based Image Retrieval (CBIR)[68], inverted indexes in search engines[69, 70], and many other data analysis area.

5 Conclusions and Future Works

We introduce and analyze the traffic-archiving systems and the key technologies of bitmap index compression. First, we provide an introduction to classical traffic-archiving systems in recent years; then we present a survey of bitmap index compression algorithms. We summarize bitmap index compression algorithms in terms of Segmentation, Chunking, Merge Compress, and Near Identical. We also propose some new bitmap encoding algorithms, such as SECOMPAX, ICX, MASC, and PLWAH+, and show their state diagrams for encoding procedures. We also evaluate their CPU and GPU implementations with a real Internet trace from CAIDA. Finally, we summarize and discuss the future direction of bitmap index compression algorithms.

With the rapid growth of Internet traffic, bitmap indexing research will encounter new challenges, which must be overcome to design more efficient high-speed indexing technologies. These improved bitmap index compression schemes will have important research value, which will provide powerful technical support for high-performance network-traffic archiving systems.

Acknowledgements

We are grateful to the teachers and students both in NSLAB and QoSlab. The authors would like to thank Prof. Jun Li of NSLAB from RIIT for his guidance.

This work was supported by the National Key Basic Research and Development (973) Program of China (Nos. 2012CB315801 and 2013CB228206), the National

Natural Science Foundation of China A3 Program (No. 61140320), and the National Natural Science Foundation of China (Nos. 61233016 and 61472200). This work was also supported by the National Training Program of Innovation and Entrepreneurship for Undergraduates (Nos. 201410003033 and 201410003031) and Hitachi (China) Research and Development Corporation.

References

[1] Google, Web History, http://www.google.com/psearch, 2008.

[2] Business Intelligence Lowdown, http://www.businessintelligeneelowdown.corn/2007/02/top_10_largest.html, 2008.

[3] Cisco Visual Networking Index: Forecast and Methodology, 20132018, http://www.cisco.com/c/en/us/solutions/collateral/serviceprovider/ip-ngn-ip-next-generationnetwork/white_paper_c11-481360.html, 2014.

[4] 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*, vol. 19, no. 1, pp. 95–101, 2014.

[5] Z. Chen, W. Huang, and J. Cao, *Big Data Engineering for Internet Traffic*, (in Chinese). Beijing, China: Tsinghua University Press, 2014.

[6] R. Lin, B. Wu, F. Yang, Y. Zhao, and J. Hou, An efficient adaptive failure detection mechanism for cloud platform based on volterra series, *Communications, China*, vol. 11, no. 4, pp. 1–12, 2014.

[7] Y. Meng, Z. Luan, and D. Qian, Differentiating data collection for cloud environment monitoring, *Communications, China*, vol. 11, no. 4, pp. 13–24, 2014.

[8] L. Pu, J. Xu, B. Yu, and J. Zhang, Smart cafe: A mobile local computing system based on indoor virtual cloud, *Communications, China*, vol. 11, no. 4, pp. 38–49, 2014.

[9] M. Li, A. Lukyanenko, S. Tarkoma, and A. Yla-Jaaski, MPTCP incast in data center networks, *Communications, China*, vol. 11, no. 4, pp. 25–37, 2014.

[10] L. Rizzo, Device polling support for FreeBSD, http://info.iet.unipi.it/luigi/polling/, 2002.

[11] L. Deri, Improving passive packet capture: Beyond device polling, in *Proceedings of SANE*, 2004, pp. 85–93.

[12] L. Rizzo, Netmap: A novel framework for fast packet I/O, in *Proceedings of the 2012 USENIX Conference on Annual Technical Conference*, USENIX Association, 2012, p. 9.

[13] A. Papadogiannakis, M. Polychronakis, and E. P. Markatos, Scap: Stream-oriented network traffic capture and analysis for high-speed networks, in *Proceedings of the 2013 Conference on Internet Measurement Conference*, 2013, pp. 441–454.

[14] G. Ma, Z. Guo, X. Li, Z. Chen, J. Cao, Y. Jiang, and X. Guo, BreadZip: A combination of network traffic data and bitmap index encoding algorithm, in *Systems, Man and Cybernetics (SMC), 2014 IEEE International Conference on IEEE*, 2014, pp. 3235–3240.

[15] L. A. Barroso, J. Clidaras, and U. Hlzle, The datacenter as a computer: An introduction to the design of warehouse-scale machines, *Synthesis Lectures on Computer Architecture*, 2013, doi:10.2200/S00516ED2V01Y201306CAC024.

[16] J. Dean, Challenges in building large-scale information retrieval systems, in *Proceedings of the Second International Conference on Web Search and Web Data Mining, WSDM 2009*, Barcelona, Spain, 2009.

[17] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, and M. A. Shah, TelegraphCQ: Continuous dataflow processing, in *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data ACM*, 2003, p. 668.

[18] C. Cranor, T. Johnson, O. Spataschek, and V. Shkapenyuk, Gigascope: A stream database for network applications, in *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data ACM*, 2003, pp. 647–651.

[19] X. Li, F. Bian, H. Zhang, C. Diot, R. Govindan, and W. H. Hong, Advanced indexing techniques for widearea network monitoring, in *Data Engineering Workshops, 2005. 21st International Conference on IEEE*, 2005, pp. 1184–1184.

[20] P. Desnoyers and P. J. Shenoy, Hyperion: High volume stream archival for retrospective querying, in *USENIX Annual Technical Conference*, 2007, pp. 45–58.

[21] K. Wu, FastBit: An efficient indexing technology for accelerating data-intensive science, *Journal of Physics: Conference Series*, vol. 16. no. 1, pp. 556–560, 2005.

[22] G. Maier, R. Sommer, H. Dreger, A. Feldmann, V. Paxson, and F. Schneider, Enriching network security analysis with time travel, *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4, pp. 183–194, 2008.

[23] P. Giura and N. Memon, NetStore: An efficient storage infrastructure for network forensics and monitoring, in *Recent Advances in Intrusion Detection*. Springer Berlin, 2010, pp. 277–296.

[24] F. Fusco, M. Vlachos, and X. Dimitropoulos, RasterZip: Compressing network monitoring data with support for partial decompression, in *Proceedings of the 2012 ACM Conference on Internet Measurement Conference*, 2012, pp. 51–64.

[25] F. Fusco, X. Dimitropoulos, M. Vlachos, and L. Deri, PcapIndex: An index for network packet traces with legacy compatibility, *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 1, pp. 47–53, 2012.

[26] J. Li, S. Ding, M. Xu, F. Y. Han, X. Guan, and Z. Chen, TIFA: Enabling real-time querying and storage of massive stream data, in *Networking and Distributed Computing (ICNDC), 2011 Second International Conference on*, 2011, pp. 61–64.

[27] Z. Chen, L. Ruan, J. Cao, Y. Yu, and X. Jiang, TIFAflow: Enhancing traffic archiving system with flow granularity

for forensic analysis in network security, *Tsinghua Science and Technology*, vol. 18, no. 4, pp. 406–417, 2013.

[28] Z. Chen, X. Shi, L. Ruan, F. Xie, and J. Li, High speed traffic archiving system for flow granularity storage and querying, presented at ICCCN 2012 Workshop on PMECT, 2012.

[29] I. Spiegler and R. Maayan, Storage and retrieval considerations of binary data bases, *Information Processing and Management*, vol. 21, no. 3, pp. 233–254, 1985.

[30] P. Cheng, Bitmap index techniques and its research advancement, *Science and Technologies Information*, vol. 27, no. 26, pp. 134–135, 2010.

[31] Z. Huang, W. Lv, and J. Huang, Improved BLAST algorithm based on bitmap indexes and B+ tree, *Computer Engineering and Applications*, vol. 49, no. 11, pp. 118–120, 2013.

[32] B. Yang, Y. Qi, Y. Xue, and J. Li, Bitmap data structure: Towards high-performance network algorithms designing, *Computer Engineering and Applications*, vol. 45, no. 15, pp. 1–5, 2009.

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

[34] P. E. O'Neil, Model 204 architecture and performance, in *High Performance Transaction Systems*. Springer Berlin Heidelberg, 1989, pp. 39–59.

[35] P. O'Neil and D. Quass, Improved query performance with variant indexes, *ACM Sigmod Record*, vol. 26, no. 2, pp. 38–49, 1997.

[36] G. Antoshekov, Byte-aligned bitmap compression, in *Proceedings of the Data Compression Conference (DCC'95)*, 1995, p. 476.

[37] K. Wu, E. J. Otoo, and Arie Shoshani, Compressing bitmap indexes for faster search operations, in *Scientific and Statistical Database Management, 2002. Proceedings. 14th International Conference on*, 2002, pp. 99–108.

[38] K. Wu, E. J. Otoo, and A. Shoshani, Optimizing bitmap indices with efficient compression, *ACM Transactions on Database Systems (TODS)*, vol. 31, no. 1, pp. 1–38, 2006.

[39] 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.

[40] D. Lemire, O. Kaser, and K. Aouiche, Sorting improves word-aligned bitmap indexes, *Data & Knowledge Engineering*, vol. 69, no. 1, pp. 3–28, 2010.

[41] A. Colantonio and R. Di Pietro, Concise: Compressed ncomposable integer set, *Information Processing Letters*, vol. 110, no. 16, pp. 644–650, 2010.

[42] 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*, 2011, pp. 913–924.

[43] F. Fusco, M. P. Stoecklin, and M. Vlachos, Net-fli: Onthe-fly compression, archiving and indexing of streaming network traffic, *Proceedings of the VLDB Endowment*, vol. 3, nos. 1&2, pp. 1382–1393, 2010.

[44] G. Canahuate, M. Gibas, and H. Ferhatosmanoglu, Update conscious bitmap indices, in *19th IEEE International Conference on Scientific and Statistical Database Management SSBDM'07*, 2007.

[45] F. Corrales, D. Chiu, and J. Sawin, Variable length compression for bitmap indices, in *Database and Expert Systems Applications*, A. Hameurlain, S. W. Liddle, K.-D. Schewe, and X. Zhou, eds. Springer Berlin Heidelberg, 2011.

[46] G. Guzun, G. Canahuate, D. Chiu, and J. Sawin, A tunable compression framework for bitmap indices, in *IEEE International Conference on Data Engineering (ICDE 2014)*, 2014, pp. 484–495.

[47] S. Chambi, D. Lemire, O. Kaser, and R. Godin, Better bitmap performance with Roaring bitmaps, arXiv preprint arXiv:1402.6407, 2014.

[48] M. Stabno and R. Wrembel, RLH: Bitmap compression technique based on run-length and Huffman encoding, *Information Systems*, vol. 34, no. 4, pp. 400–414, 2009.

[49] A. Schmidt, D. Kimmig, and M. Beine, A proposal of a new compression scheme of medium-sparse bitmaps, *International Journal on Advances in Software*, vol. 4, nos. 3&4, pp. 401–411, 2011.

[50] 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*, 2014, pp. 257–258.

[51] E. Nuutila, An efficient transitive closure algorithm for cyclic digraphs, *Information Processing Letters*, vol. 52, pp. 207–213, 1994.

[52] Y. Wen, Z. Chen, G. Ma, J. Cao, W. Zheng, G. Peng, and W. L. Huang, SECOMPAX: A bitmap index compression algorithm, in *Computer Communication and Networks (ICCCN), 2014 23rd International Conference on, IEEE*, 2014, pp. 1–7.

[53] M. F. Oberhumer, The Lempel-Ziv-Oberhumer Packer, http://www.lzop.org/, 2010.

[54] F. Reiss, K. Stockinger, K. Wu, A. Shoshani, and J. M. Hellerstein, Enabling real-time querying of live and historical stream data, in *Scientific and Statistical Database Management, 2007. SSBDM'07. 19th International Conference on, IEEE*, 2007, p. 28.

[55] W. Andrzejewski and R. Wrembel, GPU-WAH: Applying GPUs to compressing bitmap indexes with word aligned hybrid, in *Database and Expert Systems Applications*, P. G. Bringas, A. Hameurlain, and G. Quirchmayr, eds. Springer Berlin Heidelberg, 2010, pp. 315–329.

[56] W. Andrzejewski and R. Wrembel, GPU-PLWAH: GPU-based implementation of the PLWAH algorithm for compressing bitmaps, *Control & Cybernetics*, vol. 40, no. 3, pp. 627–650, 2011.

[57] 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*, 2013, pp. 327–332.

[58] S. Lakshminarasimhan, D. A. Boyuka, S. V. Pendse, X. Zou, J. Jenkins, V. Vishwanath, and N. F. Samatova, Scalable *in situ* scientific data encoding for analytical query processing, in *Proceedings of the 22nd International Symposium on High-Performance Parallel and Distributed Computing*, 2013, pp. 1–12.

[59] R. R. Sinha, S. Mitra, and M. Winslett, Bitmap indexes for large scientific data sets: A case study, in *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, 2006, p. 10.

[60] R. R. Sinha, and M. Winslett, Multi-resolution bitmap indexes for scientific data, *ACM Transactions on Database Systems (TODS)*, vol. 32, no. 3, article no. 16, 2007.

[61] Y. Zhang, F. N. Abu-Khzam, N. E. Baldwin, E. J. Chesler, M. A. Langston, and N. F. Samatova, Genome-scale computational approaches to memory-intensive applications in systems biology, in *Proceedings of the ACM/IEEE SC 2005 Conference on Supercomputing'2005*, 2005, p. 12.

[62] A. Romosan, A. Shoshani, K. Wu, V. Markowitz, and K. Mavrommatis, Accelerating gene context analysis using bitmaps, in *Proceedings of the 25th International Conference on Scientific and Statistical Database Management*, 2013, p. 26.

[63] Y. Hu, S. Sundara, T. Chorma, and J. Srinivasan, Supporting RFID-based item tracking applications in oracle DBMS using a bitmap datatype, in *Proceedings of the 31st International Conference on Very Large Data Bases*, 2005, pp. 1140–1151.

[64] K. H. Lee and B. Moon, Bitmap indexes for relational XML twig query processing, in *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, 2009, pp. 465–474.

[65] T. L. Lopes Siqueira, R. R. Ciferri, V. C. Times, and C. D. de Aguiar Ciferri, A spatial bitmap-based index for geographical data warehouses, in *Proceedings of the 2009 ACM Symposium on Applied Computing*, 2009, pp. 1336–1342.

[66] J. Zhang and S. You, Dynamic tiled map services: Supporting query-based visualization of large-scale raster geospatial data, in *Proceedings of the 1st International Conference and Exhibition on Computing for Geospatial Research & Application*, 2010, p. 19.

[67] N. Martínez-Bazan, M. Á. Águila-Lorente, V. Muntés-Mulero, D. Dominguez-Sal, S. Gómez-Villamor, and J. L. Larriba-Pey, Efficient graph management based on bitmap indices, in *Proceedings of the 16th International Database Engineering & Applications Sysmposium*, 2012, pp. 110–119.

[68] G. H. Cha, Bitmap indexing method for complex similarity queries with relevance feedback, in *Proceedings of the 1st ACM International Workshop on Multimedia Databases*, 2003, pp. 55–62.

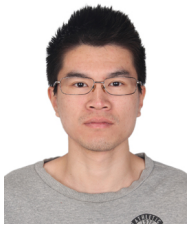
[69] M. Fontoura, M. Gurevich, V. Josifovski, and S. Vassilvitskii, Efficiently encoding term co-occurrences in inverted indexes, in *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*, 2011, pp. 307–316.

[70] D. Dash, J. Rao, N. Megiddo, A. Ailamaki, and G. Lohman, Dynamic faceted search for discovery-driven analysis, in *Proceedings of the 17th ACM Conference on Information and Knowledge Management*, 2008, pp. 3–12.

Zhen Chen is an associate professor in Research Institute of Information Technology in Tsinghua University. He received his BEng and PhD degrees from Xidian University in 1998 and 2004, respectively. He once worked as postdoctoral researcher in Network Institute of Department of Computer Science in Tsinghua University during 2004 to 2006. He is also a visiting scholar in UC Berkeley ICSI in 2006. He joined Research Institute of Information Technology in Tsinghua University since 2006. His research interests include network architecture, computer security, and data analysis. He has published around 80 academic papers.

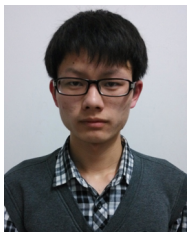
Junwei Cao is currently a professor and deputy director of Research Institute of Information Technology, Tsinghua University, China. He is also the Director of Open Platform and Technology Division, Tsinghua National Laboratory for Information Science and Technology. His research is focused on advanced 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 degree in computer science from University of Warwick, UK, in 2001. He got his MEng and BEng 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.

Yuhao Wen is an undergraduate student studying in Department of Electronic Engineering at Tsinghua University. His research interests include big data and networks.



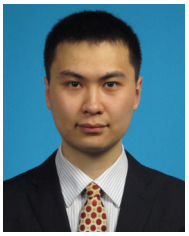
Wenxun Zheng is an undergraduate student studying in Department of Physics at Tsinghua University. His research interests is now on bitmap index compression.

Jiahui Chang is an undergraduate student studying in Department of Aerospace Engineering at Tsinghua University. He majors in Engineering Mechanics.



Guodong Peng is an undergraduate student studying in Department of Mechanical Engineering at Tsinghua University. His research interests include mechanical design and network security.

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



Ge Ma is currently a master student studying in Department of Automation at Tsinghua University. He got his BEng degree from Tsinghua University in 2013. His research interests include future network and bitmap index compressing.

Mourad Hakmaoui is a postgraduate student studying for PhD degree in Department of Computer Science and Technologies at Tsinghua University. He got his master and bachelor degrees from University of Mohammed 5th-Morocco in 2008 and 2012, respectively. His research interests include bitmap indexing algorithms and fast query for Big Data.