ISSN(Online): 2319-8753
ISSN (Print): 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Website: www.ijirset.com

Vol. 6, Issue 5, May 2017

Variable-5bits Algorithm for Compression Inverted Index

Aljaily Mamoun [1] , Faiz Yousif [2]

M.A. Student, Department of Computer Science, Sudan University of Science and Technology, Khartoum, Sudan [1]

Associate Doctor , Department of Computer Science, Sudan University of Science and Technology, Khartoum, Sudan [2]

ABSTRACT. One of the most important characteristics of file compression is to preserve memory with all its temporary and permanent images, in addition to maintaining the speed of performance in the process of compression and decompression and increase the frequency range in the transmission and reception from long distances.
The examined the performance of Variable-Byte compression scheme over sequences of integers. Variable-Byte algorithm which has high compression speed and decompression speed while the algorithm's compression ratio is weak in the case of small integers.
A new compression algorithm - Variable-5bits, has been designed, the ratio of compression is improved from 4 to 6.4 in the case of small integers. In the case of large integers, the Variable-5bits algorithm is roughly equal in the compression ratio with the Variable-Byte algorithm.
A number of experiments were performed to test the compression ratio in the new algorithm in different and varied samples when the range is small between integers, the compression ratio is high and when the range is large between the integers we get a low compression ratio.

KEYWORDS: Compression, Inverted Index, Variable-Byte, Variable-5bits.

I. INTRODUCTION

Information retrieval systems deal with a huge amount of data that needs to be organized as inverted index [1]. The use of inverted indexes and compression techniques is partially accountable for the current performance achievement of web search engines [2]. Large amount of data their performance becomes constrained by the speed at which data can be read or written and need it more storage to store. Compression reduces the physical size of the inverted index, saving storage cost and improves the performance [3], [4]. Fast compression can reduce query response times [5].
While the performance of an information retrieval (IR) system can be enhanced through the compression of its posting list, proposed many compression techniques deal with different types of posting information (document ids, frequencies, positions) [6].
There are some factors which influence compression performance like the number of integers and range of this integer in index. In our study, we focus on the performance of compression applied over sequence of integers. Indexes are mapped to such sequences of integers.

II. RELATED WORK

For an exhaustive review of 32-bit integer compression techniques, we refer the reader to Matteo, Craig and Iadh [7]. Variable byte codec [8] is a byte-aligned, oblivious codec. It uses the 7 lower bits of any byte to store a partial binary representation of the integer x. It then marks the highest bit as 0 if another byte is needed to complete the representation, or as 1 if the representation is complete. For example, 201 is 10000001 01001001. Their finding while this codec may lead to larger representations, it is usually faster than Gamma in term of decompression speed [9].

Veluchamy and Sandanam [10] also carried out an extensive experimental evaluation on synthetic data, they found that Variable-Byte compared to bitwise technique like Rice code, Variable Byte code required a single branching condition for each byte which is more cost-effective in terms of CPU cycles. Moreover, VByte has a poor compression ratio since it requires one full byte to encode small integers.

ISSN(Online): 2319-8753
ISSN (Print): 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Website: www.ijirset.com

Vol. 6, Issue 5, May 2017

III. BACKGROUND

In this section, we first introduce the block-based inverted index that we are using for comparing the two compression techniques (Variable-Byte and Variable-5bits). We then describe variable-byte scheme, before introducing the variable-5bits scheme.

1. Inverted Index Structure

An inverted index [11] is a data structure that stores a list of distinct terms which are found in the collection, this list is called a dictionary, lexicon or a term index. For each term a list of all documents that contain this term is attached, and it is known as the posting list. (Elmasri R., S. Navathe, 2011).

For each unique indexed term, the inverted index contains a posting list, where each posting contains the occurrences information (e.g. frequencies, and positions) for documents that contain the term [7].

Inverted index construction is done by collecting the documents that form the corpus.

Afterwards the preprocessing operation is done on the documents to obtain the vocabulary terms; this term is used to build the forward index (document-term index) by creating a list of the words that are in each document [12].

Postings in inverted lists are usually ordered by increasing **See Fig1.**

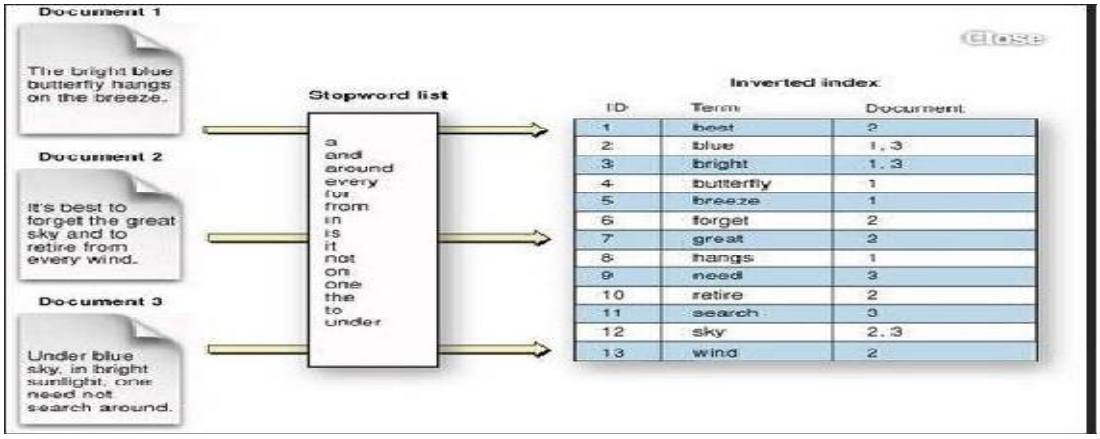

Fig 1 Inverted Index

2. Techniques for Inverted List Compression

An inverted index is composed of inverted lists, each one being an ordered list of integers. The main idea of index compression is to encode the list of integers using as few bits as possible. Instead of storing the raw integer in a 32 bit machine word, the goal is to store each integer using the smallest number of bits possible **See Fig 2.**

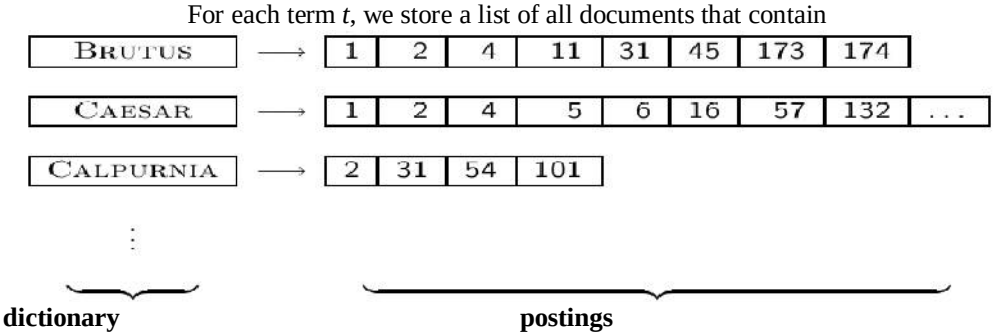

Fig2. Construct the Inverted Index

ISSN(Online): 2319-8753
ISSN (Print): 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Website: www.ijirset.com

Vol. 6, Issue 5, May 2017

2.1- Difference coding

This coding mechanism (sometimes called deltas, gaps)[6] encodes a set of integers by deducting successive values. This sort of coding has been used in the field of data compression for years. The main idea is to code the first value as it is and the remaining values will be coded as the difference between successive values. It achieves a good compression ratio if the differences between successive values are small.

Delta coding is one of the popular techniques for integer coding. It codes integer values by subtracting successive values using simple arithmetic, $\delta = Xi - (Xi - 1)$. The integers must be sorted. If the difference between two consecutive integers is small then we can code the difference with few bits, but one large difference can adversely affect the total compression ratio. Consider a sequence of integers like 24, 25, 32, 43, 55, 77. According to difference, we will store the first value as it is 24. We will store the differences of successive values rather than storing the original values. As a result, the coded sequence will be 24, 1, 7, 11, 12, 22. We can revert back to the original sequence by computing the prefix sum.

Table: Encoding gaps instead of document IDs. For example, we store gaps 107, 5, 43, ..., instead of docIDs 283154, 283159, 283202, ... for computer. The first docID is left unchanged (only shown for arachnocentric).

		encoding	postings list									
	the	docIDs	...		283042		283043		283044		283045	...
		gaps				1		1		1		...
	computer	docIDs	...		283047		283154		283159		283202	...
		gaps				107		5		43		...
	arachnocentric	docIDs	252000		500100							
		gaps	252000	248100								

Fig3. Difference Coding (Deltas)

2.2 Variable-Byte coding

An integer can be compressed as a sequence of bytes using Variable-Byte coding. Different authors referred it variously such as v-byte, variable byte [8], VB [13]. This is a byte oriented coding technique. For every byte, the first 7 bits are used to store part of the binary representation of the integer, and the last bit is used for a status bit which indicates whether the next byte is part of this integer [14]. For example, an integer k = 312 and its binary representation is

100111000. Using variable byte coding k can be represented by two bytes: 10000010 00111000. The status bit 0 in the first byte (00111000) indicates that the next byte is also part of the integer **See Fig 4.**

In total, it took 16 bits to encode the number 312. While decoding, we read byte by byte. We discard the eighth bit if it is 0 and continue reading the next bytes till we get 1 in the coding is easy to implement and known to be significantly faster than traditional bit oriented methods [4].

The main drawback of this method is that it will take at least 1 byte to store even a small integer. In the case of encoding an integer of a single bit such as 1, we have to store it using one byte and the byte looks like 10000001. As a result, we waste 7 bits. Thus, it can make Variable-Byte coding inefficient in terms of compression ratio.

docIDs	200	205	214782
gaps	200	5	214577
VB code	00010010 10000001	10100001	10001100 00110000 10110001

Fig4. Variable-Byte Scheme

IV. PROPOSED SCHEMES (VARIBALE-5BITS SCHEME)

We called **V5bits** it enhanced to VByte scheme. An integer can be compressed as a sequence of 5 bits using Variable-5bits coding. This is a 5bits oriented coding technique. For every 5bits, the first 4 bits are used to store part of the

ISSN(Online): 2319-8753
ISSN (Print): 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Website: www.ijirset.com

Vol. 6, Issue 5, May 2017

binary representation of the integer, and the last bit is used for a status bit which indicates whether the next byte is part of this integer. For example, an integer k = 200 and its binary representation is 11001000. Using V5bits coding k can be represented by 10 bits: 1110001000. The status bit 0 in the first 5 bits (01000) indicates that the next 5 bits is also part of the integer.
In total, it took 10 bits to encode the number 200. While decoding, we read 5 bits by 5 bits. We discard the 5 bits if it is 0 and continue reading the next 5 bits till we get 1 in the 5 bit for any 5 bits to record any integer **See Fig 5**. Variable-5bits coding is easy to implement.

docIDs	200	205	214782
gaps	200	5	214577
VB code	0001**0** 0011**1**	1010**1**	1000**0** 1100**0** 0110**0** 0010**0** 1100**1**

Fig.5 V5bits

To quantify the trade-offs among between two algorithms, we have to implement apply them to randomly generated data sets. We use some code for compression and decompression of integers written in Java which is available as open source under the Apache License 2.0 (https://github.com/lemire/JavaFastPFOR). Along with the implementation of the core compression algorithm, we need to develop we own testing strategy to test and verify. All the implementations have been written in Java using SDK version 1.8.0
In our experiment we focus on compression of integer values. The main objective of this study is to examine and compare the performance of enhanced V5bits scheme over VByte method.
Mainly, we comparing between them based on compression ratio and decompression speed. We conducted our experiment based on synthetic data sets. We can reproduce synthetic data at any time and perform testing on those data. We run different trials by varying different parameters such as number of integers, integer size, etc.
In order to run all of our experiments on synthetic datasets we use a machine equipped with Intel(R) Pentium(R) CPU B960 @ 2.20 GHz 2.20 GHz processor and 2 GB of RAM.

V. EXPERIMENTS AND RESULTS

To evaluate the performance of two compression schemes we performed thorough experiments on synthetic data. We varied different parameters such as integer size, number of integers, etc. We analyzed the effect of different aspects of datasets on compression. Again, size of integers may influence the performance of compression. We will take more bits to compress one large integer. We can save space by compressing small integers into smaller memory space compared to larger integers. Moreover, the difference between the consecutive integers also influences the compression ratio when we use delta coding or differential coding.
The cardinality of the data may influence the compression rate.
We have identified different factors that can influence the compression size. Our final goal is to examine and compare the performance of two compression schemes on integer's numbers.
In our experiment we examined not only compressed size but also compression and decompression speed. We examined how different factors can influence the time to compress and decompress data.
To evaluate the performance we ran several identical trials varying characteristics such as the number of integers and the maximum length of each integer.

1. Generation of data

To perform the experiment, we generated data with two different parameters.

- fixed number of integers
- Variable number of integers

After generate integers by two different parameters we applied concept of Delta coding or differential coding between integers on two parameters.

ISSN(Online): 2319-8753
ISSN (Print): 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Website: www.ijirset.com

Vol. 6, Issue 5, May 2017

Our first approach was to generate generating consecutive integers from zero to a certain limit. We gave two parameters N and Max. Our program generated consecutive integers N distinct integers from 0 to Max and stored it to an array of integers in sorted order. We used differential coding (See **2.2.1**) on the input values and stored the delta values in an array of integers. This set of delta values was used as an input to evaluate different between two compression schemes.
In our implementation we took a delta coded integer array as an input and ran two compression schemes and recorded the compression ratio, compression speed and decompressing speed for every scheme.
In all of our experiments, we measured the timing based on CPU time. In the case of compression and decompression, we ran every trial for 10 times and took the average of the recorded time for every trial. While measuring the timing we consider all the encoding and decoding operations including delta coding and prefix sum.

1.1- Varying integer size

We generated our synthetic data sets with random integers in the range [0 – $2^{\wedge 31}$) for both V5bits and VByte schemes. We generated integers from zero to the integer with the specified number of bits, i.e., the range will be from 0 to a maximum 31 bit integer which means the interval is from 0 to 2147483648. Our test includes the integer range [0 – $2^{\wedge 16}$] [0 – $2^{\wedge 17}$], [0 – $2^{\wedge 18}$) and so on. We generated 32768 numbers of distinct integers for every interval in sorted order and stored them into an array for further processing. Finally, we stored the delta values. We will get larger delta values with increase of the range, since us generating fixed number (32768) of distinct integers in a particular interval. We analyze the effect of integer size on compression. If we have all very small integers we can fit those into small memory space. We have listed the test result for both V5bits and VByte schemes.
We have plotted the test results in **Fig 6**; Fig **6** shows the result of V5bits and VByte scheme. We found the V5bits is more compressible compared to VByte in small integers and equal in large integers. In this test we changed the size of integer to some certain limit. When we generate integers within small range such as [0 – $2^{\wedge 16}$], it is obvious that we will get small delta values compared to larger range like [0 – $2^{\wedge 31}$]. Therefore, when we have lot of small integers, we can compress them into smaller memory space. For both V5bits and VByte data in Fig **6**, we found small integers can be compressed into fewer bits compared to large numbers. In the case of maximum 16 bit integers, we can compress every integer within 5 bits or 10 bits there is an exception for Variable-Byte, since it compresses in a byte oriented manner. Therefore, Variable-Byte needs at least 8 bit to compress any integer.

Fig6. Compression Ratio

Size of integers has some effect on compression speed as well. Compression speed decreases with increase of size of integer. In **Fig 7**, we found compression speed is higher for small integers. Compression speed decreases with increment of integers size. We measure the compression speed in million of integers per second (mls). The V5bits scheme has the lowest compressing speed compared to VByte, because the V5bits after convert the integer into binary reading 4 bits per reading compared with VByte reading 8 bits. To evaluate the applicability of using any compression algorithm, we are mostly concerned about the compressed size and decompression speed of the scheme.

ISSN(Online): 2319-8753
ISSN (Print): 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Website: www.ijirset.com

Vol. 6, Issue 5, May 2017

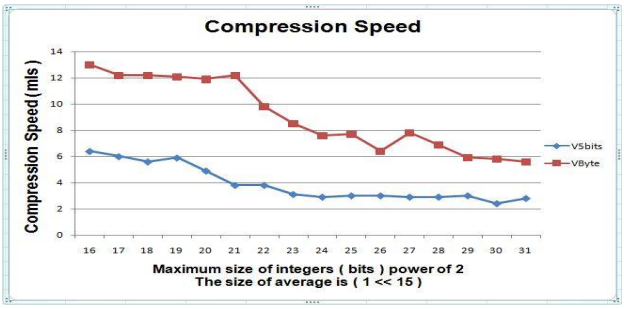

Fig7. Compression Speed

Decompression speed was also influenced by integer size see **Fig 8**. We used the same number of integers in every trial. We found the integer size influenced the performance of compression in every trial. In the case of the first trial, we have maximum number of bits is 16. Therefore, our range of random generated integer is in between [0 – $2^{\wedge 16}$]. The difference between consecutive integers is small for a small range of integers.
In the case of a large range such as [0 – $2^{\wedge 31}$], the difference between consecutive integers is large and we found it takes more bits to compress. We can compress the integers into a smaller number of bits when we have small integers and also compression and decompression time will be significantly high for small integers compared to large integers.

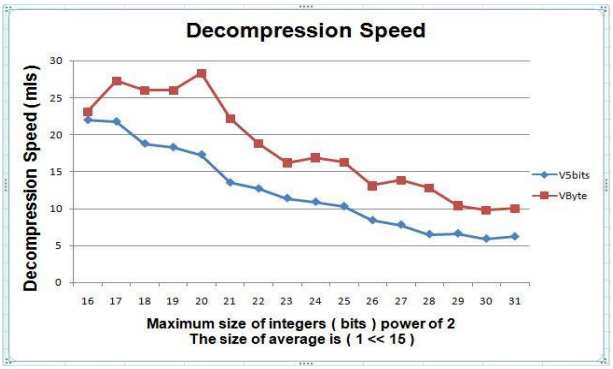

Fig8. Decompression Speed

1.2- Varying the number of integers

Our next approach to testing was to vary the number of integers within the same (total) range of integers (0 – $2^{\wedge 31}$) and evaluate the performance. In this case, we generated data sets of random integers for both V5bits and VByte. Rather than generating a fixed number of integers like 32768 we generated a variable number of integers in each run. In every trial, we generated 10 integer arrays. Each of them containing same number of integers in a certain pass and in every pass we changed the size of the array by changing the number of input integers. In our experiments we generated random integers within the same range of (0 – $2^{\wedge 31}$) but varying the number of integers i.e., we generated $2^{\wedge 11}$ integers for each input array in the first pass. We continued our trial by generating more integers than in the previous trial by a power of 2. We generated $2^{\wedge 12}$ to $2^{\wedge 22}$ integers in our test. Since the range is fixed, the average difference among random generated numbers will be higher when we generate a small number of integers and reverse in the case of generate a higher number of integers.
We ran the first 6 integers identical trials and generated $2^{\wedge 11}$, $2^{\wedge 12}$, … $2^{\wedge 16}$ integers. In every trial, we generated random numbers in the range (0 – $2^{\wedge 31}$). The difference among consecutive numbers is large, since we generating fewer of integers on a fixed large range (0 – $2^{\wedge 31}$). The differences between consecutive integers are known as delta values.
In **Fig 9** we found that to compress integers with large delta values takes more bits. Moreover, it takes more time to decompress.

ISSN(Online): 2319-8753
ISSN (Print): 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Website: www.ijirset.com

Vol. 6, Issue 5, May 2017

In the case of ran the last 6 integers, we generated $2^{\wedge 17}$, $2^{\wedge 18}$, … $2^{\wedge 22}$ integers in the same range ($0 – 2^{\wedge 31}$). Delta values will be lower in high integers. As a result, we can achieve very good compression ratio and also good decompression speed.
Compression size is increasing whenever we generating more integers on a fixed range ($0 – 2^{\wedge 31}$). We will get small delta values when we have more integers. As a result we can compress integers with fewer bits.

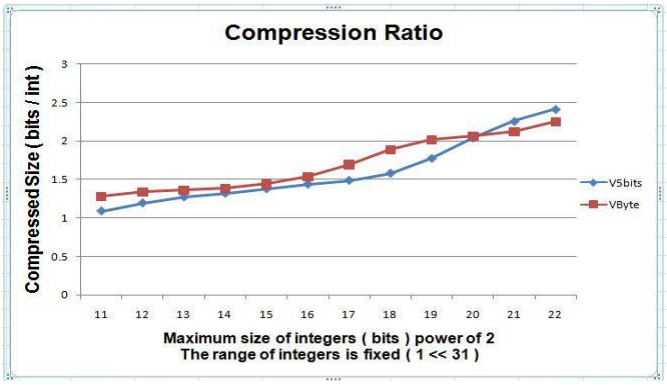

Fig9. Compression Ratio

Fig 10 shows the compression speed. We Compression speed increases with the increment of number of integers, since delta value decreases as the number of integers increases. We can see the effect of the number of integers on compression speed in **Fig 10**. VByte compression speed faster than V5bits.

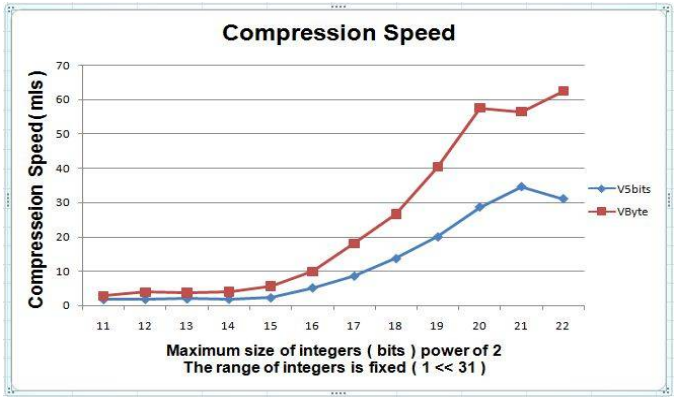

Fig10. Compression Speed

Decompression speed was also influenced by integer size. We can see in **Fig 11**, decompression speed increases with the increase in number of integers.
VByte faster in terms of decompression speed than V5bits.

ISSN(Online): 2319-8753
ISSN (Print): 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Website: www.ijirset.com

Vol. 6, Issue 5, May 2017

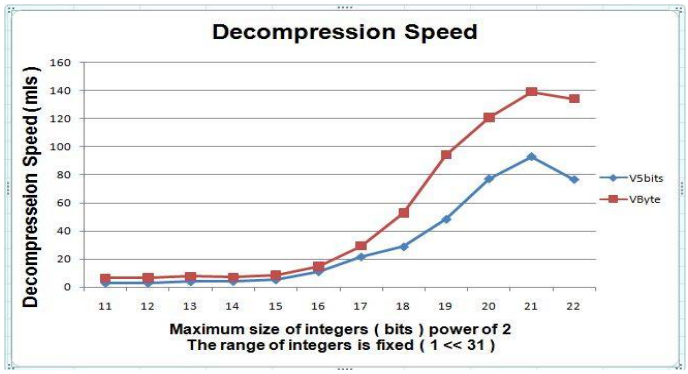

Fig11. Decompression Speed

In our synthetic data test, we generated a fixed number of integers and also a variable number of integers, but in both of my cases we have seen the size of integers influences the compression ratio and speed of compression and decompression.
However, V5bits is clearly the winner for compression size when generated small integers in case of fixed number of integers and when generated more integers in case of variable number of integers.
VByte is clearly the winner for speed of compression and decompression but it does not compress

VI. CONTRIBUTIONS

This research is based on improving compression ratio for Variable-Byte scheme.
Variable-Byte coding is easy to implement and known to be significantly faster than traditional bit oriented methods.
The main drawback of this method is that it will take at least 1 byte to store even a small integer [14].We compare our enhanced approach against original approach for inverted indexes on synthetic data. We introduce enhanced scheme for Variable-Byte scheme, the Variable-5bits scheme.
We show that V5bits provides the best compression ratio for small integers compare with Variable-Byte scheme and trade-off between compression speed, decompression speed and compression ratio.

VII. FUTURE WORKS

We have tested the performance of V5bits compression schemes in the case of synthetic data. In future we can evaluate the outcome of these schemes by real dataset to assess real performance. Observe in the results of experiments that did not get a high compression ratio for large numbers, we will work to improving compression ratio for large numbers.

REFERENCES

[1] Kobayashi, M. and Takeda, K.:Information retrieval on the web, ACM Computing Surveys, Vol.2, No.2, pp.144-173(2000)
[2] Dean, J.: Challenges in building large-scale information retrieval systems: invited talk. In: Proc. WSDM '09. (2009)
[3] Anh, V.N. and Moffat, A.: Inverted index compression using word-aligned binary codes, Information Retrieval, Vol.8, No.1, pp.151-166 (2005)
[4] Tortman, A.: Compressing inverted files, Information Retrieval, Vol.6, No.1, pp.5-19 (2003)
[5] B¨uttcher S, Clarke CLA. Index compression is good, especially for random access. Proceedings of the 16th ACM conference on Information and Knowledge Management, CIKM '07, ACM: New York, NY, USA, 2007; 761–770, doi:10.1145/1321440.1321546.
[6] Witten, I.H., Bell, T.C., Moffat, A.: Managing Gigabytes: Compressing and Indexing Documents and Images. 1st edn. (1994)
[7] Catena, Matteo, Macdonald, Craig, and Ounis, Iadh, On Inverted Index Compression for Search Engine Efficiency, Springer.com, 2014
[8] Williams, H.E., Zobel, J.: Compressing integers for fast file access. The Computer Journal 42 (1999)
[9] Scholer, F., Williams, H.E., Yiannis, J., Zobel, J.: Compression of inverted indexes for fast query evaluation. In: Proc. SIGIR '02. (2002)
[10] Compressing inverted index using optimal FastPFOR, Veluchamy and Sandanam, journal of information processing Vol.23 No.2 185-191 (Mar. 2015).
[11] Zobel, J., Moffat, A. and Ramamohanarao, K.: Inverted files versus signature files for text indexing, ACM Trans. Database Systems, Vol.23, No.4, pp.453-490 (1998)
[12] Manning, C.D., P. Raghavan, and H. Schütze, *Introduction to information retrieval*. Vol.1. 2008: Cambridge University press Cambridge.
[13] A. Stepanov, A. Gangolli, D. Rose, R. Ernst, and P. Oberoi. Simd-based decoding of posting lists. In Proceedings of the 20th ACM International Conference on Information and Knowledge Management, CIKM '11, pages 317–326, New York, NY, USA, 2011. ACM.
[14] J. Zhang, X. Long, and T. Suel. Performance of compressed inverted list caching in search engines. In WWW '08, pages 387–396, 2008.