Int. J. Advance Soft Compu. Appl, Vol. 7, No. 3, November 2015
ISSN 2074-8523

A Lossless Compression Technique to Increase Robustness in Big Data Transmission System

Shiladitya Bhattacharjee[1], **Lukman Bin Ab. Rahim**[1], **and Izzatdin Bin A Aziz**[1]

[1]Department of Computer & Information Science,
Universiti Teknologi PETRONAS, Bandar Seri Iskandar, Malaysia
e-mail: shiladityaju@gmail.com, lukmanrahim@petronas.com.my, izzatdin@petronas.com.my

Abstract

Data loss during big data transmission is an important issue. Data loss occurs due to various transmission errors, channel noises and huge data size, which increases channel congestion and transmission delay. However, these issues can be controlled by using data compression. Generally, these compression techniques are lossy and lossless in nature. The lossy technique degrades data quality during compression while in lossless compression, the output remains the same as the input. Among the two types of lossless compression, variable length coding (VLC) is efficient to compress data while fixed length coding (FLC) produces poor efficiencies in big data compression. Furthermore, VLCs offer poor robustness against various errors and channel noises during transmission. Hence, to solve the problem of compression efficiency, robustness and time delay in big data transmission, we have designed a FLC based compression. Efficiencies of the proposed technique in various aspects are measured using standard files of various types and sizes as inputs. The result shows that it offers better compression efficiencies by producing lower compression ratio and better compression percentage than other existing techniques. Its capacity to produce better signal to noise ratio (SNR) and throughput justify its capacity to increase robustness and reduce time delay.

Keywords: *Lossless Compression, Compression Ratio, Fixed Length Coding, SNR, Throughput.*

1 Introduction

Big data is the collection of data sets so large and complex that it is always difficult to process and transmit the data over the network. The large size itself is a big

challenge for big data. Due to its huge size, channel congestion and transmission delay increases and data loss may occur. Various compression techniques are invented to control these data sizes. However, some compression techniques are inefficient to produce desired compression efficiencies and some are inefficient to offer robustness against various errors, channel noise and data loss.

With the rapid use of Internet, a transmission of big size electronic media files e.g. text, image, audio and video requires compression to avoid any transmission overhead and reduces channel congestion. As we know, compression can be divided into two categories: lossy and lossless compressions. In the lossy compression, some of the statistically redundant data sets are permanently eliminated to compress the data and the eliminated data cannot be retrieved during decompression. However, these data losses do not impact on the originality and the user may not identify the difference. As the lossy compression techniques [1] remove some data permanently, it offers higher compression efficiency. Lossy compression [1, 2] is generally used for video, audio and image compression. The examples of these lossy compressions are MP3, Mpeg and Jpeg. But in the lossless compression, the numbers of data bit remain the same as original after decompression. In lossless compression techniques, data are rewritten to in a more efficient way [3]. As there is no information losses occurred in it, the lossless compressed file size will be larger than the lossy compressed file size. Rarely, some changes to data are required but if the compression technique is strong, these changes will not be noticeable during decompression [4]. It can be used in video, audio, image and text compression. Zip, PNG and GIF are some examples of lossless compressions [3, 4].

Among the two categories of lossless coding, variable length coding techniques (VLCs) produce their compressed codes by depending upon prefix codes where the prefix code makes the difference between two consecutive codes. Apart from these, the prefix codes can be easily influenced by the various types of channel noises or data errors. Sometimes, if a single bit of the prefix code gets corrupted during the transmission, the whole compressed code will be unreadable. Thus, the corruption of prefix code may cause partial or complete data loss [8, 9]. Therefore, the existing VLC compressions are not robust against the various data errors and channel noises. The example of such VLCs are Huffman coding, Lampel Ziv coding, dictionary based coding and others [9]. On the other hand, another category of lossless coding is fixed length coding techniques (FLCs). FLCs can solve the robustness issue as they do not depend upon prefix code during compressed code generation. However, these existing FLCs are inefficient to produce required compression efficiencies in big data application [6, 7]. The available FLCs are run length coding, ASCII coding, EBCDIC and so on [7]. Therefore, we have designed a FLC based compression technique for big data which fulfills the following objectives:

- Higher efficiency than the existing technique for big input size data.
- Better robustness by reducing the data error during transmission.

- Better processing speed to reduce the transmission delay and channel efficiencies.

According to the objectives, our proposed FLC based compression can be applied in many real life big data applications when data loss is an important concern. The proposed compression technique can be applied in areas where data loss and corruption during transportation will affect the applications. Examples of these areas are scientific research, healthcare, industrial data storage, and governmental and geographical surveillance.

In this paper, Section 2 discusses the background study, Section 3 depicts the proposed technique more precisely and Section 4 contains the results of implementation to justify our claim. Section 5 represents an assessment platform which includes experimental setup and some important definitions, which will be used in result analysis section. A conclusion is drawn in Section 6 to conclude the effectiveness and the drawback of the proposed technique.

2 Background Study

Here is the list of nomenclature which will be used throughout the text:

Nomenclature:

Each of the following symbol represents,

$\sum$	Finite set of all numbers and characters
$\emptyset$	Null Sequence
$\lfloor\ \rfloor$	Floor function
$\lceil\ \rceil$	Celling function
modulo	Reminder Division
$\langle\ \rangle$	Element sequence of a set

During big data transmission, its size is a considerable issue. So, to control such data size various compression techniques are used. These compression techniques are either lossy or lossless in nature. As lossy compression techniques degrade data quality during retrieval of the original data from the compressed string. In [1-2], the proposed lossy compression techniques are designed for real time data compression. Here the difference between the original and decompressed data is very insignificant. They offer high compression efficiency and high processing speed too. However, it does not offer higher robustness and even small changes in the compressed code can damage the entire data. Therefore, we have not included lossy compression in our discussion.

Among the two types of lossless compression, examples of FLCs are ASCII coding and run length coding [6] and examples of VLCs are Arithmetic Coding, Huffman Coding, Lempel-Ziv Coding and Dictionary based coding. Fixed length coding

converts the symbols of a source file into a new weighted binary code depending upon a few certain bits. Probability of each symbol to be appeared in the original file is estimated and then put in decreasing order during the generation of compressed code [5, 6].

These types of coding techniques are generally asymptotic in nature and less sensitive to channel noise i.e. if any bit gets corrupted during transmission, it does not affect the whole data block. In (FLC) technique, no special symbol is used to separate the characters but the opposite occurs in VLCs. The main disadvantage of fixed length coding is that it produces equal length of code word for both max and min frequent symbol. Seldom, the code lengths are the same or even larger than the original character or symbol lengths [5-7].

Variable length codes permits inputs for compressing and decompressing without any error. Independent and similar distributed inputs are compressed by these techniques which are near to their entropy [8, 9,20-21]. Generally prefix code can be defined in such a way that no code word is a prefix of any other code word [10, 11]. Coding scheme, structured in this scheme is as similar as the prefix property and unambiguously decodable at any time. In this representation, the size of the set of integers (n) should be known in advance. Code size (P') can be determine as

$$P' = 1 + \lfloor log_2 n \rfloor \qquad (1)$$

Entropy coding can be used for different media, i.e. video, image, audio or text, as it has the property of not to be media specific. Furthermore, the retrieved data are of the same size as the original and there are no losses of data occurred during the retrieval, so they are called the lossless coding techniques [24]. The process of entropy coding can be differentiated into two parts: modeling and coding. The Huffman coding and the arithmetic coding techniques are the most relevant examples of such entropy coding techniques [20-21]. Other different types of VLC techniques that are widely used are Lempel Ziv, Dictionary based coding [19] and Deflate [23].

During the transmission process, compressed code can be erroneous by the various types of data errors and noises, whether they are hardware or software. Sometimes noises in the channel or environment plays a role to corrupt the transmitted data [21, 22]. Data errors can be of various types: single bits, multiple bits, discrete or continuous errors. Generally, data link layer is responsible for controlling such transmission errors. However, most of the time, all data errors cannot be corrected. Therefore, when we transmitted a compressed string over the internet, these compressed codes should be robust against such data errors and noises [23].

According to [12], in arithmetic compression techniques, message is encoded as a real number in an interval between zero to one. As it works on a single character rather than several code words, it produces better compression ratio than others. The main disadvantage of such coding is if one bit is corrupted during the compression or during transmission of compressed code over the network the whole

code will be irretrievable [12-13]. In Huffman coding, compressed codes are generated by traversing the Huffman binary tree. Generally, Huffman binary tree is designed by depending upon the frequencies of different symbols belonging to the input file [14-15]. As the prefix code is generated depending upon the frequencies of the different symbols and traversing the binary tree, any minor changes in the compressed code due to any error or noise will make the whole compressed code irretrievable. Lempel-Ziv compression technique incorporates a principle or protocol for parsing of symbols from a finite alphabet into substrings where coding scheme maps these substrings accordingly into genuine code words of fixed length [16]. This type of coding technique is categorized into a few categories i.e. LZ77, LZ78 and etc. LZ77 uses all the features of Lempel Ziv coding with the exception that it uses the sliding buffer or window to store the substring [17]. Even in the LZ77 and LZ78 compression technique, a sliding window is used to store the character table during the creation of compressed code. Similarly, this sliding window is used during decompression too. So, if any compressed code gets corrupted during transmission, data loss may occur. Therefore, though the variable length coding techniques are channel efficient as they produces better compression efficiencies, they are not robust against the continuous or discrete data errors, channel noise and they are inefficient to protect data losses.

In [4-5], the proposed lossless based data compression for floating seismic depends upon arithmetic coding that can achieve higher compression ratio and low data loss rate as compared to other lossless algorithms. It even offers higher SNR which shows its potentiality to offer robustness against various noise and errors. However, it processes the data quite slowly and its implementation is very expensive. In [7-9], the techniques being used generate compressed codes by trimming the redundant texture and using space transformation method. These types of compression techniques are very useful when the dynamic expansion is not possible. These techniques are very slow and the usage of modular arithmetic as well as data transformation decreases correlation in every case.

From the discussion above, we have seen that both fixed length and variable length coding based compressions have some disadvantages where the FLCs offer poor compression efficiency and VLCs offer poor robustness against various data errors and noises. Therefore, existing compression techniques are not suitable for compressing large input size file for transmission. Thus, we proposed in this paper a new FLC compression algorithm that is has higher efficiency and robust against data errors and channel noise for transmitting large files.

3 Proposed Methodology

To overcome the limitations of both existing fixed and variable length coding, we have proposed a fixed length coding based compression technique for big data transmission. It is useful for squeezing big data file and not fully dependent on

prefix code thus offering high robustness against channel noises and errors. To elaborate more precisely we have drawn our proposed model in Fig.2.

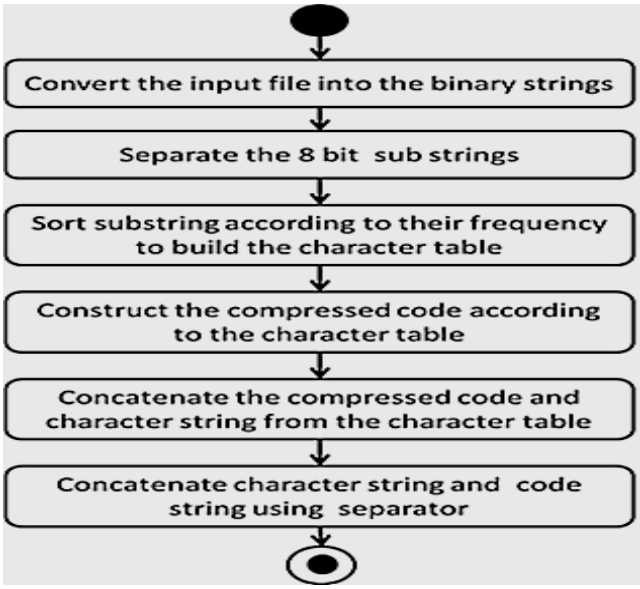

Fig. 2: Proposed Fixed Length Compression Model

From Fig.2, we can see that, the proposed compression technique can be divided into two major parts: creation of character table and creation of compressed code. The different operations related to our proposed compression technique are described in the following subsection.

3.1 Operations related to proposed compression technique

In the first half of the proposed compression model, the character table is generated. Therefore, to create a character table, take the input file (IF) and convert it into a binary string as,

1) Check the IF size is divisible by 8 or not by

$$K = ((IF\ Size)\ modulo\ 8) \qquad (2)$$

Here K is an ordinary variable. If $K < 8$ then concatenate $(8 - K)$ number of bits at the Most Significant Bit (MSB) position of the IF string.

2) Separate $\left(\frac{IFSize}{8}\right)$ numbers of substrings from the input file (IF) string and store them into the sub strings $\langle Sub_n \rangle$ as $0 \leq n \leq \left(\frac{IFSize}{8}\right)$.

3) Construct the symbol table $SymbTab$ by eliminating redundant data from $\langle Sub_n \rangle$ by

$$SymbTab = \binom{Sub_n}{m} \quad (3)$$

Here m is the number of irredundant substring in $\binom{Sub_n}{m}$, where $0 \leq m \leq 256$.

4) Accumulate the frequencies of each $\binom{Sub_n}{m}$ in $\langle Freq_m \rangle$ from the $\langle Sub_n \rangle$.
5) Sort all the string of $\langle SymbTab_m \rangle$ according the frequencies at $\langle Freq_m \rangle$ in decreasing order using any sorting algorithm.

In the second part, we create the code table $\langle Code_m \rangle$ and separator Sep using Algorithm-1.

Algorithm-1: Compressed Code Generation

(b1) a, b, c, d, e are ordinary variables where $a = 0$;
(b2) $str,\ str1, str2$ are string variables;
(b3) if $(m \geq 0)$ and $(m \leq 2^7)$
(b4) $a = a + 5$; // if $m \leq 40$ put $a = 4$
(b5) if $(m \geq 0)$ and $(m \leq (2^a - a))$
(b6) for $b = 0$ to a
(b7) $c = m/2^b$;
(b8) $d = c\ modulo\ 2^b$;
(b9) Concatenate (str, d);
(b10) end for
(b11) $Code_m = str$;
(b12) end if
(b13) if $(m > (2^a - a))$ and $\left(m \leq \left(2^{(a+1)} - (2 \times a)\right)\right)$
(b14) $e = (2^a - 4)$;
(b15) for $b = 0$ to a
(b16) $c = e/2^b$;
(b17) $d = c\ modulo\ 2^b$;
(b18) Concatenate $(str1, d)$;
(b19) end for
(b20) Repeat step (b6) to (b10);
(b21) Concatenate $(str2, str1, str)$;
(b22) $Code_m = str2$;
(b23) end if

Algorithm-1: Compressed Code Generation (Contd…)

(b24) if $(m > (2^{(a+1)} - (2 \times a))$ and $m \leq ((3 \times 2^a) - (3 \times a))$
(b25) $e = (2^a - 3)$;

(b26) Repeat step (b15) to (b22);
(b27) end if
(b28) if $\left(m > \left((3 \times 2^a) - (3 \times a)\right)\right)$ and $m \leq (2^{(a+2)} - (4 \times a))$
(b29) $e = (2^a - 2)$;
(b30) Repeat step (b15) to (b22);
(b31) end if
(b32) if $\left(m > \left(2^{(a+2)} - (4 \times a)\right)\right)$ and $\left(m \leq 2^{(a+2)}\right)$
(b33) $e = (2^a - 1)$;
(b34) Repeat step (b15) to (b22);
(b35) end if
(b36) $e = 2^a$;
(b37) for ($b = 0$ to a)
(b38) Repeat step (b16) to (b17);
(b39) Concatenate (d, Sep);
(b40) end for
(b41) end if
(b42) if $(m \geq 0)$ and $(m \leq 2^8)$
(b43) $a = a + 6$;
(b44) Repeat step (b5) to (b40);
(b45) end if

Pseudo code lines (b1) and (b2) declare and initialize some ordinary and string variables. Lines (b3) to (b41) is for creating code when the value of m varies $0 \leq m \leq 127$. Lines (b5) to (b12) creates a code substring when $0 \leq m \leq 26$. Lines (b6) to (b9) convert the value of m into a binary string to create a compressed code for each symbol. Line (11) put these binary strings into the code table. Similarly, lines (b14) to (b19) are for creating the prefix code of the second level. Lines (b20) to (b23) are for creating a code for the second range and this code is concatenated with the prefix code. Line (b22) adds this concatenated code into the code table. In, fourth and fifth level, the prefix and the compressed codes are generated using similar fashion. These compressed codes are then included into the code table with the help of lines (b13) to (b41). Similarly, lines (b42) to (b45) generate the code and include them into the code table when $128 \leq m \leq 256$.

6) Replace all the substrings from the $\langle Sub_n \rangle$ by the corresponding code from $\langle Code_m \rangle$ to form a code string table $\langle CodeStrTab_n \rangle$ to create compressed string $(CompStr)$.

7) Concatenate all code string from the symbol table $\langle SymbTab_n \rangle$ into a single string $SymbStr$ as,

$$SymbStr = \left(SymbStr \times 10^{l(SymbTab_m)}\right) + SymbTab_m \qquad (4)$$

Where, $l(SymbTab_m) = \left(\left\lfloor log_{10}^{SymbTab_m} \right\rfloor + 1\right)$

8) Concatenate all strings of $\langle CodeStrTab_n \rangle$ into a single $SymbCodeStr$ using Eqn.4.
9) Create the final compressed string ($CompStr$) by concatenating code string ($SymbStr$), separator string (Sep) and compressed string ($CompStr$) using Eqn.4. Finally, transmit the compressed string ($CompStr$) to the receiving end.

3.2 Operations related to Decompression

Decompression is done at the receiving end after receiving the complete compressed data. Decompression technique is a reversal process of the proposed compression technique. During decompression, the symbol table ($SymbStr$) is separated based on separator (Sep') from the compressed string ($CompStr$). The 8 bits symbols are generated from the $SymbStr$ by reversing the process and form the symbol table. Then construct the code table from the symbol table by using Algorithm-1. Similarly, separate each code substring from $SymbCodeStr$ and replace each code substring by the corresponding symbol from the symbol table by comparing them from the code table. Finally, concatenate all symbols into a single string using Eqn.4 and generate the output file.

4 Assessment Platform

To analyze the performance of our proposed technique for big data transmission, we need to measure its capacity to produce compression efficiency and to offer robustness against various errors and data noises. Therefore, this section includes definition of required parameters that relate to our experiment and the experimental setup.

4.1 Experimental setup

We have conducted several experiments to measure the efficiency of the proposed algorithm in different aspects. We have used LINUX (Fedora 18) as the operating system. However, any operating system can be used for implementing it. We have used a 32 GB of DDR3 RAM and cloud storage at the High Performance Computing Center of Universiti Teknologi PETRONAS for our experiment. However, the minimum hardware requirement is 2GB of RAM and 1TB of storage space. The proposed compression technique is implemented using the Java programming language to conduct all experiments in NetBeans Integrated Development Environment. The input file sizes vary for conducting our experiments from 35 GB to 1TB (considered as big data). Files of different types and sizes have been taken as input to measure the efficiency of our proposed compression technique. Data transmissions are done via wireless network infrastructure during the experiment.

4.2 Some Important Definition

This subsection is included to provide a brief idea about some important parameters and describes their different features. These parameters help to justify our claim by analyzing the results of different experiments at the result analysis section.

4.2.1 Compression Ratio (CR)

In any data compression technique, the Compression Ratio represents the ratio of compressed file size and original file size. It is an important parameter to measure the efficiency of any compression technique.

$$\mathrm{CR} = \frac{Copressed\ File\ Size}{Original\ File\ Size} \tag{5}$$

Generally, the compression ratio is inversely proportional to the percentage of compression, i.e. if any compression technique produces low compression ratio then it will produce high percentage of compression or vice versa.

4.2.2 Percentage of Compression (CP)

Percentage of compression (CP) signifies the ability to compress the input data (in %) after certain compression process is applied on that input file. The percentage of compression (CP) can be formulated as,

$$CP = \frac{Actual\ Size - Compressed\ Size}{Actual\ \ size} \times 100 \tag{6}$$

The percentage of compression signifies the capacity of reducing a file size of any compression technique. If any compression technique produces higher CP, it signifies that this particular compression technique is efficient to compress the input file or vice versa.

4.2.3 Signal to Noise Ratio (SNR_{dB})

Signal-to-noise ratio is the ratio of the amplitude of a desired analog or digital data signal to the amplitude of noise in a transmission channel at a specific point in time. SNR is typically expressed logarithmically in decibels (dB). SNR can be formulated using the following formula:

$$SNR_{dB} = 10 \times Log_{10}^{\{\sum_n x^2(n)/\sum_n [x^2(n)-y^2(n)]\}} \tag{7}$$

In Eqn.7, $x(n)$ is the original sample length whereas $y(n)$ is the sample length of the output file. Generally, if SNR_{dB} of a file is high then it is said to be less noisy or less erroneous or vice versa.

4.2.4 Percentage of Information Loss (IL)

During data transmission, some portion of the transmitted data can be modified or corrupted by channel noises or some unwanted interference. This information cannot be retrieved at the receiving end. This facet is known as information loss (*IL*) and it is measured in percentage. The percentage of information loss can be formulated as:

$$IL = \frac{Original\ Size\ - Retrieve\ Size}{original\ \ Size} \times 100 \qquad (8)$$

Generally percentage of *IL*, produced by any compression technique is inversely proportional to its capacity to produce robustness against various errors, i.e. when the *IL* is high then the robustness is low and vice versa.

4.2.5 Throughput (TP)

Throughput or *TP* is the amount of work being done in a given time. It is measured to calculate the processing speed of compression or decompression techniques. Generally, *TP* is inversely proportional to time and can be measured as:

$$TP(Comp.) = \left(\frac{Compressed\ \ Size}{Compression\ Time}\right) \qquad (9)$$

$$TP(Decomp.) = \frac{Retrive\ File\ Size}{Decompression\ Time} \qquad (10)$$

4.2.5 Peak Signal-to-Noise Ratio (PSNR)

The peak signal-to-noise ratio (PSNR) is the ratio between a maximum power of a particular signal and the power of the incorporated noise into that signal. It is used to measure the quality of reconstructed images that have been compressed. Each picture element (pixel) has a color value that can change when an image is compressed and then uncompressed. Signals can have a wide dynamic range, thus PSNR is usually expressed in decibels.

If the original signal strength of any image file is $s[n]$ and the signal strength with the noise is $x[n]$ then the peak signal to noise ratio (PSNR) can be calculated as:

$$PSNR = \frac{P_S}{P_W} = \frac{P_S}{|x[n]-s[n]|^2} \qquad (11)$$

Where,

$$P_s = \sum_{-\infty}^{\infty} s^2[n] = |s[n]|^2 \quad (12)$$

According to this equation, if the PSNR of any image is higher then it is less erroneous and is less affected by noise and vise versa.

5 Result Analysis

Compression efficiency, robustness and processing speed produce by the proposed technique are calculated in terms of their corresponding parameters. We have also considered some standard existing VLCs and FLCs to compare our proposed algorithm in different aspects to prove our claims.

5.1 Compression efficiency measurement

Compression efficiency is calculated by measuring the compression ratio offered by various VLCs, FLCs and the proposed compression technique using Eqn. 5. The results are tabulated in Table 1 and Table 2.

Table 1: *CR* offered by our tech. and other VLCs

File Name	Arithmetic coding	Huffman BWT	LZSS BWT	Dictionary based	Proposed Technique
BIB	0.6819	0.5546	0.5356	0.5085	0.4621
GEO	0.6888	0.5571	0.5321	0.5023	0.4618
OBJ1	0.6887	0.5547	0.5399	0.5061	0.4613
PAPER1	0.6879	0.5551	0.5301	0.5073	0.4604
PAPER2	0.6851	0.5525	0.5354	0.5045	0.4597
PAPER3	0.6834	0.5556	0.5347	0.5022	0.4604
PAPER4	064845	0.5588	0.5342	0.5081	0.4647
PAPER5	0.6819	0.5501	0.5325	0.5076	0.4661
PAPER6	0.6827	0.5585	0.5366	0.5079	0.4612
PROGC	0.6817	0.5527	0.5371	0.5067	0.4609
PROGL	0.6820	05781	0.5345	0.5081	0.4598
PROGP	0.6818	0.5533	0.5369	0.5089	0.4586

Table 2: *CR* offered by our tech. and other FLCs

File Name	ASCII Coding	EBCDIC	Run Length Coding	Proposed Technique
BIB	0.9546	0.9256	0.9985	0.4621
GEO	0.9571	0.9221	0.9623	0.4618
OBJ1	0.9547	0.9299	0.9761	0.4613
PAPER1	0.9551	0.9301	0.9773	0.4604
PAPER2	0.9525	0.9354	0.9845	0.4597
PAPER3	0.9556	0.9247	0.9822	0.4604

PAPER4	0.9588	0.9342	0.9881	0.4647
PAPER5	0.9501	0.9325	0.9776	0.4661
PAPER6	0.9585	0.9266	0.9879	0.4612
PROGC	0.9527	0.9271	0.9767	0.4609
PROGL	0.9578	0.9245	0.9681	0.4598
PROGP	0.9533	0.9269	0.9789	0.4586

From Table 1 and Table 2, we can see that our proposed technique produces lower compression ratio than other existing standard FLCs and VLCs. Here, the compression ratio for various existing compression techniques and the proposed technique are calculated by using different standard Calgary Corpuses as input file.

The efficiencies of the proposed compression technique are further compared with other FLCs and VLCs with respect to their ability of producing percentage of compression (*CP*). The percentages of compression produced by these existing FLCs and VLCs and our proposed technique with the help of Eqn.6 are shown in Table 3 and Table 4.

Table 3: *CP* produced by our tech. and other VLCs

File Size (GB)	Arithmetic coding	Huffman BWT	LZSS BWT	Dictionary based	Proposed Technique
35	21.01	30.2	34.1	41.6	56.7
56	23.34	30.9	35.2	42.0	57.0
90	18.39	32.1	33.0	43.5	57.1
120	24.23	33.7	34.7	43.0	56.9
200	25.32	31.3	34.0	43.7	58.0
350	22.12	30.6	35.1	42.9	58.2
450	23.45	31.5	35.2	44.0	58.8
600	21.67	32.0	34.3	45.0	55.9
800	21.45	29.9	36.7	44.1	56.5
1024	21.21	30.7	35.0	43.9	56.8

Table 4: *CP* produced our tech. and other FLCs

File Size (GB)	ASCII Code	EBCDIC Code	Run Length Coding	Proposed Technique
35	0.0002	0.0001	-1.0055	56.7
56	0.0003	0.0002	-1.0026	57.0
90	0.0001	0.0004	0.0001	57.1
120	0.0001	0.0004	-0.998	56.9
200	0.0002	0.0001	-0.762	58.0
350	0.0003	0.0002	-0.985	58.2
450	0.0001	0.0003	-0.976	58.8
600	0.0003	0.0002	-1.112	55.9
800	0.0002	0.0001	-1.007	56.5

1024	0.0001	0.0002	0.0001	56.8

In Table 3 and Table 4, it can be seen that our proposed compression technique offers higher percentage of compression (*CP*) than the various exiting standard FLCs and VLCs. Here, we have used various text files of different size as input. From Table 1 to Table 4, we can see that our proposed technique offers low compression ratio and higher percentage of compression which reflects that our proposed technique is more compatible to reduce data overhead rather than the other existing compression techniques. Thus, we have obtain our first objective.

Apart from that, we can see from Table-2 and Table-4 that, the existing FLCs produce unacceptable compression efficiencies which is not applicable for big data transmission. Therefore, we have not compared the performance of the proposed compression techniques with the existing FLC techniques.

5.2 Robustness to protect data error and information loss

We have further tested the ability of our proposed technique to produce robustness against various data errors. For this purpose, we have calculated signal to noise ratio (SNR_{dB}) of our proposed technique and various exiting techniques with the help of Eqn.7. From our previous subsection we can see that, the existing FLCs produce very poor compression efficiencies. So, we have not included or compared this FLCs with our proposed technique. Fig.3 shows the SNR_{dB} values generated by the proposed techniques and various existing VLCs using different samples as input, taken from related text files of different sizes.

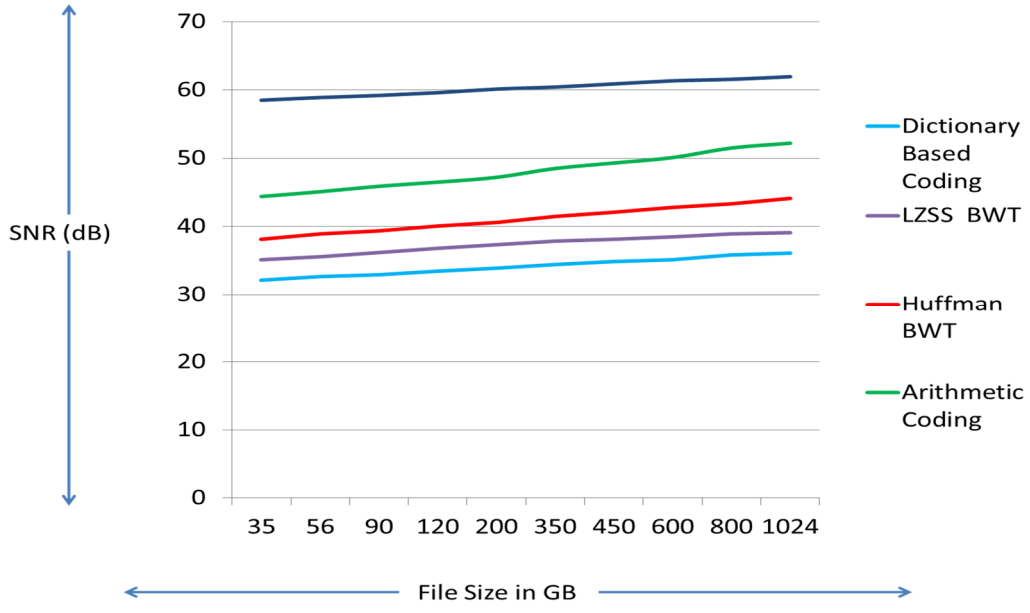

Fig.3: SNR_{dB} produced by proposed tech. and other VLCs

From Fig.3, we can see that our proposed technique can produce SNR_{dB} from 58.5 dB to 62.9 dB which are higher and consistent than other existing techniques. This fact indicates that the probability of noises and errors in retrieving the data samples is very low while our proposed compression technique is used apart from other existing.

We have also used our compression technique to compress image files and compare the PSNR value produced by it with the various existing image compression technique using the Eqn.11. Standard image is being used in this experiment and the results are shown in Table-5.

Table 5: PSNR offered by existing image compressions and proposed technique

Image Name	JPEG-LS	JPEG2000	SPIHT	ADCTC	Proposed Technique
House	32.69	33.10	34.78	36.37	45.2
Milk	32.26	32.53	34.86	35.76	44.8
Jet	31.95	32.89	34.76	35.73	46.3
Tiffany	31.32	32.90	34.80	35.49	49.2
Pepper	32.44	32.57	34.79	35.46	48.9
Baboon	32.65	32.71	34.78	34.73	47.3
Random-noise	31.82	31.85	34.81	34.33	47.8

According to Table-5, we can see that our proposed technique is more efficient and produce better PSNR than the other existing image compression technique for every standard input image. This fact implies that our proposed compression technique is much more efficient to reduce the affect of various data noise during transmission. This fact also reflects that the perceptual difference between the original image and the decompressed image is very low which can be seen by the following Fig.4.

Fig.4(a) Original Image

Fig.4(b) Decompressed Image

Fig. 4(a) and Fig. 4(b) show that the decompressed figure is perceptually the same as the original figure. Therefore, Table-5 and Fig.4 justifies that our proposed technique is efficient to produce higher robustness for the different file formats.

Another parameter for justifying the capacity of our proposed technique to offer robustness against various noises and data errors is percentage of information loss. The percentage of information loss (*IL*) produces by the proposed technique and various existing VLCs are calculated with the help of Eqn.8 using various input text

files of various sizes. The percentage of information losses offer by our proposed fixed length coding based compression technique and various VLC compression techniques are compared are shown in Fig.5.

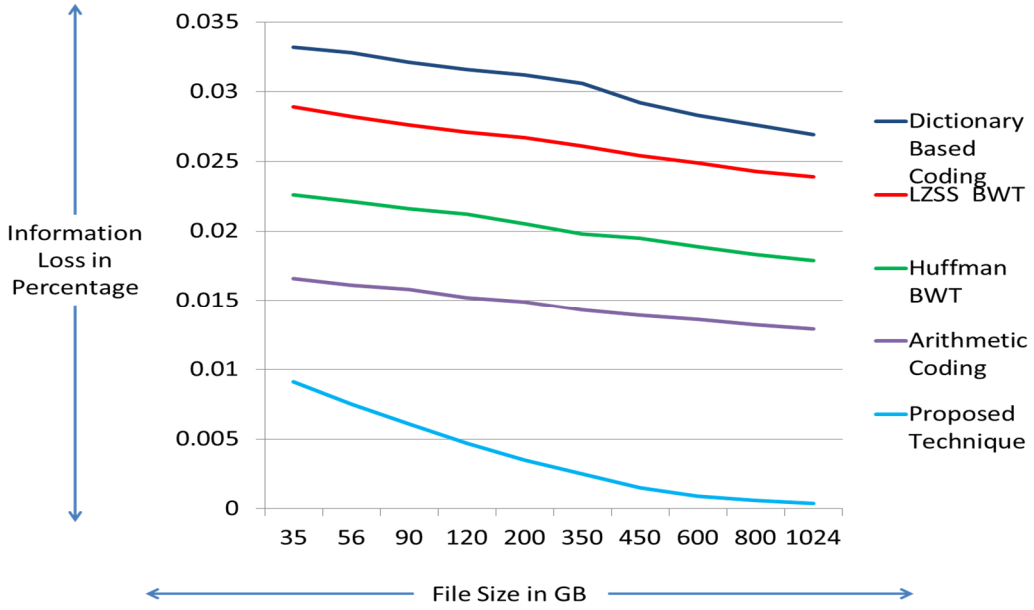

Fig.5: IL produced by propose tech. and other VLCs

From the Fig.5, we can see that, our proposed technique is producing lower percentages of information loss (*IL*) than other existing VLCs. This fact also highlights that the proposed FLC based compression technique offer more robustness against the other existing VLC based compression techniques.

Therefore, from the Fig.3, Fig.4, Figure-5 and Table-5 we can conclude that our proposed compression technique produced a better robustness against various channel noise, environmental noise, and hardware or software errors during the transportation process in wireless environment. These facts justify our second claim.

5.3 Measurement of processing speed

Another important aspect is increasing the channel efficiency in any transmission system. Generally, the channel efficiency can be increased with the increment of data processing speed and with good compression efficiency. From Table-1 to Table-4, we have already seen that our proposed technique is better than other existing in this aspect. Now, we examined how it is efficient to produce the time efficiency in terms of processing speed. From the definition sub section, we can see that the processing speed produced by any compression or decompression technique can be measured by measuring its capacity of offering *Throughput.* The *Throughput* offers by our proposed technique and various existing variable length coding

techniques at the time of their compression and decompression are measured with the help of Eqn.9 and Eqn.10 using various input text files of different sizes. The following Table 6 includes the comparison of *Throughput* offered by the proposed technique and various existing VLCs.

Table 6: *TP* offered by our tech. and other VLCs

Throughput in MB/Sec (Compression (C) / Decompression (D))										
Input File Size (GB)	Arithmetic Coding		Huffman BWT		LZSS BWT		Dictionary Based		Proposed Technique	
	(C)	(D)	(C)	(D)	(C)	(D)	(C)	(D)	(C)	(D)
35	1.56	1.58	0.78	0.79	0.81	0.82	0.93	0.95	3.52	3.51
56	1.61	1.62	0.82	0.85	0.86	0.84	0.99	0.98	3.56	3.53
90	1.64	1.63	0.85	0.87	0.89	0.91	0.95	0.96	3.54	3.55
120	1.65	1.66	0.88	0.89	0.84	0.85	0.96	0.97	3.56	3.54
200	1.71	1.72	0.85	0.86	0.87	0.86	0.98	0.99	3.57	3.55
350	1.77	1.76	0.88	0.87	0.91	0.93	0.95	0.97	3.58	3.57
450	1.76	1.78	0.87	0.86	0.95	0.97	0.97	0.99	3.59	3.57
600	1.75	1.77	0.86	0.88	0.98	0.99	0.95	0.96	3.57	3.56
800	1.73	1.79	0.88	0.89	0.96	0.97	0.97	0.98	3.61	3.59
1024	1.75	1.75	0.91	0.89	0.99	0.98	0.98	0.99	3.62	3.58

From Table 6, we can see that our proposed technique produces better *Throughputs* in every situation for both compression and decompression. These facts establish that our proposed technique is far better to produce higher processing speed than the other existing VLCs during both compression and decompression. Thus, we have proved that our third claim.

6 Conclusion

From the literature review, we have found that channel congestion and transmission errors are the prime grounds of such data loss. On the other hand, the big data size increases the channel congestion during transmission and transmission delay. So we have proposed a fixed length coding based compression technique, which reduces the file sizes considerably as well as reduces data loss by offering high robustness against various data errors, and minimizing channel congestion and transmission delay. Therefore, this paper discusses how our proposed fixed length coding based lossless compression technique reduces information overhead during the transmission, increases the processing speed and reduces the effects of channel noises or errors during transmission. From Table 3 and Table 4 we can see that our proposed technique achieved up to 56.8% of compression efficiency during our experiment. As our proposed technique includes the fixed length coding technique, so channel noise effects very less on data, thus it is reducing the data error. The

SNR_{dB} of the proposed technique increased up to 61.95 dB meaning that the error rates in the retrieve information are very low. Table-5 and Fig.4 show that the proposed technique offers high robustness against various noises during image file compression. It justifies that our proposed algorithm is robust again channel noises and errors for different file formats. As it produces high throughput (up to 3.59 MB/Sec), therefore, the proposed technique offers high processing speed. It also reduces the size of the input files, so it is space efficient too.

The variable length coding techniques are popular as they produce better compression percentage. From the literature review, we found that variable length coding techniques are not robust to the channel noise. If some portion of the variable length code gets corrupted during the transmission, the whole compressed code will be uncompressible. But, though our proposed technique relates to the fixed length coding technique, it produces much greater compression efficiency than some well-known existing techniques in terms of compression ratio and bits per code. In our case if some portion is lost or gets corrupted during the transmission, it will not affect the decompression operation at all. In the section 5, we have compared our proposed algorithm with some existing compression techniques. We have also compared our proposed technique with some existing fixed length coding technique with the some well-known existing fixed length coding techniques with respect to their ability to produce compression ratios (*CR*) and percentage of compression (*CP*). From Table 1 to Table 4, we can see that the compression efficiency, produced by our proposed technique is far better than other existing fixed length coding and variable length coding based compression techniques in terms of *CR* and *CP*.

We have shown that we have achieved the three objectives from the results of our experiments. The proposed fixed length based compression is efficient for reducing channel congestion, reducing errors and reducing transmission delay during big data transmission. However, from our discussion it can be also seen that, our proposed technique is partially dependable of prefix code to a certain extend. So, if we transmit the information over a noisy network and some bits in the prefix code are corrupted during the transmission, our proposed technique is not capable to avoid information loss. As we see form Fig. 5 there are some information loss occurred during data retrieval, which implies that more research is needed to reduce the data loss. The throughputs shown in the Table-6 are good but it can be improved further via research to produce better processing speed and reduce the transmission delay.

References

[1] Lukin V.V., Zriakhov M.S., Ponomarenko N.N., Krivenko S.S., Zhenjiang M. 2010. Lossy compression of images without visible distortions and its application. IEEE, pp. 698-701.

[2] Goyal V.K., Fletcher A.K., Rangan S. 2008. Compressive sampling and lossy compression. Signal Processing Magazine, IEEE, Vol. 25, pp. 48-56.

[3] Al-Bahadili H. 2008. A novel lossless data compression scheme based on the error correcting Hamming codes. Computers & Mathematics with Applications, Vol. 56, pp. 143-150.

[4] Fout N., Kwan-Liu M. 2012. An Adaptive Prediction-Based Approach to Lossless Compression of Floating-Point Volume Data. Visualization and Computer Graphics, IEEE Transactions on, Vol. 18, pp. 2295-2304.

[5] Klein S.T., Shapira D. 2014. Practical fixed length Lempel–Ziv coding. Discrete Applied Mathematics, Vol. 163, pp. 326-333.

[6] Sugimoto K., Hattori R., Sekiguchi S.I. 2011. A novel high efficiency fixed length coding for video compression based on symbol probability estimation. pp. 1-4.

[7] Kattan A., Poli R. 2008. Evolutionary lossless compression with GP-ZIP*. ACM, pp. 1211-1218.

[8] Ye B., Zhao Q., Zhou D., Wang X., Luo M. 2011. Test data compression using alternating variable run-length code Integration. The VLSI Journal, Vol. 44, pp. 103-110.

[9] Baron D., Jacob T. 2012. Variable Length Compression of Codeword Indices for Lossy Compression. Signal Processing Letters. IEEE, Vol. 19, pp. 849-852.

[10] Gorev M., Ellervee P., 2010. Variable byte-length data compression algorithm. IEEE, pp. 353-356.

[11] Bhavsar K., Mehta U. 2011. Analysis of test data compression techniques emphasizing statistical coding schemes. ACM, pp. 1219-1224.

[12] Chuang C.-P., Chen G.-X., Liao Y.-T., Lin C.-C. 2012. A Lossless Color Image Compression Algorithm with Adaptive Arithmetic Coding Based on Adjacent Data Probability. IEEE, pp. 141-145.

[13] Tang J., Zhang X., Zhao L., Zou C. 2010. A novel arithmetic coding on data compression and encryption with asymptotic deterministic randomness. IEEE, pp. V2-10-V12-14.

[14] Sacaleanu D., Stoian R., Ofrim D.M. 2011. An adaptive Huffman algorithm for data compression in wireless sensor networks. IEEE, pp. 1-4.

[15] Golander A., Taharlev S., Glass L., Biran G., Manole S. 2013. Leveraging predefined huffman dictionaries for high compression rate and ratio. ACM, Article 19.

[16] Shun J., Zhao F. 2013. Practical Parallel Lempel-Ziv Factorization. IEEE, pp. 123-132.

[17] Santhanam N.P., Modha D. 2011. Lossy Lempel-Ziv like compression algorithms for memoryless sources. IEEE, pp. 1751-1756.

[18] Jain P., Jain A., Agrawal C. 2013. Effective dictionary based data compression and pattern searching in dictionary based compressed data. IEEE, pp. 1-6.

[19] Mansour A.M.A., Fouad M.A. 2013. Dictionary based optimization for adaptive compression techniques. IEEE, pp. 421-425.

[20] Marpe D., Schwarz H., Wiegand T. 2010. Entropy coding in video compression using probability interval partitioning. IEEE, pp. 66-69.

[21] Zheng Y., Qi C., Wang G. 2010. A New Image Pre-Processing for Improved Performance of Entropy Coding. IEEE, pp. 1-6.

[22] Nkom B. 2011. Concise schemes for realizing 1-Wire® cyclic redundancy checks. IEEE, pp. 70-79.

[23] Brown R.D. 2011. Reconstructing corrupt DEFLATEd files, digital investigation.Vol. 8, pp. S125-S131.

[24] Horvath K., Stogner H., Uhl A., Weinhandel G. 2011. Lossless compression of polar iris image data, Pattern Recognition and Image Analysis. Springer, pp. 329-337.