An Implementation of Efficient Text Data Compression

Virendra Kumar Swarnkar, Prof. K. J. Satao

Abstract: ***The size of data related to a wide range of applications is growing rapidly. Typically a character requires 1 Byte for storing. Compression of the data is very important for data management. Data compression is the process by which the physical size of data is reduced to save on memory or improve traffic speeds on a website. The purpose of data compression is to make a file smaller by minimizing the amount of data present. When a file is compressed, it can be reduced to as little as 25% of its original size which makes it easier to send to others over the internet. Data compression may take extensions such as zip, rar, ace, and BZ2. It is normally done using special compression software. Compression serves both to save storage space and to save transmission time. The aim of compression is to produce a new file, as short as possible, containing the compressed version of the same text. Grand challenges such as the human generated project involve very large distributed databases of text documents, whose effective storage and communication requires a major research and development effort. Several text compression schemes have been introduced for reducing storage space and transfer time via a computer network. The aim of this paper is to introduce a scheme for text data compression which will take less storage space, will yield better compression rate and compression ratio.***

Index Terms—Data compression, Data decompression, Compression algorithms, Compression ratio, Compression rate.

I. INTRODUCTION

Generally commercial computer system contain significant redundancy when data stored on disks is transferred over communication links. A mechanism or procedure which recodes the data to lessen the redundancy could possibly double or triple the effective data densities stored or communicated. Moreover, if compression is automatic, it can also aid in the rise of software costs. Several problems are encountered when common compression methods are integrated into computer systems. For example, the first one is, poor runtime execution speeds interfere in the attainment of very high data rates. The second one is, most compression techniques are not flexible enough to process different types of redundancy. The third one is blocks of compressed data that have unpredictable lengths present will yield storage space management problems. Each compression strategy solves a different set of these problems and consequently, the use of each strategy is restricted to applications where its natural weakness presents no critical problems. Data compression is the process of encoding a body of data D into a smaller body Ω(D). If Ω(D) can be decoded back to D, without loss of information, then it is said to be a reversible data compression. The situation in which some acceptable approximation to D is obtained in the decoding is known as non-reversible data compression. Usually non-reversible algorithms are used for image compression. In this paper we have discussed only about ASCII text compression techniques. The primary advantages in utilizing data compression techniques are: Data storage space such as disks or tapes can be greatly reduced with data compression. A compressed file generally takes less storage space than an uncompressed one. Also, compressed files can be decompressed when users demand the original copy. With data compression the same amount of information can be sent over a network in much less time than decompressed data. For data communications, a sender can compress data before transmitting it and the receiver can decompress the data after receiving it. The main parameters of interest in data compression are compression ratio, compression rate, and storage space.

Text files are usually stored by representing each character with an 8-bit ASCII code. Data compression is important for storage systems because it allows more bytes to be packed into a given storage medium, when the data is compressed. By using compression technique file transfer time can be reduced and also communications bandwidth can be reduced [1] .

Using the right terminology we would say that the first is called a model that predicts the probability of the source, it then makes a probability distribution. The second one is the coder, it makes and emits codes based on the probabilities assigned by the model. The closer the predictions made by the model to the data, the more compression we can achieve, because we'll assign less bytes to most of the data.

A text segment is a collection of words and a word consists of characters. All the characters are unique and they are the basic units of word. That's why to store a text segment it is needed to store all the words separately. To store a word, all the characters that the word contains needed to be stored. For this type of storing mechanism a huge amount of disk space is needed and in the current world this technique is used. But this type of system of storing text makes the text segment consume more space. Let us assume that a text segment

Virendra Kumar Swarnkar, Computer Science & Engineering; Bharti College of Engineering & Technology, Durg, Chhattisgarh, India, swarnkarvirendra@gmail.com

Prof. K. J. Satao, Computer Science & Engineering; Head, Department of Information Technology, Rungta College of Engineering & Technology, Bhilai, Chhattisgarh, India, kjsatao@gmail.com

An Implementation of Efficient Text Data Compression

contains some word "n" times, which is 7 characters in lengths (take it as average length) then for repeated presence of the same word of "n" times it need n*7 Bytes. If some sort of indexing within the text segment is done, then the text segment size can be reduced. But this process is still not effective because it needs extra space to make the indexing table and sometimes it may increase the file size rather than decreasing. So in this paper we have implemented a new scheme which will yield better performance in terms of less storage space, compression rate and compression ratio [2] .

II. METHODOLOGY

In this paper we have discussed data compression and decompression, and illustrated how to compress and decompress data, efficiently and effectively, from the Java applications using the java.util.zip package. While it is possible to compress and decompress data using tools such as WinZip, gZip and Java Archive, these tools are used as standalone applications. It is possible to invoke these tools from the Java applications, but this is not a straightforward approach and not an efficient solution. Figure 1.1 shows a basic data-compression block diagram.

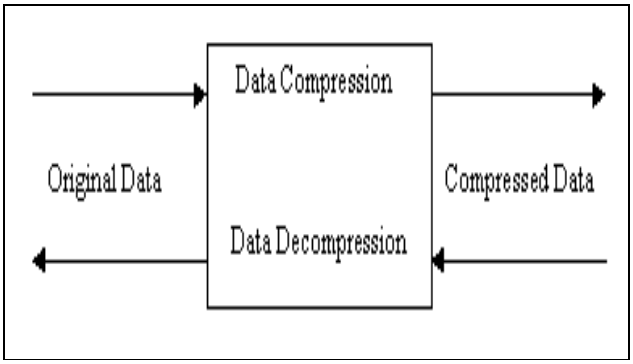

Fig. 1.1: Data-compression block diagram

The simplest type of redundancy in a file is the repetition of characters. For example, consider the following string:

BBBBHHDDXXXXKKKKWWZZZZ

This string can be encoded more compactly by replacing each repeated string of characters by a single instance of the repeated character and a number that represents the number of times it is repeated. The earlier string can be encoded as follows:

4B2H2D4X4K2W4Z

Here "4B" means four B's, and 2H means two H's, and so on. Compressing a string in this way is called run-length encoding.

As another example, consider the storage of a rectangular image. As a single color bitmapped image, it can be stored as shown in Figure 1.2.

Figure 1.2: A bitmap with information for run-length encoding

There are many benefits to data compression. The main advantage of it, however, is to reduce storage space requirements. Also, for data communications, the transfer of compressed data over medium results in an increase in the rate of information transfer. In this paper the following procedures are used for compression and decompression of text data.

Procedure for Compression:

1. User will write text on text area or paste text from other document.
2. User can open any document from the menu compress.
3. Software takes all text from text area in to a string object.
4. After that it converts data in byte array format and passes to the compress class.
5. Imports package java.zip.Deflater.
6. Creates object of ByteArrayOutputStream and sets to null.
7. Creates object of deflater.
8. Sets level of Deflater by calling method setLevel() and sends argument as best compression of Deflater.
9. Sets input data in the form of bytes array by calling method setInput() .
10. Finishes deflater action by calling method finish().
11. Allocates space for ByteArrayOutputStream object.
12. Creates byte array object and allocates space.
13. Initializes size of deflated content and writes on ByteArrayOutputStream object.

Procedure for Decompression:

1. Software imports package java.zip.Deflater.
2. Creates object of ByteArrayOutputStream and sets to null.
3. Creates object of inflater by allocating object.
4. Sets level of Inflater by calling method setLevel() and sends argument as best compression of Inflater.
5. Sets input data in the form of bytes array by calling method setInput().
6. Finishes inflater action by calling method finish().
7. Allocates space for ByteArrayOutputStream object.
8. Creates byte array object and allocates space.
9. Initializes size of deflated content and writes on ByteArrayOutputStream object.

III. RESULTS

In this paper, we have implemented the technique i.e. .veer format with text files which consist of various types of data viz. random, only digits (0 to 9), only characters (a to z), mix of digits and characters (0 to 9 and a to z) and a file which consist of only zeroes. By using the implemented technique i.e. .veer format, we have calculated the compressed format in order to know the storage space. We have also calculated the compression rate as well as compression ratio. With the comparison of both file it is clearly shown the difference between two different format file. From the following table it can be observed that, the implemented technique takes less storage space than another one. The following table gives the result :

S.NO	File Name	File Content	.Txt File (Size in KB)	.Veer File (Size in KB)
1	FILE 1	Random	28.9	5.91
2	FILE 2	0 TO 9	860	3.51
3	FILE 3	a to z	198	0.73
4	FILE 4	0 TO Z	968	4.02
5	FILE 5	only 0	169	0.68

Compression ratio is calculated as per the following formula:-

Compression ratio = (Compressed Size /Uncompressed Size)*100

The following table gives the result of Compression Ratio:

S.No.	File Name	Compression Ratio
1	FILE 1	20.44
2	FILE 2	0.40
3	FILE 3	0.37
4	FILE 4	0.41
5	FILE 5	0.40

Compression rate is calculated as per the following formula:-

Compression rate= 100-Compression Ratio

The following table gives the result of Compression Rate :

S.No.	File Name	Compression Rate
1	FILE 1	79.55
2	FILE 2	99.59
3	FILE 3	99.62
4	FILE 4	99.58
5	FILE 5	99.59

The two figures below represent the screen shots of .VEER compression technique:

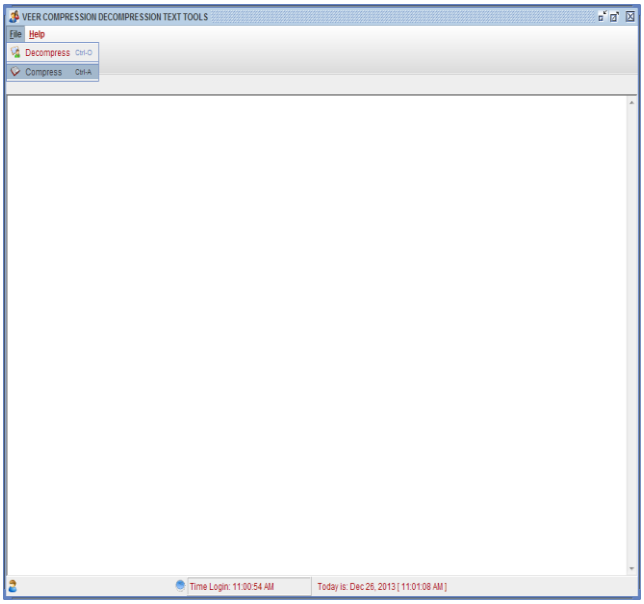

Figure No.1.3

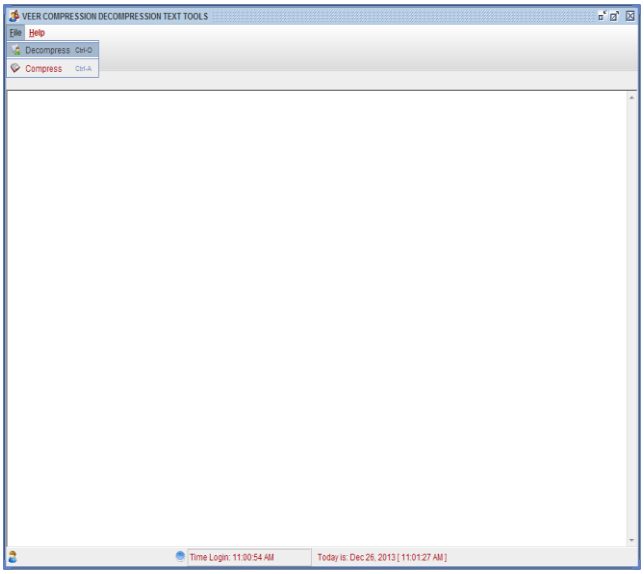

Figure No.1.4

IV. SOURCE CODE ROUTINES FOR COMPRESSION AND DECOMPRESSION

For compression :-

```
public class Compress {

   public byte[] compressByteArray(byte[] bytes){

      ByteArrayOutputStream baos = null;
      Deflater dfl = new Deflater();
      dfl.setLevel(Deflater.BEST_COMPRESSION);
      dfl.setInput(bytes);
      dfl.finish();
      baos = new ByteArrayOutputStream();
      byte[] tmp = new byte[4*1024];
      try{
         while(!dfl.finished()){
            int size = dfl.deflate(tmp);
            baos.write(tmp, 0, size);
         }}
     catch (Exception ex){

      } finally {
         try{
            if(baos != null) baos.close();
         } catch(Exception ex){}
      }

      return baos.toByteArray();
   }

   public static void main(String a[]){

      Compress mbc = new Compress();
      byte[] content = mbc.compressByteArray("Compress
java2novice.com".getBytes());
      System.out.println(new String(content));
   }
}
```

For decompression : -

```
public class Decompress {

public byte[] decompressByteArray(byte[] bytes){

      ByteArrayOutputStream baos = null;
      Inflater iflr = new Inflater();
      iflr.setInput(bytes);
      baos = new ByteArrayOutputStream();
      byte[] tmp = new byte[4*1024];
      try{
         while(!iflr.finished()){
            int size = iflr.inflate(tmp);
            baos.write(tmp, 0, size);
         }
      } catch (Exception ex){

      } finally {
         try{
            if(baos != null) baos.close();
         } catch(Exception ex){}
      }

      return baos.toByteArray();
   }
```

V. CONCLUSION

This paper presents an implementation of efficient text data compression. It describes the usage of java.util.zip package to compress and decompress the text data, as well as it shows how to compress and decompress serialized data to save disk space. In this paper, we have observed that the contents of the file (i.e. the number of different characters or symbols and the frequency for each symbol) are effective factors on the performance of the data compression. Data compression still is an important topic for research in these days, and has many applications.

REFERENCES

[1] Data Compression Techniques on Text Files: A Comparison Study, *July 2011*

[2] Charles Bloom "Compression : News Postings : Kraft Inequality" (http://www.cbloom.com)

[3]http://en.wikipedia.org/wiki/Burrows%E2%80%93Wheeler_transform

[4] Data compression theory and algorithms, http://www.maximumcompression.com/algoritms.php

[5] **Made Agus Dwi Suarjaya ,** A New Algorithm for Data Compression Optimization, *IJACSA, Vol. 3, No.8, 2012*

Virendra Kumar Swarnkar, did his B.E from Rungta College of Engineering and Technology, Bhilai(C.G.). Currently he is pursuing M.Tech in Software Engineering from Rungta College of Engineering & Technology, Bhilai, Chhattisgarh, India. He has published 3 papers in international journals. His area of interest in Software Engineering , Operating System, Database Management System, etc.

Prof. K. J. Satao is Professor of Computer Science & Engineering and Head of Information Technology Department at Rungta College of Engineering & Technology, Bhilai, Chhattisgarh State, India. as obtained M.S. degree in Software Systems from Birla Institute of Technology and Science, Pilani, Rajasthan State, India in 1991. He has published over 50 Papers so far in various reputed National & International Journals, Conferences, and Seminars. He is Ex. Dean of Computer & Information Technology faculty in Chhattisgarh Swami Vivekanand Technical University, Bhilai, India(A State Government University). He is Ex. member of the Executive Council and the Academic Council of the University. He is a member of the Computer Society of India and the Indian Society for Technical Education. He has worked in various Engineering Colleges for over 26 Years and has about 4 Years industrial experience as well. His areas of research include Operating Systems, Editors & IDEs, Information System Design & Development, Software Engineering, Modelling & Simulation, Operations Research, etc.