IJCSI International Journal of Computer Science Issues, Volume 15, Issue 3, May 2018
ISSN (Print): 1694-0814 | ISSN (Online): 1694-0784
www.IJCSI.org https://doi.org/10.5281/zenodo.1292416

Dictionary Based Compression using Bit Wise Technique for Cloud Migration

Shivam Saini[1], Shivani Pathak[2], Dr. Sandeep Sharma[3]

[1] **Department of Computer Engineering and Technology, Guru Nanak Dev University, Amritsar-143005, India**

[2] **Department of Computer Engineering and Technology, Guru Nanak Dev University, Amritsar-143005, India**

[3] **Department of Computer Engineering and Technology, Guru Nanak Dev University, Amritsar-143005, India**

Abstract

In era of cloud computing, data migration is an important issue to minimize cost and time incurred to transmit the data across the network. Data compression is a technique that helps to significantly reduce the transmission time and thus reducing cost. In this paper, a new dictionary-based compression algorithm using bit wise technique has been introduced. The algorithm constitutes two dictionaries, primary dictionary (Static) and secondary dictionary (Dynamic). The data to be transmitted is first divided into chunks of fixed size(16K) and then transmitted after compression along with secondary dictionary. The proposed algorithm will conduce to reduce the number of data packets that needs to be transmitted with better bandwidth utilization thereby augmenting the transmission rate of data.

Keywords*: Cloud computing, Virtual machine migration, Compression, Decompression.*

1. Introduction

Cloud computing has become a significant technology trend in various IT infrastructure. It can be defined as a new style of computing in which dynamically scalable and often virtualized resources are provided as a service over the Internet. Cloud computing delivers infrastructure, platform, and software that are made available as subscription-based services in a pay-as-you-go model to consumers. These services are referred to as Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS) in industries [1]. Advantages of the cloud computing technology include cost savings, high availability, and easy scalability. Cloud computing typically uses a network of data centers that are geographically dispersed. The distance between clients and applications is impacted by geographical distance. In Cloud, a service provider is responsible for managing the cloud resources as a manager. To run the applications, it will allocate resource by virtual machine (VM) to each user and cloud computing is one of the illustrious technology because of its virtualization facet [2]. Virtualization is used in system management to deal with load balancing, efficient resource utilization etc. Live virtual machine (VM) migration transfers ongoing process from overloaded virtual machine to another virtual machine [3]. It has become an extremely powerful tool for system management in a variety of key scenarios, such as VM load balancing, fault tolerance, power management and other applications. The performance of live VM migration is affected by various factors. The network transmission rate together with the configuration of migration algorithm is one of the cardinal factor that affects it [4]. In general approach, physical memory image is pushed across network to the new destination while the source VM continues running. Pages dirtied during the migration must be iteratively resent to ensure memory consistency. By iterative it means that pre-copying occurs in several rounds and the data to be transmitted during a round are the dirty pages generated in the previous round. But the data to be migrated is too large and consumes a lot of time

and storage and thus increasing the number of iterations and cost for data migration. To overcome these limitations, data compression technique can be implemented. Data Compression is a process by which the file size is reduced by re- encoding the file data to use fewer bits of storage than the original file [5]. The original file can then be recreated from the compressed representation using a reverse process called decompression. A simple characterization of data compression is that it involves transforming a string of characters in some representation (such as ASCII) into a new string (of bits, for example) which contains the same information but whose length is as small as possible. The same program is used to decompress(decrypt) the data so that it can be read as the original data. There are two mainly two types of Data Compression Lossy Compression and Lossless Compression [6]. A lossy data compression method in which the data may be lost and retrieved data after decompression may not be exactly same as the original data whereas a Lossless data compression is a technique that allows the exact original data to be reconstructed from the compressed data as no data is lost. Since we are only dealing with the text in this paper, we are considering the lossless compression. The main motive of this paper is to reduce the storage, time and cost that.is incurred during the live virtual machine migration through data compression.

2. Earlier work

Virtualization technology allows multiple operating systems to run concurrently on the same physical machine. Over times various new methods and technologies have been introduced. A number of live migration techniques that transfers the whole system at run time state, including CPU state, local disk storage and memory data of the virtual machine has been introduced. C. Xianqin, et al. presented a technique to improve the performance of live migration [7]. In this an optimized iterative pre-copy algorithm is used which reduce the dirty rate of VM. E. Zaw et al. proposed Network aware VM Migration strategy [8]. In this, migration of VM is purely base on network traffic and network latency and various strategies are used for migration. H. Jin et al. presented an approach, VM placement and migration for data intensive application, which helps to minimize data transfer time [9]. The approach places the VMs on target physical machines by considering the network conditions between the physical machines and the data storage. From time to time to improve the performance of virtual machine migration a number of methodologies have been introduced. Huffman coding works on the entropy of the words. In Huffman coding [10] the most frequently occurring symbols are given the shortest code words. These codes are created by storing the symbols of the alphabet in a binary tree data structure according to their entropy i.e. their probability of occurrence. A 0 is appended while traversing the tree to the right child, and a 1 while traversing the tree to the left child. Therefore, symbols which occur frequently are provided the shortest codes. A variety of data compression algorithms has been designed over years. Run Length Encoding (RLE) Algorithm, Huffman Encoding Algorithm, Adaptive Huffman Encoding Algorithm, Shannon Fano Algorithm Arithmetic Encoding Algorithm and Lempel Zev Welch (LZW) Algorithm has served as fundamental techniques for compression and decompression. In [11] a new dictionary and memory-based text compression technique is presented. The original words in a text file are transformed into codewords using a dictionary of frequently used words in English language.

3. Proposed dictionary-based compression using bit wise technique

The proposed algorithm deals with data compression by mainly focusing on the most frequently used words in English language. It is a dictionary based [12] algorithm in which two dictionaries are used**.** The basic idea is to take the advantage of commonly occurring words by using a dictionary [13]. The repeating occurrences in the dataset are replaced by a code. The data to be transmitted is first divided into blocks of fixed size of 16K as S_0, S_1, S_2,…,S_n. Each block is compressed and immediately transferred as CS_0, CS_1, CS_2, …, CS_n. At the encoding of each block, a new secondary dictionary (dynamic) [14] is created by replacing the previous one, whereas primary remains static. This secondary dictionary is always transmitted along with the compressed data block over the network each time a new a block is compressed. At the other end, the decoder uses primary dictionary already available, and the secondary dictionary that is received over the network along with the compressed data block for decoding. In this paper, we have also focused on the spaces present in the text for better compression results. In any text file, there are about 20% spaces. The algorithm has been designed in such a way that some spaces between words can be removed without any difficulty to retrieve the original words of the target text file

3.1 Primary dictionary

The Primary dictionary is a static dictionary that comprises 64 most frequently used words in English having length 2 and 3**.** It is built up before compression occurs, and it does not change while the data is being compressed. Each word in the dictionary is assigned an 8-bit code which is fixed and whose first two bits are always 10 i.e. (10XX XXXX) where each X can take a value of 0 or 1.

WORD	CODE
is	1000 0000
the	1000 0001
The	1000 0010
of	1000 0011
in	1000 0100
it	1000 0101
to	1000 0110
as	1000 0111
on	1000 1000
at	1000 1001
are	1000 1010
and	1000 1011
for	1000 1100
but	1000 1101
all	1000 1110
its	1000 1111
can	1001 0000
if	1001 0001
so	1001 0010
or	1001 0011
do	1001 0100
an	1001 0101
no	1001 0110
be	1001 0111
we	1001 1000
he	1001 1001
by	1001 1010
me	1001 1011
my	1001 1100
up	1001 1101
go	1001 1110
us	1001 1111
am	1010 0000
not	1010 0001
you	1010 0010
any	1010 0011
had	1010 0100
her	1010 0101
was	1010 0110
one	1010 0111
our	1010 1000
out	1010 1001
day	1010 1010
get	1010 1011
has	1010 1100
him	1010 1101
his	1010 1110
how	1010 1111
man	1011 0000
new	1011 0001
now	1011 0010
old	1011 0011
see	1011 0100
two	1011 0101
way	1011 0110
boy	1011 0111
did	1011 1000
let	1011 1001
put	1011 1010
say	1011 1011
she	1011 1100
too	1011 1101
use	1011 1110
who	1011 1111

Fig. 1 Primary dictionary

3.2 Secondary dictionary

Besides primary dictionary, secondary dictionary is not fixed [14]. The dictionary can constitute utmost 64 words. Initially, the dictionary is empty. It will add the 2 and 3 length words that are present in the block which were not found in the primary dictionary. Besides this, the dictionary will also add 4 and 5 length words until it is vacant. As in the primary dictionary, each word in this dictionary is also assigned an 8-bit code and whose first two bits are always 11 i.e. (11XX XXXX) where each X can take a value of 0 or 1. Every time when a new word is encountered satisfying the conditions, it is added just after the last entry in the dictionary. Starting code for first string in secondary dictionary will be fixed. The next codes can be obtained by doing increment of one step. Whenever the block is encoded, a new secondary is maintained for that block.

EXAMPLE

The following text has been encoded using Primary and Secondary Dictionary

Dolphins are regarded as the friendliest creatures in the sea and stories of them helping drowning sailors have been common since Roman times. The more we

learn about dolphins when they are ill, care in the community, as we do.

WORD	CODE
sea	1100 0000
them	1100 0001
have	1100 0010
been	1100 0011
since	1100 0100
Roman	1100 0101
more	1100 0110
learn	1100 0111
about	1100 1000
when	1100 1001
they	1100 1010
ill,	1100 1011
care	1100 1100
do.	1100 1101

Fig. 2 Secondary dictionary

3.3 ENCODING

0100 0100 0110 1111 0110 1100 0111 0000 0110
1000 0110 1001 0110 1110 0111 0011 0010 0000
1000 1010 0111 0010 0110 0101 0110 0111 0110
0001 0111 0010 0111 0010 0110 0101 0110 0111
0110 0001 0111 0010 0110 0100 0110 0101 0110
0100 0010 0000 1000 0111 1000 0001 0110 0110
0111 0010 0110 1001 0110 0101 0110 1110 0110
0100 0110 1100 0110 1001 0110 0101 0111 0010
0111 0100 0010 0000 0111 0011 0111 0010 0110
0101 0110 0001 0111 0100 0111 0101 0111 0010
0110 0101 0111 0011 0010 0000 1000 0100 1000
0001 1100 0000 1000 1011 0111 0011 0111 0100
0110 1111 0111 0010 0110 1001 0110 0101 0111
0011 0010 0000 1000 0011 1100 0001 0110 1000
0110 0101 0110 1100 0111 0000 0110 1001 0110
1110 0110 0111 0010 0000 0110 0100 0111 0010
0110 1111 0111 0111 0110 1110 0110 1001 0110
1110 0110 0111 0010 0000 0111 0011 0110 0001
0110 1001 0110 1100 0110 1111 0111 0010 0111
0011 0010 0000 1100 0010 1100 0011 0110 0011
0110 1101 0110 1101 0110 1111 0110 1110 0010
0000 1100 0100 1100 0101 0111 0100 0110 1001
0110 1101 0110 0101 0111 0011 0010 1110 0010
0000 1000 0010 1100 0110 1001 1000 1100 0111
1100 1000 0110 0100 0110 1111 0110 1100 0111
0000 0110 1000 0110 1001 0110 1110 0111 0011
0010 0000 1100 1001 1100 1010 1000 1010 1100
1011 1100 1100 1000 0100 1000 0001 0110 0011
0110 1111 0110 1101 0110 1101 0111 0101 0110
1110 0110 1001 0111 0100 0111 1001 0010 1110

4. Compression algorithm

The data set is divided into blocks each of length 16K as S_0, S_1, S_2…, S_n. The algorithm starts by initializing secondary dictionary as empty. The algorithm reads the input file word by word and stores it in a variable 'word' until the EOF (End of File) is reached. This input file is tokenized at spaces. Cfile is the file that will be obtained after compression and is initially empty. This algorithm first checks the length of the word and stores it in a variable 'length'. If the length is 2 or 3, it checks the word's availability in Primary Dictionary. If the word is present, then the corresponding code is written in the Cfile. If not found, the search is done in secondary dictionary. If it is found there, then the corresponding code is written in the Cfile and the next word is read. Else the algorithm checks for the vacancy in secondary dictionary. If vacant, then word is added to the dictionary and a code is assigned to the word by incrementing the previous code by 1 and is also written in the Cfile. If no space is available, then ASCII value for each character in the word is written in the Cfile. A space is also added after the word. If the word length is 4 or 5 then the same procedure is applied to check the availability of word in secondary dictionary as described above. In case of any other length, the ASCII value for each character in the word is written in the Cfile. A space is also added after the word.

This algorithm is applied to each block of data set and compressed into CS_0, CS_1, CS_2,…, CS_n.

$S_0 \longrightarrow CS_0$

$S_1 \longrightarrow CS_1$
$\vdots \qquad \vdots$
$S_n \longrightarrow CS_n$

After compression, each block is sent in iterations as (CS_0+SD_0), (CS_1+SD_1), (CS_2+SD_2), …,$(CS_n+ SD_n)$, where SD_i represents the secondary dictionary formed for i^{th} block (i=0,1,2, …,n)

Total Size of uncompressed data = $\sum_{i=0}^{n} S_i$ (1)

Total Size of compressed data = $\sum_{i=0}^{n}(CS_i + SD_i)$ (2)

```
int length=0;
String Cfile="";
   String word="";
   int secondary_counter=0;
   while (! EOF)
   {
      word= read_word (inputfile);
      length=word.length;
      if (length = = 2 || length = = 3)
      {
         boolean findP = check_Primary_Dictionary (word);
         if (findP = = true)
         {
            Append the corresponding code for that word in
the Cfile;
         }
         else if (findP = = false)
         {
            check_Secondary(word);
         }
      }
      else if (length = = 4||length = = 5)
      {
         check_Secondary (word);
      }
      else
      {
       Append the ASCII value for each character of word in
the Cfile
      }
   }
   // Function checking for word in Secondary Dictionary
 check_Secondary (String word)
   {
      boolean findS=check_Secondary_dictionary (word);
      if (findS = = true)
      {
         Append the corresponding code for that word in
Cfile;
      }
      else
      {
         If (secondary_counter<64)
         {
            Add the word to Secondary Dictionary and assign
            the code by incrementing previous code by 1.

            Append the assigned code for that word in Cfile;
         }
         else
         {
          Append the ASCII value for each character of word
in
          the Cfile;
            Append the ASCII value for space in the Cfile;
         }
      }
   }
```

Fig.3 Compression algorithm

Percentage Compression =

$$\left[\frac{\{(\sum_{i=0}^{n} S_i)-(\sum_{i=0}^{n}(CS_i+SD_i))\}}{(\sum_{i=0}^{n} S_i)}\right]*100 \quad (3)$$

5. Decompression algorithm

Once the data has been transmitted along with secondary dictionary over the network, it will be received by the recipient where primary dictionary is already available. Using these two dictionaries, the encoded data can be recovered easily without any loss. Dfile is the string file where original data will be retrieved from compressed file after decompression. The Decompression Algorithm starts by reading first 8 bits from received data and checks the first two bits. If the bits are 00 or 01, then the ASCII character corresponding to those 8 bits is written without checking any of the dictionaries. If the bits are 10, those 8 bits are looked up in the Primary Dictionary and the corresponding word is written and a space is automatically inserted. Else if the bits are 11, those 8 bits are looked up in the Secondary Dictionary and the corresponding word is written and a space is automatically inserted. After the decoding of 8 bits, next 8 bits are fetched and same process is repeated until EOF. This is how the algorithm works efficiently without any loss of data.

```
String Dfile;
   While (! EOF)
   {
      Read next 8 bits;
      If (first two bits==00||first two bits==01)
      {
        Append the character in Dfile corresponding
to 8 bits from ASCII table;
      }
      else if (first two bits==10)
      {
         Search in Primary Dictionary and append the
corresponding word in Dfile;
         Append a space in the Dfile.
      }
      else if (first two bits==11)
      {
         Search in Secondary Dictionary and append
the corresponding word in Dfile;
         Append a space in the Dfile.
      }
   }
```

Fig.4 Decompression algorithm

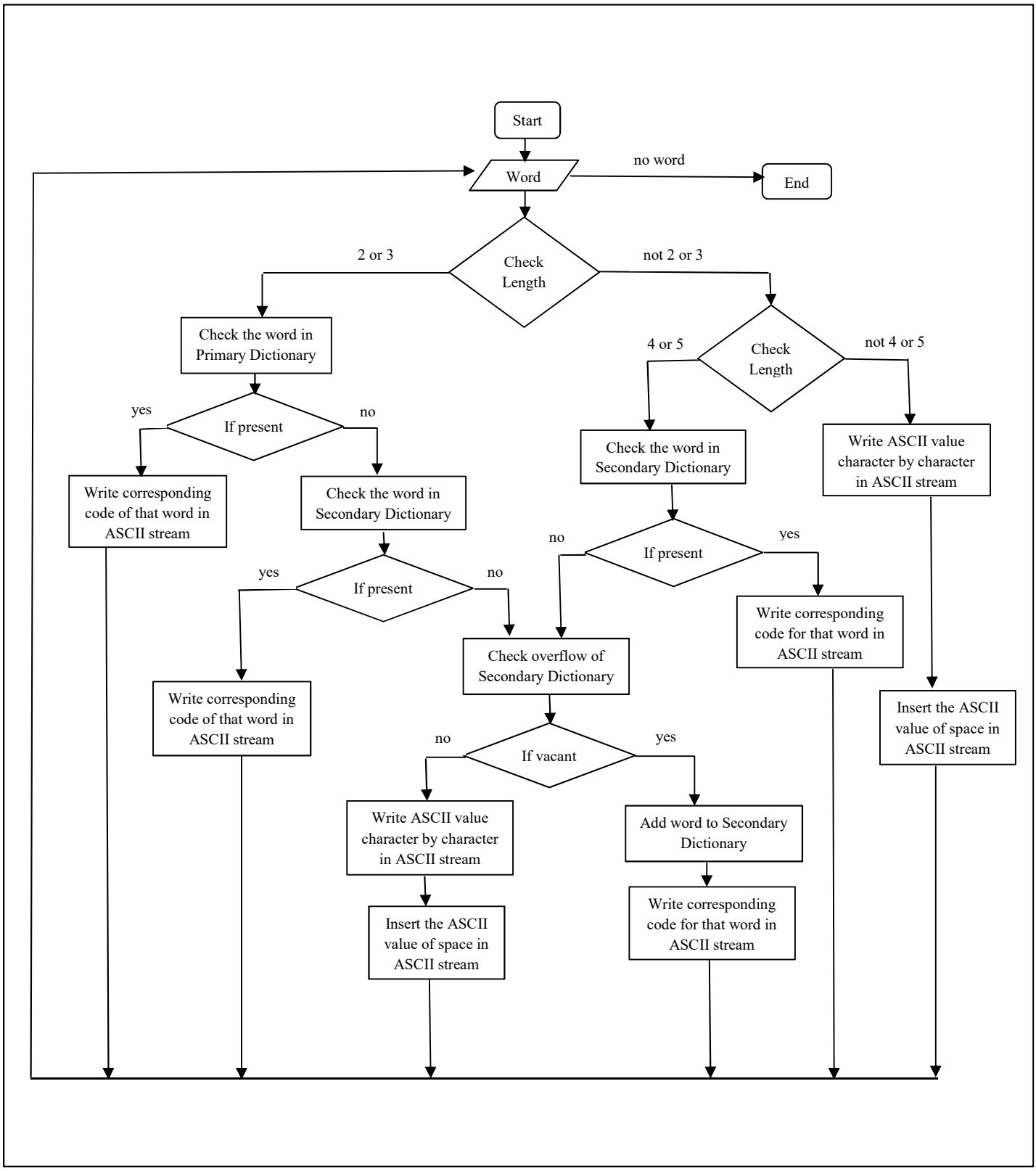

Fig. 5 Flowchart of Compression algorithm

6. Implementations and results

The proposed algorithm when implemented on different yields good results. The algorithm works on different file sizes irrespective of the content of the file and yields the almost same results. Since the decompressed files are rebuild using the dictionary so there is no loss of information and hence the algorithm proves itself to be lossless. Although the algorithm takes up some time for compression, but the total time taken for compression along with data transmission is less as compared to the time taken for transmission of uncompressed data. The table below shows the comparison between the uncompressed data and the data after compression and shows the percentage compression achieved.

Table 1. Results

	Uncompressed Data Size (bits)	Compressed Data Size (bits)	%age Compression
Data Set 1	10976	8592	21.72
Data Set 2	104944	81168	22.66
Data Set 3	230664	191840	17

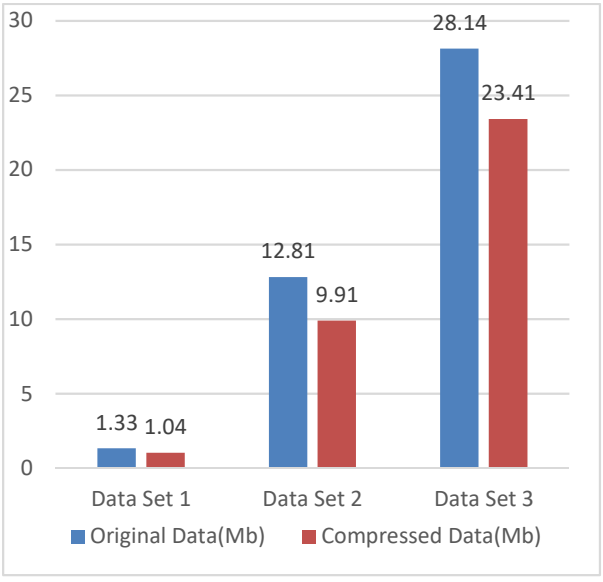

Fig. 6 Comparison of original and compressed data

7. Conclusions

In contrast to the existing algorithms that solely focus on either data compression or on data migration, this algorithm considers both the aspects and produce efficient results. The results show that by implementing the algorithm, a significant amount of time and cost for data transmission can be minimized during virtual machine migration which plays a pivotal role in cloud computing to ensure the Quality of Service. The mechanism that the algorithm follows is quite simple and can be easily implemented without much complexities. Also, the creation of a dynamic dictionary which is accessible while decompressing, is another conspicuous feature which in turns makes the algorithm a lossless one. The space complexity may be reduced by selection of proper data structure.

References

[1] John W. Rittinghouse and James F. Ransome CRC Press, Taylor & Francis Group, Boca Raton, FL, 2010, "CLOUD COMPUTING: Implementation, Management, and Security"

[2] Rong Yu, Jiefei Ding, Sabita Maharjan, Stein Gjessing, Yan Zhang, Danny H.K. Tsang, "Decentralized and Optimal Resource Cooperation in Geo-Distributed Mobile Cloud Computing", IEEE Transactions on Emerging Topics in Computing, DOI 10.1109/TETC.2015.2479093

[3] Hai Jin, Li Deng, Song Wu, Xuanhua Shi, Xiaodong Pan," "Live Virtual Machine Migration with Adaptive Memory Compression", Huazhong Universityof Science and Technology, Wuhan, 430074, China

[4] Analysis and Survey of Issues in Live Virtual Machine Migration Interferences, Ms. Tarannum Bloch, Dr. R Sridaran, Dr. Prashanth CSR, International Journal of Advanced Networking Applications (IJANA), ISSN No.: 0975-0290

[5] Mark Nelson and Jean-Loup Gaily, "The Data Compression Book", Second Edition, M&T Books.

[6] Bhupinderjit kaur, "International Journal of Emerging Technology and Advanced Engineering",

ISSN 2250-2459, ISO 9001:2008 Certified Journal, Volume 3, Issue 7, July 2013)

[7] C. Xianqin and G. Xiaopeng, "Application-Transparent Live Migration for Virtual Machine on Network Security Enhanced Hypervisor", China communications, (2012).

[8] E. Zaw and N, Thein, "Improved Live VM Migration using LRU", International Journal of Computer Science and Telecommunications, (2011). March

[9] H. Jin and L. Deng and S. Wu, "Live Virtual Machine Migration with Adaptive Memory Compression", (2010).

[10] D. A. Huffman, "A method for the construction of minimum redundancy codes," *In Proc. IRE 40*, volume 10, pages 1098–1101, September 1952.

[11] Md. Ziaul Karim Zia1, Dewan Md. Fayzur Rahman2, and Chowdhury Mofizur Rahman3, Two-Level Dictionary-Based Text Compression Scheme, 11th International Conference on Computer and Information Technology (ICCIT 2008)

[12] Debashis Chakraborty, Debajyoti Ghosh, Piyali Ganguly, "A Dictionary based Efficient Text Compression Technique using Replacement Strategy", International Journal of Computer Applications (0975 – 8887) Volume 116 – No. 16, April 2015

[13] Mani Arora, Derick Engles and Sandeep Sharma, "MDS Algorithm for Encryption", *Journal of Computer Science* Volume 11, Issue 3

[14] Debashis Chakraborty, Sandipan Bera, Anil Kumar Gupta and Soujit Mondal, "Efficient Data Compression using Character Replacement through Generated Code", IEEE NCETACS 2011, Shillong, India, March 4-5,2011, pp 334.

Shivam Saini Currently pursuing Bachelor of Technology in Computer Science and Engineering from Guru Nanak Dev University, Amritsar, India. His research interests include cloud computing and data analytics.

Shivani Pathak Currently pursuing Bachelor of Technology in Computer Science and Engineering from Guru Nanak Dev University, Amritsar, India. Her research interests include cloud computing and IOT.

Dr. Sandeep Sharma Received Ph. D. in Parallel Processing in 2010 from Guru Nanak Dev University, Amritsar, India. He is a professor and Head of the Department of Department of Computer Engineering and Technology at Guru Nanak Dev University. His research interests include Parallel and Cloud Computing, Big Data and Data Sciences. He is a member of Expert Committee to evaluate ongoing Major Research Projects for Sciences, Engineering & Technology by UGC (2017).