International Journal of Psychosocial Rehabilitation, Vol. 24, Issue 06, 2020
ISSN: 1475-7192

IMPROVISED SECURITY AND COMPRESSION ALGORITHM ON HUFFMAN ENCODING

[1]M Vijayakumar [2]Dr. Shiny Angel

***ABSTRACT**--Huffman Encoding is an absolute algorithm of data encryption and size compression that converts streams of multi-typed data into streams of binary data and enhances the pattern occurrences in the same. The yield of huffman encoding calculation is a specific kind of ideal prefix code, generally referred as lossless data compression. A combination of two previously defined but now obsolete algorithms of data compression that, when combined and executed over test cases of textual data can compress at massive ratios. The procedure also involves a minimal amount of system resources such as memory and RAM. This feature makes the procedure immensely efficient and beneficial to cases of data transmission where the system resources available are less and/or the bandwidth of the channel is insufficient. The method of de-duplication works efficiently on data with larger inherent or generated patterns. Hence, the output of Huffman Encoding will work as an appropriate input for the de duplication module. Data de-duplication, which can also be referred as an efficient compression is a process that can be used to eliminate redundant patterns present in data thus decreasing the storage overhead. It makes sure that only one unique illustration of a pattern is maintained on the source of storage. Repeated data blocks in files are substituted with a referential pointer that points to its unique copy of data pattern.*

***Keywords**-- Data Encryption and Compression, Redundancy, Storage Overhead Reduction, de-duplication, optimal prefix code.*

I. INTRODUCTION

Data compression can be viewed as the way toward altering information utilizing less stockpiling required than the first structure. Compression gives decrease in storage efficiency, time of transmission and data transfer capacity. Compression very well may be comprehensively named lossy and lossless. Huffman coding is a lossless compression method that can decrease the storage space required to store the data banking on the occurrence probability. It depends on the possibility that more frequent letters in an instance of data ought to have less bits as representation than the less normal letters.

Huffman Encoding method of compression was exhibited by David A. Huffman. Pointless reiteration of characters or parts of information in a document is known as redundancy. It signifies the fundamental distinction among lossy and lossless pressure. It detects and modifies only the excess in the information to perform the

[1] Department of Computer Science Engineering, SRM Institute of Science and Technology, Kattankulathur, Tamil Nadu, India, vijayakm@srmist.edu.in,

[2] Department of Software Engineering, SRM Institute of Science and Technology, Kattankulathur, Tamil Nadu, India, shinyant@srmist.edu.in

Received: 22 Sep 2019 | Revised: 13 Oct 2019 | Accepted: 15 Jan 2020

compression. So, if there is no redundancy of data in any file then the file cannot be efficiently compressed by this method of lossless data compression.

The method of de-duplication works best on data with larger inherent patterns. Hence, the output of Huffman Encoding will work as appropriate input for the de duplication module. The method of de-duplication is characterized by the recognition of patterns present in the code and the compression of the data through the application of the repetitive nature of the same. In this case, all patterns that occur in data are treated as redundancies, i.e. they are an unnecessary use of space and they contribute to the high size of files.

Hence, like all other redundancies, this needs to be solved. This is done by simply assigning particular characteristic pointers to the patterns occurring in the data and then replacing all instances of the pattern with its respective pointer. Thus, by optimizing the size of the pointers, we can optimize the size of the overall compressed file. In this combined algorithm, first, Huffman Encoding was performed upon the data stream(alphanumeric), making it an efficient data-set i.e. binary output as an input for the de duplication module. The organization of this paper is as follows:

In Section II, the diagrametic representation of Huffman algorithm is illustrated. In Section III, the Huffman compression algorithm is explained stepwise along with its block diagram flow (using sample input). Section III, consists of diagrametic illustration of de-duplication algorithm along with its benefits. Then in Section IV, architecture diagram is covered along with how it works. Section V, gives challenges faced and steps taken to overcome them. Finally, conclusions are given in section VI.

II. ALGORITHM ANALYSIS OF HUFFMAN CODING

The flowchart of the Huffman algorithm is shown below:

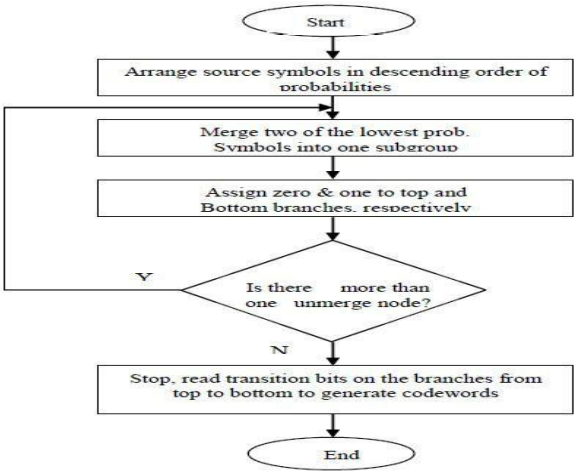

Figure 2.1: The flowchart of the Huffman algorithm is shown below

Main Properties:

1. Codes for more frequent characters are shorter than the codes for less frequent characters
2. Unambiguous

3. Unique prefix property that is no code is the
4. prefix of any other code and all the codes are present at the leaf nodes of the Huffman tree.
5. If prior statistics are available and accurate than this algorithm gives the most satisfactory results.
6. Huffman coding is optimal when probability of each input symbol is the negative power of two.
7. The frequencies are based upon the actual occurrence of the characters in the textual file taken for compression.

III. HUFFMAN COMPRESSION ALGORITHM

There are two major steps involved in Huffman Coding:

1. We have to build a Huffman tree from the given input characters.
2. To assign proper binary codes to each characters the tree is traversed since the unique code of each characters is present at leaf nodes.

Algorithm Analysis of Huffman Coding

The data structure initially used was an array having all the unique characters and their frequencies of occurrences as an input to the algorithm. the output obtained was a Huffman tree with unique binary codes

1. For each unique character or symbol, we created a leaf node and a minimum(min) heap of all leaf nodes (Min Heap is used as a priority queue) was build. in order to compare the two nodes in min heap we used the value in frequency field of an input array. Initially, the Least frequent character is at root.
2. Then the two nodes which has the minimum frequency from the min heap is excluded and kept separately.
3. Then a new internal node that has the frequency equal to the sum of the frequencies of two nodes. The first extracted node in the previous step is taken the left child of the current node and the other extracted node as its right child. this new node with its left child and right child is added to the min heap.
4. The steps#2 and #3 is carried out continuously until and unless one and only one node is left out in the heap. This is the completed tree and the remaining node is considered as the root node.

The following example based on a data source using a set of five different symbols. The symbol's frequenciesare:

TABLE 1:

Symbol	Frequency -
A	25
B	11
C	12
D	9
E	9

----> total 186 bits (with 3 bit per code word)

Huffman Code Tree –

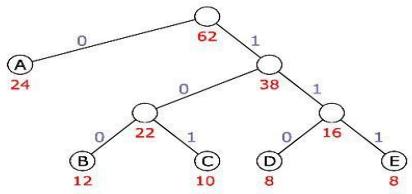

Figure- 3.1

TABLE 2:

Symbol	Frequency	Code	Code Length	total Length
A	25	0	1	24
B	11	101	3	36
C	12	100	3	30
D	9	110	3	24
E	8	111	3	24

---

186 bit tot. 138 bit

(3 bit code)

The difference lies in the code length. Instead of fixed length codes, Huffman algorithm gives varied lengths of codes. Bit reduction is based on the mapping of an alphabet to a modified representation comprised of strings of variable sizes, so that symbols with a higher frequency of occurrence have compact representations than those occurring more frequently. It is a popular variable length encoding method.

Advantages:

- Algorithm is simple and easy to execute.
- Lossless process due to redundancy control.

IV. DATA-DEDUPLICATION ARCHITECTURE

Data deduplication

It is the process of traversing a data set at the sub-file or data components level and storing and/or sending only unique references to data. There are various ways to perform this process but the significant difference distinguishing data deduplication is that data is shared at a sub-file or data components level.

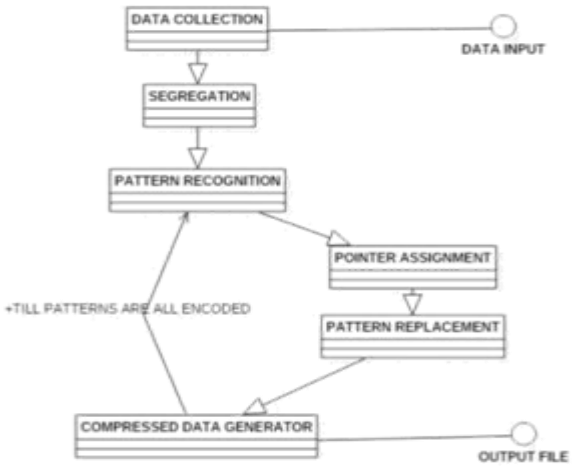

Figure- 4.1: Architecture of De-duplication

Compression

It is the encoding of data such that the overall storage requirement is reduced. Lossless data compression processes ensure the exact original data to be decompressed from the resulting data while lossy data compression methods result in the permanent loss some of the original data components.

Lossless methods

They are used to compress text-based data (such as source code and XML) and where the permanent loss of components cannot be accepted (such as for medical data and HD audio). Some of the file formats that use lossless data compression are ZIP, PNG, GIF and Dolby True HD. Audio and visual data is often stored in formats that implement lossy compression techniques because superior space efficacy can be achieved with insignificant loss in perceivable quality.

Benefits of de-duplication:

- Data deduplication method based on storage reduction optimizes the space requirement for a given bunch of files.
- Its efficiency is more recognized in applications where there is undoubtedly large amount of redundant data, piled up on a single storage entity—a common scenario. In such cases of data restoring mechanisms, that is done on regular basis to prevent permanent data loss, most of the data in a particular backup tends to remain unchanged in their original format from the previous backup.
- General backup systems attempt to make use of this method by omitting those files that has not been changed or by separately storing the significant differences between the data.
- Network data de-duplication aims to reduce the byte count which is to be transferred between network endpoints, this will result in the reduction of the bandwidth requirement and transmission cost.
- Virtual servers and virtual desktops are benefitted from de-duplication technology since it allows for marginally separate files or data for every single virtual machine that is to be combined into a single storage space entity/unit.

V. IMPLEMENTATION OF DATA-DEDUPLICATION (ALGORITHM)

Suppose an organization is running a virtual desktop environment with multitudes of identical workstations all placed in an expensive storage array, purchased specifically to support this structure. So, they're running multiple copies of Windows, Office, their ERP, and any other tools that their users might need. Each individual workstation consumes, suppose 25 GB of disk space. With just a few such workstations, they would come to consume a lot of disk space.

Using deduplication, they can store just one copy of their individual virtual machines and then modify the storage array to assign pointers to its references. Whenever the data deduplication algorithm notices a data pattern which is already stored in the storage area or memory, the system saves a small reference pointer in the place of copy of the data, rather than storing that similar constituents of the redundant data again and again, thus

freeing up the memory that would have been occupied otherwise. In the figure below, note that the images on the left depicts what happens without deduplication. The images on the right depicts deduplication in process. Deduplication enables just one data reference to be written for each component, thus freeing up the rest of the memory resource.

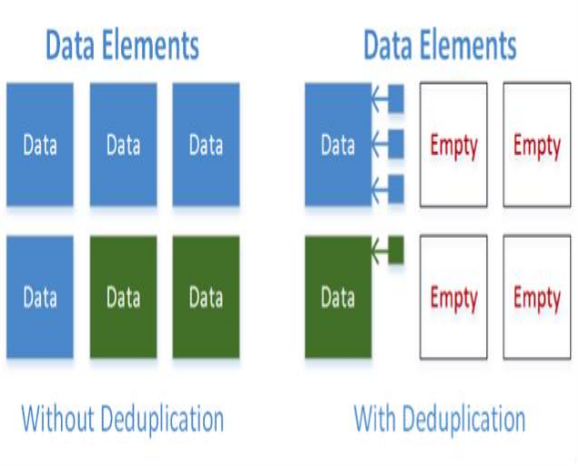

Figure 3: How De-duplication works

De-duplication works by creating a unique data reference, that works as a fingerprint of sorts, for each entity that is written into the storage resource. As new data is entered into the array, if the fingerprints are matching, then any additional data copies are not saved wholly but rather as tiny pointers to the original data. If a new data item is entered completely – one that is not there in the array from before – the data is stored in its fullest form.

VI. CHALLENGES FACED

1. Size & Nature of the Pointers

The size of the pointer is one of the challenging tasks in de-duplication. As the input data might be numeric or alphanumeric, the pointers need to be well established so that the unique pointers don't get mixed up with the rest of the uncompressed data.

2. Sample Space

Given an alphabetical data, the sample space is set at 26, for numeric it is 9 and similarly for alphanumeric it is 35. Hence, the possibilities of finding patterns is less, and the size of the patterns found are smaller. Hence, the efficiency is a challenge.

The Proposed System

1. The system we propose, will perform Huffman Encoding on the sample data and then obtain a stream of binary data.
2. Then we can perform de duplication on the obtained data so that the compression ratio is optimised beyond regular efficiency.
3. If further layers of security are required, then standard algorithms can be used upon the compressed encoded idea.

Received: 22 Sep 2019 | Revised: 13 Oct 2019 | Accepted: 15 Jan 2020

VII. CONCLUSION

This is a combination of two previously defined but now obsolete algorithms of data compression that, when combined and executed over test cases of textual data can compress at massive ratios. The procedure also involves a minimal amount of system resources such as memory and RAM. This feature makes the procedure immensely efficient and beneficial to cases of data transmission where the system resources available are less and/or the bandwidth of the channel is insufficient.

As per the current state of the procedure, the process works best on textual data (alphanumeric + special characters). The process can be further extended to image files, zip files, etc. but only after scraping the file into binary data using a Python Scraper. Another important assumption is that the data that will be subjected to this should not be stored in the Unicode format and just in the raw text format. Hence this procedure will be immensely helpful for lower power systems.

After observing the working of the algorithm on data of regular use, we have come to an average compression ratio of around 60percent over experimental data of 20 test cases involving simple and regular textual statements and strings. Although the ratio may differ from case to case we have enough data to set a worst-case scenario at around 50percent. All of this data excludes the legend that stores the Huffman Codes and De-Duplication References. Combined, the ratio of compression stands to an average of 50 percent and worst case around 45 percent.

REFRENCES

1. Nidhi Dhawale, Implementation of Huffman algorithm and study for optimization, 2014 International Conference on Advances in Communication and Computing Technologies (ICACACT 2014)
2. G.Soma Sekhar1 and E.Sreenivasa Reddy, An Enhanced Data Security with Compression for MANETs, International Journal of Computer Networks and Communications Security, VOL. 2, NO. 12, December 2014, 456–461.
3. T.SubhamastanRao, M.Soujanya, T.Hemalatha, T.Revathi, Simultaneous data compression and encryption, International Journal of Computer Science andInformation Technologies, Volume2(5),2011.
4. AL.Jeeva, V.Palanisamy, K.Kanagaram, Comparative Analysis of Performance Efficiency and Security Measures of Some Encryption Algorithms, International Journal of Engineering Research and Applications, Volume 3,Issue 6, June 2013
5. Mamta Sharma S.L. Bawa, Compression Using Huffman Coding, International Journal of Computer Science and Network Security, VOL.10 No.5, May 2010.
6. Harshraj N. Shinde, Aniruddha S. Raut, Shubham.Vidhale, Rohit V. Sawant, Vijay A. Kotkar, A Review of Various Encryption Techniques, International Journal of Engineering and Computer Science, Volume 3, Issue 9, 2014.
7. http://www.snia.org/sites/default/files/Understanding_Data_Deduplication_Ratios-20080718.pdf
8. http://www.3gendata.com/whitepaper/3gen_datad edup_white_paper_new.pdf