International Journal of Engineering Research & Management Technology
ISSN: 2348-4039
January-2021 Special Issue ICSCTCE-2021
Email: editor@ijermt.org www.ijermt.org

INVESTIGATION AND COMPARISON OF COMPRESSION TECHNIQUE OF GENERALIZED MASSIVE OPTION WITH ASCII

Sushil Kumar [1] **Dr. Anoop Kumar Chaturvedi** [2]
Ph.D. Scholar[1], Professor[2]
Department of Computer Science and Engineering
LNCT University, Bhopal

ABSTRACT:
Compression is an indispensable prerequisite in order to utilize the storage room effectively. In this article, we review that the proposed scheme has greatly increased the performance of compression for experimental outcomes. This paper is primarily intended to reduce memory space and transfer time. In addition, we can produce an experimental outcome that illustrates that our methodology achieves strong compression. In almost all applications, it is very important to reduce the compression ratio of large volume of data set. You will never conceive about database in today's age without a computer programmer. Compression plays an important role any time the database arrives. In virtually any area of compression implementations can be stock market, banking or reservations. With the assistance of compression, very large data can be generated with very little memory. Disk input and output capacity has a very large database bottleneck. The encoding of data bases can be used for reducing the bandwidth needs of disc input for very large data transfers. The writers are researching the compressed archive and suggesting methods for facilitating regular procedures, such as retrieval, inserts, deletion and modifications.

Keyword: Data Compression, Lossy, Lossless, ASCII, Massive, BCD, Decompression, Compression Ratio.

INTRODUCTION:

Data compression is the process of converting an input data stream (the source stream or the original raw data) into another data stream (the output, or the compressed stream) that has a smaller size [20]. A stream is either a file or a buffer in memory.

The research main objective is to simply characterize the compression of data, convert a string of characters into a new string of bits that contain the same content, but the length of which is as limited as possible, in some images like ASCII. In the fields of data transfer and data retrieval, data compression has significant applications. Many data processing systems need massive data volumes to be processed, and as the usage of computers expands to new disciplines, the number of such applications is continuously growing. At the same time, the emergence of computer communication networks has led to significant data transmission through communication links. Storage and/or connectivity expenses are minimized by compressing data to be processed or exchanged. The result is to increase the ability of the transmission medium as the volume of data to be transmitted is limited. Likewise, compressing a file to half of its original size is tantamount to doubling the storage medium's capacity. It can then become possible to store the data at a higher, and quicker, storage hierarchy level and reduce the demand on the computer system input/output channels.

A BRIEF REVIEW OF THE WORK ALREADY DONE IN THE FIELD

A variety of researchers found Huffman's letter frequency-based text compression schemes. Further recent study has considered string matching systems. Detailed surveys of techniques and schemes of compression are performed. Some focused on fast algorithm deployment, parallel algorithms, and VLSI deployment.

Few researchers explored the impact of compression on and performance of database systems. Alsberg suggested saving disc time and space by using minimal bit compression over a set length. He also states that more documents fit onto a page as the records are narrower, which ensures that the records used together are more easily processed. By relating the order of compressed and uncompressed values, Toyama and Ura strengthened the compression scheme in Alsberg. The search and retrieval of compressed data was effective for Egger and Shoshani. Goyal considered the methods of compression which allow compressed data search (selection). In the IMS storage method, Cormack addresses the compression of full documents (not fields).

Other researchers studied encoding and entry systems for science databases with very many constants (for example, zeroes) (matrix transposition). In order to store space and raise the node fan, multiple database schemes are used for hints of prefix and postfix truncation. Hot season. But in the majority of information management systems, real compression techniques are usually not used. In particular, the data compression efficiency effects on commonly used database operations such as relational link, duplicate removal and set intersection have not yet been taken into account.

The Oracle compression algorithm for large data storage tables builds information by extracting double values in a database cube (aka. database page). The algorithm is a method of pernicious dictionary compression. One database block is used to construct the compression window for which a dictionary is generated (symbol table). Compressed data contained in a relational block is thus autonomous. In other words, all the data required to reconstruct the uncompressed data in a block are usable.

NOTE WORTHY CONTRIBUTIONS IN THE FIELD OF PROPOSED WORK

Data compression has been noted to be significant in the field of data processing and data storage. The number of such applications is increasingly growing as computer use is extending to new applications, including the bank, stock exchanges, train bookings etc., where data base sizes are expanding proportionately with the time. The use of the computer is increasing. Therefore, our key incentive is to look for only the techniques or designs that allow us to store this information compactly and to use it for further processing.

PROPOSED METHODOLOGY DURING THE TENURE OF THE RESEARCH WORK

A traditional data database management system can represent real world information using the following types of attribute value or data types.

1. Character (Printable and Non-printable)

2. Number (Integer and Float)

3. Time & Date

These data types are usually viewed in such a way as to ease and efficient processing without seeking how much space they require. You should take advantage of the fact that, in the sense of the database, a specific attribute has a restricted area and type. Our own code is specified, which can handle all the domain values, which should be included in an attribute.

The following attribute forms also endorse a commercial database.

1. Character

2. Memo: Memo (alphanumeric)

3. Date

4. Complete

5. Day: Time

In the proposed design, codes and methods are specified according to certain types of attributes. This programming process can either be added to a new file at the time of design before it is saved or can be applied to a file with certain details.

CHARACTER:
In general 8- bit ASCII code have been used for representing character, but when one declare any attribute to be of character type it often interested only in alphabet character from A-Z or a-z [16]. The front end of an application either data mining or database often equipped with the reporting capability, so without bothering about upper case or lower case letter they can choose any one of them for our system and store the data according to it. Now for 26 character he needs only 5-bits to represent any one of them and the remaining 6 combination of bits could be used to support the character that are required to manage the string or character database like end of line, space, full stop etc.

So it redefined the coding in the following way.

00000-a, 00001-b

00010-c, 00011-d

00100-e and so on up to 11001-z. and the remaining 6 could be used for the following purpose.

11010-space
11011-end of line
11100-comma
11101-full stop
11110-" "
11111-' '

So in this way a compact and complete representation of information is possible.

MEMO:
The memo field often includes character other than alphabets like numbers underscore plus minus etc. and it is found that at most 61 symbol are used in general, so in this situation 6 bits are sufficient to represent them [16].out of these 32 are the same as that in the previous case and remaining one are used for numbers and special character like %,$,#,_.

These are assigned as follows.

000000-a 000001-b
000010-c , 000011-d
000100-e and so on up to 011001 for z .

And the remaining 38 could be used for the following purpose.

011010-space,	011011-end of line
011100-comma,	011101-full stop
011110-" "	011111-' '

100000-0, 100001-1,100010-2,100011-3 and so on up to 101001 - 9 to represent numeral .The remaining combinations are used to define the following symbol

101010-!	101011-@	101100- #	101101- $
101110- ^	101111-&	110000- *	110001-(
		110010-)	110011- -
		110100- _	110101- =
110110 - +	110111-<	111000->	111001-?
111010-/	111011- :	111100-;	111101-|
111110-\	111111- Unused		

Memo field is often used to store information about address, telephone number etc. so if one want to represent sushilkumar24@yahoo.com. It will take about 23 bytes but with this method they just require 18 bytes all together. So with this coding mechanism slight space saving could be achieved.

DATE:

Date field is often associated with temporal database, and most often used inside the data ware house, that contain historical information about any aspect of life.

Most database store the date as 8-byte entry (2 for day, 2 for month and 4 for year). Here it proposes the following format that takes only 2-byte to store any particular date as compared to 8-byte entry.

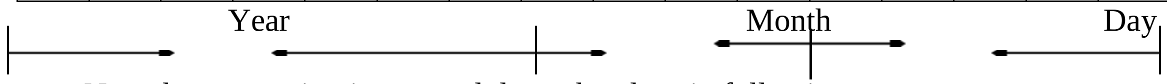
Y	Y	Y	Y	Y	Y	Y	M	M	M	M	D	D	D	D	D

In this method any given date is converted in a 16-bit number using the following formula.

Date = 512*(year-1980)+32*month + day

Let 09/09/2004 is any date, then its corresponding 2-byte no will be 12585, and its corresponding binary equivalent is 0011000100101001. And putting it into our bit format what one get is the binary equivalent of day, month and year. In this field for NULL 0 and for temporal variable UC or NOW 511 could be used.

0	0	1	1	0	0	0	1	0	0	1	0	1	0	0	1



Now the conversion into actual date take place in following manner

Year: - First right shifts the entry by 9 to get the year.

0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0

Its decimal equivalent is 24; by adding this with the base year (1980) they will get 2004.

Month: - First left shifts the entry by 7 position followed by right shift by 12 positions.

1	0	0	1	0	1	0	0	1	0	0	0	0	0	0	0

After left shift by 7 position

0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1

After right shift by 12 positions, its decimal equivalent is 9.

Day: - First left shift by entry by 11 positions followed by right shift of 11 position.

0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0

After left shift by 11 position

0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1

After right shift of 11 positions, its decimal equivalent is 9.

NUMBER:

This type of attribute is also used to describe a certain number, quantity or extent. Depending of the type or precision of the quantification that may be either the integer or float.

The most compact and precise representation of this category of information is by far its own binary counterpart. However, several approaches have been identified that benefit from the design and format of the amount.

DIFFERENTIATION METHOD:

When you compress such data, you can either check for the smallest or largest value for this attribute and save it with the table structure [1]. Now that the initial value is distributed linearly the value could be seen as the difference between the original value and this value which, in turn, is less than the original value.

For example, if you have the following values such as755, 762.792,720,725,789, if you choose 720 as the basic values and save it along with the meaning of the attribute to the table, then the original data is stored as 35, 42, 72, 0, 569. These values will also be used immediately during database processing without further processing if the query has been changed.

BCD STRUCTURE:

For several years, binary encoded decimal code is by far the most common one [16]. During her analysis, one has observed that each single digit is taken to BCD by a decimal number, and that her binary counterpart is entered into the category when four binary digit numbers are converted. But if he used two digits, he will notice an odd thing that would represent the two digits using the discarded bit pattern rather than taking 8 digits to represent them:

Like 410, 4A or (0100 1010) in the BCD or 110011010 in a single binary, which would only require 9 bits instead of 0100 0001 0000.

The number outside the 2-byte integer boundary can be expressed.

For example, 87312 cannot be saved as an integer since its binary counterpart requires a 17-bit representation (10101010100010000) so that if you are using our solution, it can take only 16-bit as follows.

87312□ 873C□1000 0111 0011 1100

7. Time: The usual ways of storing a time stamp of any event require 6-byte and implemented in that manner in most of the database systems. But here again it could save substantial space by representing a time stamp in a manner that require only 2-byte for its storage.

The Bit-wise distribution of Hour, Minutes & Second is shown in following fig.

H	H	H	H	H	M	M	M	M	M	M	S	S	S	S	S

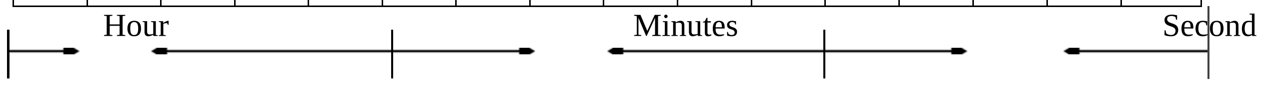
The time can be converted into a 2-byte value using the following formula

Time = 2048*HOUR+32* MINUTE + SEC/2

This scheme is quite common in traditional MS-DOS file system where second value is measured in two-second interval. If one wants to be more precise they have to add one more bit to accommodate second entry because 6 bits are required to represent 60-second domain. They will take the first mentioned scheme to represent time of any event.

Let one have a time value of 16:40:24 in HH: MM: SS format, applying this to formula the time = 34060 & its binary equivalent is:

1	0	0	0	0	1	0	1	0	0	0	0	1	1	0	0



So in this way one can represent this time stamp. Now to get each of these separately they have to perform the bit wise shift operation in the following way. In this field 1952 could be used for temporal variable and 1984 for NULL value.

Hour: Right shift the entry by 11 Bit position.

0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

Its decimal equivalent is 16.

Minutes: Left shift by 5-Bit position followed by right shift of 10 times.

1	0	1	0	0	0	0	1	1	0	0	0	0	0	0	0

After left shift by 5-bit position

0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0

After right shift of 10-bit position, its decimal equivalent is 40.

Second: Left shift by 11-bit position followed by right shift of 11 times & multiply it by 2.

0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0

After left shift by 11-bit position

0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0

After right shift of 11-bit position its decimal equivalent is 12; multiplying it by 2 will get the desired second.

QUERY PROCESSING:

Now with this modified storage structure the operation involve in extracting the data remains the same but with some minor change with each type of attributes

CHARACTER & MEMO:

when there is a query related with these type of attribute the original data inside the query need to be converted into this new modified coding (either in 5-bit or in 6-bit for character or memo respectively). Once it happen the query processing could moves ahead in its normal fashion. The partial ordering that defines the relationship between symbol or operator remains the same as in case of standard encoding, so when conversion takes place the same partial ordering applies again, so that A remains smaller then B and so on. So matching and comparison could be done in the normal way.

DATE & TIME:

when one have a query in which they have to access a an attribute of type date or time, then again the data inside the query need to be converted so that it could become comparable with the stored data, Now with the help conversion formula (for date & time) this could be used for further processing. With these formulas one could rest assure that a small data or time value will always produce the value that will be always less than the value produced for lager date and time.

For example 31/12/1995 =8095 and 01/01/1996 =8225.

So the standard comparison and matching operation could be performed directly.

NUMBER:

when there is a query in which one wants to have access of an attribute of type number, the differentiation method offer several methods over the traditional approach. Now they have a base value for a particular numbered type attribute, so if there is any query that requires the extraction of data on the basis of this attribute value, then he can tell the presence or absence of data without having to look out for the entire data base. In other words if the required data have a value on this attribute that is less than the base value (in case when base value is smallest one for that attribute) or greater than the base value (in case when base value is the largest one) then one can rest assure that the data cannot be stored there with such condition. Beside the query related with the data extraction the query for definition, updation, insertion & modification could be executed in the same way as in the normal case but with some minor modification's as he has done for the data extraction queries. The mapping procedure described for the data retrieval queries can be applied again with these queries as well.

EXPECTED OUTCOME OF THE PROPOSED WORK

It is trying to implement this concept on simple c++ data file with storing data about an employee with his name, his associated department, his salary and his date of join in that department. With that structure I will take field width as required in the standard database file.

Because implementing it on fixed format system that is why it uses this approach otherwise if one has variable length system they could take full advantage of this proposed scheme. Otherwise if they have implemented it on a variable boundary system he could achieve the theoretical benefit of the proposed scheme.

The De-compression will also be take place in reverse way of the compression scheme. In De-compression scheme first they have to read data from decompressed file and convert them back to this compressed structure in the original one.

After going through the suggested implementation (i.e. proposed methodology). If the data stored in such format one can get a compression ratio of around 50% or closer to that value. It gives Substantial improvement over a traditional approach.

REFERENCES

1. S. R. Kodituwakku, U.S. Amarasinghe, "Comparison of Lossless Data Compression Algorithms for Text Data", International Journal of Computer Science and Engineering (IJCSE) Vol. 1 No. 4 416-425, PP. 416-4125, Year 2011.
2. Rexline S. J, Robert L, "Dictionary Based Preprocessing Methods in Text Compression – A Survey", International Journal of Web Based Communities (IJWBC), Vol. 1, No. 2, PP. 13-18, August 2011.
3. N. Magotra, W. McCoy, S. Stearns Dept. of EECE, "A Lossless Data Compression in Real Time F. Livingston.", IEEE, PP. 1058-6393, year 1995.

4. Hengjian Li, Jizihi Wang, Yinglong Wang, Jiashu Zhang,"Effects of Image Lossy Compression on Palmprint Verification Performance", IEEE, Year 2010.
5. Ming-Bo Lin, Member and Yung-Yi Chang, *"*A New Architecture of a Two-Stage Lossless Data Compression and Decompression Algorithm*",* IEEE, Vol 17, No. 9, PP. 1063-8210, September 2009.
6. Md. Nasim Akhtar, Md. Mamunur Rashid, Md. Shafiqul Islam, Mohammod Abul Kashem, Cyrll Y. Kolybanov, "Position Index Preserving Compression for Text Data", Journal of Computer Science and Technology (JCS&T) Vol.11, No. 1, PP. 9 – 14, , April 2011.
7. Haroon Altarawneh, Mohammad Altarawneh, "Data Compression Techniques on Text Files: A Comparison Study", International Journal of Computer Application (IJCA), Vol. 26, No.5, july2011.
8. Tanakorn Wichaiwong, Kitti Koonsanit, Chuleerat Jaruskulchai, "A Simple Approach to Optimized Text Compression's Performance*",* 4th International Conference on Web Services Practices, IEEE, PP. 66 – 70 Year 2008.
9. M. Baritha Begum, Dr. Y. Venkataramani, "An Efficient Text Compression for Massive Volume of Data", International Journal of Computer Application (IJCA), Vol. 21, No. 5, PP. 5 – 9, May 2011.
10. Umesh S. Bhadade, Prof. A. I. Trivedi, "Lossless Text Compression using Dictionaries", International Journal of Computer Application (IJCA), Vol. 13- No. 8, PP. 27-34, January 2011.
11. O. Srinivasa Rao, Dr S. Pallam Setty,"Comparative Study of Arithmetc and Huffman Data Compression Techniques for Koblitz Curve Cryptography", International Journal of Computer Application (IJCA), Vol.14, No.5, January 2011.
12. Yusuf Mohd. Kamir, Mat Deris. Mohd. Snfian, Abidin. Ahmad Faisal Amri, Madi.Elissa Nadia, "Achieving Capability in Data Compression Using LZW++ Technique", DCSMS International Journal of Computer Science and Network Security, Vol.9, No.8, August 2009.
13. R.C. Agarwal, K. Gupta, S. Jain, S. Amalapurapu, "An Approximation to the Greedy Algorithm for Differential Compresssion", IBM J.RES. & DEV. Vol. 50 No. 1, January 2006.
14. Jonathan Gana Kolo, Anandan Shanmugam, David Wee Gin Lim, Li-Minn Ang and Kah Phooi Seng,"An Adaptive Lossless Data Compression Scheme for Wireless Sensors, Vol. 2012, PP. 1-20, Year 2012.
15. Shyue-Kung LU and Ya-Chen Huang, "Improving Reusability of Test Symbols for Test Data Compression", Journal of Information Science and Engineering, Vol. 28, PP. 357-364, Year 2012.
16. En-hui Yang and John C. Kieffer, "On the Performance of Data Compression Algorithms Based Upon String Matching", IEEE Transactions on Information Theory, Vol. 44, No. 1, January 1998.
17. Subhasri S Nayak, S. P. Sahoo, Rishul Matta and Samar Kumar Pattanayak, "A Modified approach to lossless Data Compression", IJREISS, Vol. 2, No. 11, PP. 110-122, November 2012.
18. Thanos Makatos, Yannis Klonatos, Manolis Marazakis, Michail D. Flouris, and Angelos Bilas, "ZBD: Using Transparent Compression at the Block Level to Increase Storage Space Efficiency", IEEE, Year 2010.
19. Ying Li and Khalid Sayood, "Lossless Video Sequence Compression Using Adaptive Prediction", IEEE, PP. 1057-7149, Year 2007.
20. En-hui Yang and John C. Kieffer, Fellow, IEEE, "On the Performance of Data Compression Algorithms Based upon String Matching", ITIT, Vol. 44, No. 1, January 1998.
21. Debra A. Lelewer and Daniel S. Hirschberg, "Data Compression", ACM Computing Surveys, Vol.19, No. 3, September 1987.
22. Rogelio Comp, Hasimoto-Beltran, Sohail A. Sheikh and Ashfa**q** **A.** khokhar**, "**Compression-Efficient Forward Error Control Mechanism for Image Transmission over ATM Networks", IEEE, Year 2000.
23. Yee Wan Wrong, Kah Phooi Seng, and Li-Minn Ang, "Audio-Visual Recognition System in Compression Domain", IEEE, Vol. 21, No.5, May 2011.
24. S.K. Mukhopadhyay, M. Mitra, S. Mitra , "An ECG Data Compression Method via R-Peak Detection and ASCII Character Encoding", ICCCET,18 and 19 March 2011.
25. Cornel Constantinescu, Joseph Glider and David Chambliss, "Mixing Deduplication and Compression on Active Data Sets", Data Compression Conference, Year 2011.
26. Yangjun Chen,Yibin Chen, "Decomposing DAGs into Spanning Trees: A New Way to Compress Transitive Closures", IEEE, Year 2011.
27. Thanos Makatos, Yannis Klonatos, Manolis Marazakis, Michail D. Floris and Angelos bilas, "ZBD: Using Transparent Compression at the Level to Increase Storage Spacev Efficiency", International Workshop on Storage Network Architecture and Parallel I/Os, Year 2010.
28. S. A. Ahsan Rajon and Md. Tafiqul Islam, "An Effective Approach for Compression of Bengali Text", IJCIT, Vol. 01, No. 02, PP. 30-36, Year 2011.
29. Sangjoon Lee, Jungkuk Kim, Myoungho Lee, member IEEE, "A Real-time ECG Data Compression and Transmission Algorithm for an e-Health Device", IEEE, PP. 1-8, Year 2011.

30. James E. Fowler, Kenneth C. Adkins, Steven B. Bibyk, Stanley C. Ahalt, Member IEEE, "Real-Time Video Compression Using Differential Vector Quantization", IEEE Transactions on Circuits and Systems for Video Technology, Vol. 5, No. 1, February 1995.
31. Jorge Bernardino and Henrique Madeira, "Experimental Evaluation of a New Distributed Partitioning Technique for Data Warehouses", IEEE, PP. 312-321, Year 2001.