Journal of Theoretical and Applied Information Technology
31st July 2023. Vol.101. No 14

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

A PARALLEL SPARSE DATA COMPRESSION (PSDC) METHOD FOR FILE STORAGE OPTIMIZATION USING CLOUD ENVIRONMENT

N.SRIKANTH, T.PREM JACOB [2]

[1] Research Scholar, Sathyabama Institute of Science and Technology, Department of CSE, Chennai, India

[2] Professor, Sathyabama Institute of Science and Technology, Department of CSE, Chennai, India

[1]srinekkalapu51@gmail.com, [2]premjac@yahoo.com

ABSTRACT

Huge, distributed data-intensive applications consume data at fast speeds, causing adverse I/O effect on storage services while managing data. Cloud computing is the future of data storage, communication, and resource sharing. Cloud computing makes data-intensive applications appealing to a wider public who cannot afford pricey large-scale distributed infrastructures. Compressing large amounts of intense application data for internet transmission saves time and storage capacity. This study proposes Sparse Data Compression Algorithm (SDC) to reduce memory entries and reduce storage space. Conventional compression technology cannot process big data with a high compression rate and low energy cost. Using the Cloud framework, this study implements Parallel Sparse Data Compression (PSDC), a high-speed lossless data compression algorithm. The PSDC algorithm chunks data compression data and submits to cloud virtual cores to reduce computational complexity. PSDC in the cloud environment speeds up compression and improves ratio. Eventually, the proposed method outperforms standard compression algorithms in compression ratio, compression size, and compression time.

Keywords: *Data-intensive applications, Cloud Computing, Data Compression, Parallel Computing, and Sparse Representation.*

1. INTRODUCTION

"Big Data" has emerged as a buzzword in both the business world and the academic world due to the proliferation of many large-scale applications that are data intensive. Not only do these data-rich applications have a very big data volume, but they also potentially have complicated data structures and high update rates. Existing data management systems have major lapses in performance, scalability, and programmability. Users that require processing massive volumes of data but lack the financial resources to maintain their own large-scale infrastructure can rent the resources they need to operate their applications by paying only for the resources used throughout the course of the application's execution. Users who lease these resources can lease other key resources for running applications. Due to the fact that the expanding cloud computing model offers a new framework for operating computer resources, it is attracting significant interest from both industry and academics. Users rent virtual computers and storage space from the cloud provider rather than just buying and operating hardware [1].

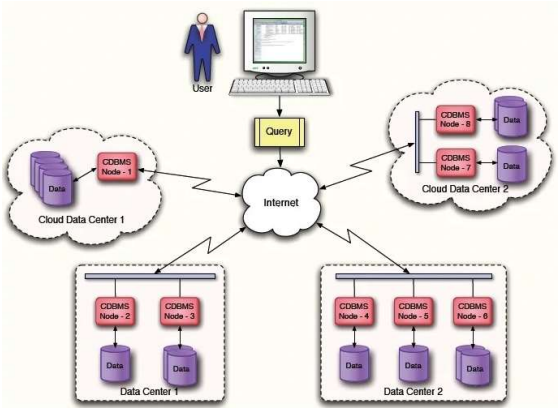

Figure.1. Cloud Model for Data Management

Data compression techniques serve a variety of purposes, including lowering disc space requirements, communication, or data transmission times, and more. The quantity of information that must be managed by software and exchanged between systems has been expanding along with the quantity of data that must be processed. Data compression technology is seen as a crucial component in these situations to maintain the

Journal of Theoretical and Applied Information Technology
31st July 2023. Vol.101. No 14

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

information infrastructure. Although there are many other devices besides PCs that use data compression technology, such as modems, routers, digital cameras, facsimiles, CDs, MDs (Mini discs), video on demand (VOD), TV conference systems, DVDs, digital telephones, and other devices, the term "data compression" is typically used to describe information that's kept in files and transferred over phone lines. To compress anything means to express it in a form that takes up less space than the original, uncompressed version [2]. There are two distinct types of compression methods in terms of being able to recover the original data. The two kinds of compression are referred to as lossless and lossy, respectively. The decompressed data exactly mimics the original when employing a lossless approach. cloud data compression, in essence, entails discussing a stream of symbols and turning it into codes [3]. the final stream of codes will be less than the original symbols if the compression was successful. a model is used to determine the code to output for a given symbol or group of symbols. modeling and coding are the two parts that constitute data compression. cloud computing is the newest big thing, as per advanced developments[4]

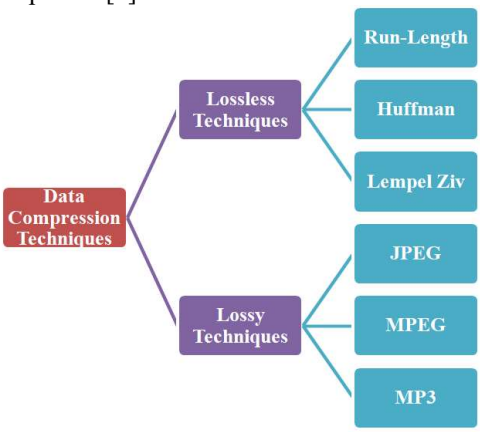

Figure.2. Data Compression Techniques Classification

Limpel and Ziv (L Z) codes [5], Huffman codes [6], Adaptive Huffman [7], Run-length encoding [7], Arithmetic coding [8], Multigroup compression method [8], Shannon-Fano code [9], and dictionary-based approaches [9] are just few of the lossless compression algorithms that have been suggested in the past. If you choose a lossy compression method, the decompressed data won't be an exact replica of the original. Huffman coding and arithmetic coding both suffer from the need for a double scan, the first to determine probability and the second to encode the text. Static or dynamic dictionary encoding is possible. It's more troublesome to use static dictionary coding since the dictionary must be transmitted with the compressed data to the receiver. LZ77 [5], LZ78 [5], and LZW [5] are examples of adaptive dictionary data compression methods. All of the aforementioned algorithms, however, are sequential, with the result that the i-th block can only be programmed once the i-1-th block has finished. Lossless methods can be parallelized to bypass this restriction (i.e., Parallel Sparse Data Compression). This paper's primary contribution is a very efficient compression method that may be used by cloud-based applications to improve their download speeds and reduce their space requirements.

The rest of the present paper is structured as described below. In Section 2, we will talk about the research that has been done on compression strategies. In Section 3, we describe the various known compression strategies, which serve as a foundation for the work that we present in this study. Following that, Section 4 comprehensively analyzes the proposed compression algorithms. The findings are discussed in detail in Section 5. The conversation is brought to a close with the presentation of Section 6.

2. RELATED WORK

Compression is one of the effective methods that can be used to enhance the storage of data. A number of different methods have been investigated for effective storage, and this section describes them in depth. This paper, which introduced a method that is known as CULZSS[10], LZSS lossless data compression on CUDA, presents how LZSS lossless data compression algorithm is applied on CUDA, before analyzing its performance using 5 different datasets. [11] Compressing files with the LZO technique on NVIDIA CUDA architecture is a topic of discussion here, as are the various ways in which LZO could be implemented on GPUs using the Fermi architecture. For compressing an unstructured data stream with double precision, FPC makes use of a pair of value predictors that are based on hash tables. It provides an adjustable parameter, in which compression efficiency is traded for processing speed. The chunked processing of incoming data that is enabled by the thread-parallel variation of the pFPC algorithm [12] enables higher prioritization of compression throughput. SPDP [13] links one-dimensional prediction to a variation of LZ77 for compressing data without specialization for either the single- or double-precision formats. This allows the data to be compressed more efficiently. MPC is a quick compression method for GPUs that was developed by Microsoft. A new algorithm that maps very well to the hardware that is being targeted is

created by combining a straightforward one-dimensional value predictor to a bit-regrouping strategy for zero-bit elimination in the remaining results. For delinking data points in an n-dimensional grid and the neighbours that have previously been processed, the APE and ACE compressors make an intelligent selection from a pool of various value predictors. A modification of the Golomb coding algorithm is used to condense the residuals demonstrate two distinct parallel implementations of lossless data compression algorithms based on BWT, one of which is data parallel, and the other of which is task parallel. To compress parallel data, total data file gets broken as minute parts with every cluster undergoing unique compression before being written to the disc in the correct order. The compression algorithm responds variedly to several threads or processes so that task parallel compression is properly implemented. The results of the experiments reveal that the parallel data technique offers a significant increase in speed. [14] demonstrate a parallel implementation of a lossless data compression algorithm similar to bzip2 that runs on the GPU. They do this by parallelizing Burrows-Wheeler transform (BWT), Move-to-front transform (MTF), and the Huffman coding phases of compression pipeline. According to the findings of the experiments, their implementation is noticeably more time-consuming than bzip2's.

3. COMPRESSION METHODS

Data compression can be understood as the process of taking a stream of data in X and converting it into data that takes up less space. If the compression was successful, the resulting stream of data, denoted by the letter D, will be less extensive than the initial data, denoted by the letter X. A model serves as the foundation for making the decision to produce an output of specific compact data that combines data rules that are applied in processing input symbols. These rules indicate which code is going to be modified. For the purpose of the efficient optimization of the data, the number of approaches is provided[15].

3.1.Huffman encoding

The Huffman encoding [7] system works like this: the symbol that occurs with a higher frequency is given a shorter code, while the symbol that occurs with a lower frequency is given a longer code. Longer codes are built in such a way that they cannot be used as prefixes by lesser codes. This is an important aspect of the construction of longer codes. This piece of coding creates what is known as a Huffman tree, just a binary tree. In a binary tree, the left branch stands for the number zero, and the right branch stands for the number one.

3.2.Arithmetic Encoding

Similar to how the Huffman encoding works, this method uses a varying number of bits for each symbol. Contrary, Arithmetic coding follows a different technique []. The output is saved as a single floating-point value that is higher than or equal to 0 but is less than 1. This method is far more adaptable than the Huffman encoding.

3. 3. LZ77

Dictionary work is performed in LZ77 method as component of sequences that were encoded earlier. The encoder performs an analysis on the input sequence by activating the sliding window, which is made up of two distinct components known as Search and Look-ahead buffer that hold sections of the latest encoded sequences. However, Look-ahead buffer holds sequence parts for encoding later. LZ77 does not have its own external dictionary, which can create issues when decompressing files on a different system. In this particular approach, whenever there is no match of any strings, that string is encoded as a length and offset, which will take up additional space. This extra step also increases the amount of time that the algorithm needs to complete its task[10].

3. 4. LZ78

An explicit dictionary is kept up to date with dictionary-based compression method like LZ78 compression technique. The encoded output is comprised of two components: an index that points to the dictionary item with the longest matching string, as well as the first symbol that does not match. In addition to this, the index and symbol pair are added to the dictionary by the algorithm. Before a symbol shows up in a dictionary, the codeword receives index value 0, which is included in dictionary as well. This occurs when the symbol is not yet known. The algorithm builds the dictionary with this approach. LZ78, much like LZ77, has a compression rate that is low but a decompression speed that is high. Although LZ78 is faster than LZ77, it may not always produce a compression ratio that is as high as that of LZ77. The greatest benefit that the LZ78 algorithm has over the LZ77 method is that it performs fewer string comparisons throughout each phase of the encoding process [10].

3. 5. LZW

The sequential LZW algorithm begins by initializing the dictionary to contain all strings of length one. This is done so that the dictionary is ready for use. After that, it goes through the dictionary and finds the longest string that corresponds to the current

Journal of Theoretical and Applied Information Technology
31st July 2023. Vol.101. No 14

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

input. It then generates the code for that string. The matched string, along with the character that comes after it, is added to the dictionary by this function. This process will continue until all of the characters in the input have been dealt with[16].

Algorithm 1 : LZW(S,N)
Input :S =A File contain a String
Output : C_S = The compressed File contain string
Begin :
N = string_length(S);
D = Dictionary;
k = 0;
LW[k] = f ;
while(i <= N)
{
j = availble(D, S[i - j]);
append(D, S[i - j]);
LW[k] = retrieve(D, S[i - j -1];
k ++;
i = j;
}
end

Figure.3. LZW Algorithm

3.6. PLZW

If we had N processors, the simplest technique to parallelize LZW would be to partition the data into N chunks and assign a CPU to compress each chunk in turn (PLZW). As a result, the amount of time spent computing will be cut down due to the fact that each processor will operate on a chunk independently. This method does have some drawbacks, one of which is that it can lead to an increase in the number of redundant entries in the dictionaries of the processors (due to the fact that the identical items might be placed in many dictionaries), which will result in a lower compression ratio[19]

4. PROPOSED METHOD: SPARSE DATA COMPRESSION ALGORITHM (SDCA)

Data compression offers a lot of promise in the form of huge storage capacity as well compatibility with varied network's bandwidth. These days, data compression has gained much traction as it has the inherent capacity to reduce bandwidth and storage need, entailing encoding lesser bits, apart from consuming less time for transmission.

Algorithm 2 : Parllel_LZW(S,N)
Input : S = A File contain a string
Output : C_S = The compressed File contain string
Begin
p = no.of.processors
b = no.of.blocks of S
while(i <= p)
{
LZW(b[i])
i++
}
return b
end

Figure.4.Parallelize LZW Algorithm

In the current research, a unique model consisting of compression algorithm with cryptographic method is proposed. By examining existing compression techniques, including arithmetic coding, Huffman coding, LZ78, and LZW, a new approach called the sparse data compression algorithm is proposed (SDCA). By removing memory entries with values of "0," it gives the substantial benefit of conserving more store space by preventing the waste of storage places. Thus, like a special provision, "0" gets saved if "1" goes beyond a given limit. The basic principle behind this algorithm is based on the sparse matrix representation. It is illustrated in the following matrix shown in eq(1).

$$\mathbf{SD}_c \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \quad (1)$$

In general, the storage is required 3X3 =9 for accommodate given values on server storage .The entries for the position (1,1),(1,2),(1,3), (2,2),(3,1),(3,2),(3,3) contain the value "0" . If the actual storage location contains the value "0" repeatedly, the storage locations become waste for many applications resides on cloud servers. The sparse data compression algorithm working principle has two major components such as Data (the original data bit), header (total number of required storage) and link (if the continuation of data bit). It stores the position bits having the values 1 except all zero entries, in the given input representations. The detailed explanation for the sparse data compression algorithm is described in the Algorithm.

Journal of Theoretical and Applied Information Technology
31st July 2023. Vol.101. No 14

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Algorithm : Sparse Data Compression Algorithm (SDCA)

Input : S = A file contain a data

Output : C_S = A Compressed File

Begin

Divide the entire data into n blocks $S_i = \{S_1, S_2, S_3, ..., S_n\}$ where each $S_i = 6$ character length

{

If block size=6 then

{

Rotate the S_i string and sort in lexicographic order (i.e M_i is a Matrix)

}

else

{

Add a special character to M_i sort in lexicographic order to make the block of sizes six and rotate the string and sort in lexicographic order

}

Calculate $< L , x >$ where L is the last column of the matrix M_i row number in the matrix M_i and x is the correct string row number in the matrix M_i.

Convert L into Binary data using Hash tables (Given) and store number of bits in K.

if $K \% 8 = 0$

{

Add m bits to K bits by considering the following condition:
If last bit in L is 1 then 0 bits are added or If last bit in L is 0 then l are added until it satisfies the condition $(K+m) \% 8 = 0$.

}

else

{

Store L in Sparse Matrix /* only sets of 1's stored and x value for decompression*/

}

4.1. Data sharing in cloud using SDCA and encryption

A scalable, dependable, and quickly developing technology is cloud computing that provides a variety of pay-per-use services to its subscribers. Such services consist of compute, storage, and many applications, among other things. The emphasis in this case is on cloud computing storage security. Data that has been encrypted gets loaded onto cloud storage services, allowing key to be safely stored on local server to do further decryption. However, this study focused on the efficient compression algorithm for optimizing the file storage prior to the encryption. Figure 5, describe about data sharing in cloud in different stages, initially data is checked, and it converted into binary form. Later applied SDCA, for efficient compression to the data. At last stage compressed data is encrypted and uploaded into the cloud storage

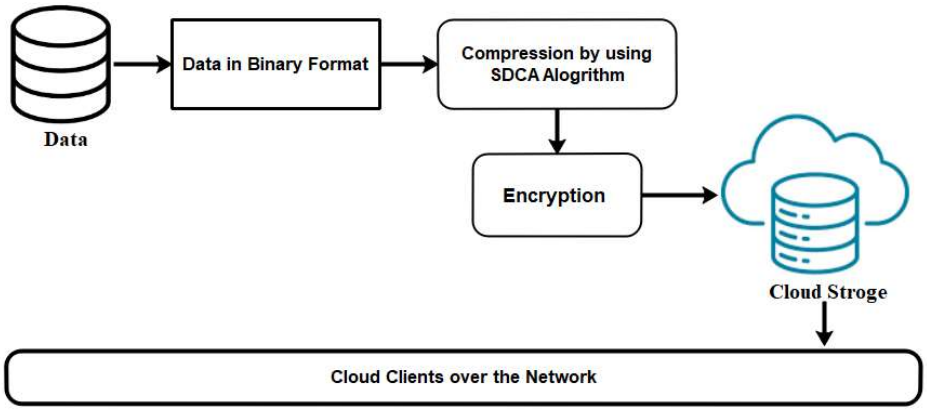

Figure 5. The Secure Data Sharing Over Cloud Using Compression and Encryption

4.2. Parallelized Sparse Data Compression (PSDC) Algorithm

While performing repetitive calculations on large amounts of data, parallel processing offers significantly quicker options over sequential processing due the capacity of parallel processors to handle many data streams at once on its inherent multithreading prowess. The parallelized mechanism of the Sparse Data Compression algorithm where the divided blocks are distributed onto different cores, on each core the SDCA method will be executed individually, and final results are collected to form compressed data. The detailed algorithm is given below. The Figure. states the parallelized mechanism of the Sparse Data Compression algorithm, wherein the divided blocks are distributed into different cores, while on each core, the SDCA method is executed individually, and the final results are collected to form a compressed data. The detailed algorithm is given below:

5. EXPERIMENTAL RESULTS

5.1. Performance measure:

The performance metrics often establish whether a compression approach is effective or not, besides its compatibility with specific criterion. The data compression ratio is defined in the same way as the physical compression ratio, and it is utilized to assess how much a substance has been physically compressed much like the ratio between compressed size and uncompressed size.

$$\text{Compression\_Ration}(C_R) = \frac{\text{Size after compression(D)}}{\text{Size before compression(X)}}$$

The ratio between source file size and compressed file size is known as compression factor, just opposite of compression ratio:

$$\text{Compression\_Factor}(C_F) = \frac{\text{Size before compression(X)}}{\text{Size after compression(D)}}$$

Saving Percentage: The shrinking of the source file calculated as a percentage.

$$\text{Saving\_Percentage}(S_P) = \frac{\text{Size before compression(X)} - \text{Size after compression(D)}}{\text{Size before compression(X)}}$$

The performance and efficiency of compression are computed using the three criteria mentioned. The performance of the compression algorithm improves with a decreased compression ratio. Also, a high compression factor is necessary for improved compression techniques. Here, the saved percentage provides us the clue about the actual percentage of compression attained.

Journal of Theoretical and Applied Information Technology
31st July 2023. Vol.101. No 14

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Parallelized Sparse Data Compression (PSDC) Algorithm

Input : S = A file contain a data

Output : C_S = A Compressed File

Divide the entire data into n blocks $S_i = \{S_1, S_2, S_3, ..., S_n\}$ where each $S_i = 6$ character length

Initialize the Cores/Workers as $C_i = \{C_1, C_2, C_3, ..., C_n\}$;

Assign each block S_i;

$S_i = M_i$;

if block size=6 then

{

Rotate the S_i string and sort in lexicographic order (i.e M_i is a Matrix)

Add a special character to M_i sort in lexicographic order

}

Calculate $< L, x >$ where L is the last column of the matrix M_i row number in the matrix

M_i and x is the correct string row number in the matrix M_i.

Convert L into Binary data using Hash tables (Given) and store number of bits in K.

if $K \% 8 = 0$

{

Add m bits to K bits by considering the following condition:

If last bit in L is 1 then 0 bits are added or If last bit in L is 0 then l are added

until it satisfies the condition $(K+m) \% 8 = 0$.

}

else

{

Store L in Sparse Matrix /* only sets of 1's stored and x value for decompression*/

}

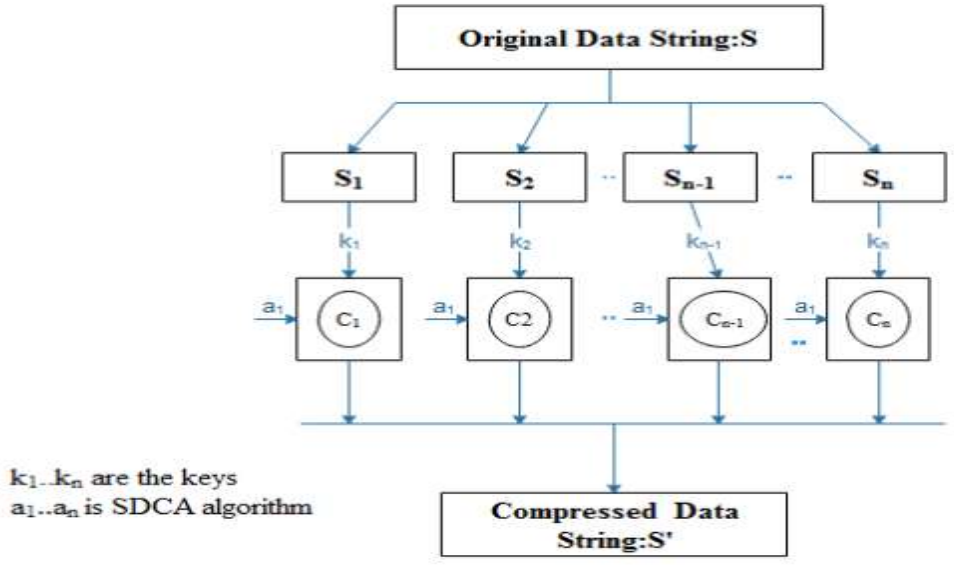

Figure.6. Parallelized Sparse Data Compression (PSDC) Algorithm

5.2. Results of SDCA

The possibility for reducing access times between cloud clients and cloud servers, as well as the possibility for using less cloud storage due to "NULL" value waste, are two significant benefits of the designed system. The effectiveness of cloud storage, the quality of the data that can be retrieved, and the speed of execution are all taken into consideration in the relative efficiency study. The comparison of several compression methods, including Arithmetic Coding, Huffman Coding, LZ78, LZW, and SDCA, is shown in the following graph.

Table.1. Performance Comparisons with SDCA

Input Characters	Storage Efficiency (KB)	Retrieval Efficiency	Time of Execution Efficiency(Sec)
Arithmetic Coding	13.7	83	16.05
Huffman Coding	17.3	73	11.05
LZ78Coding	13.5	65	10.08
LZW Coding	18.2	70	13.01
SDCA	11.4	87	8.82

Table 1 illustrates comparison between original size and post-compression text file sizes. It further makes a compares the compression ratio of the existing algorithms vis-a-vis suggested algorithm. For the purpose of analysis, we have included Arithmetic Coding, Huffman Encoding LZ78 Coding and LZW encoding Algorithms. The overall observed the storage efficiency is less i.e., 11.4(KB) in proposed method compared to all the standard methods. Also, it is shown results from the table 1, the Retrieval Efficiency is also improved and produces least time for execution i.e, 8.82 in SDCA. Figure 7 illustrates the compression ratio between proposed method and standard methods, which is obtained from compressed file: original file sizes as depicted in the plot. A brief analysis of the result yielded by the SDCA demonstrates the possibility of achieving constant compression ratio, notwithstanding file sizes or their contents.

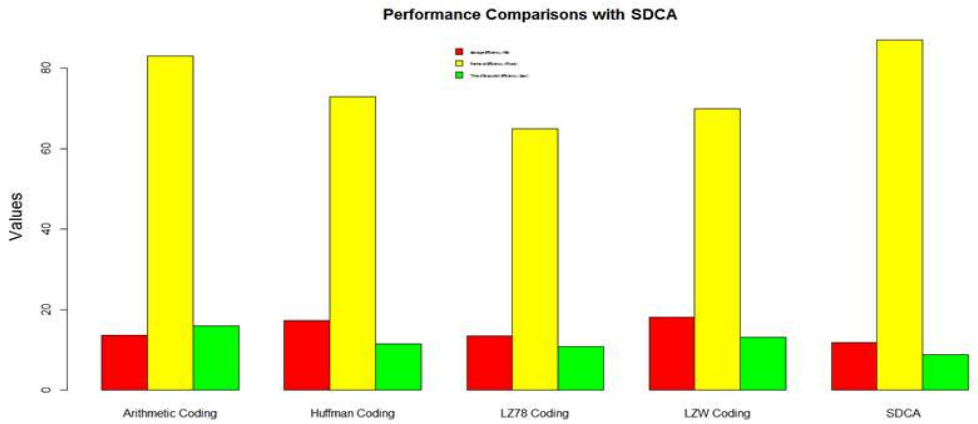

Figure.7.Performance of Compression Methods with Compression Ratio

Table 2. File Size: Original vs. Compressed

File Name	Original File Size (in KB)	Compressed Size using SDCA (in KB)	Compressed Size using WinRAR (in KB)	Compressed Size Using Arithmetic Compression (inKB)
File1	828	158	223	315
File2	1091	212	265	387

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

File3	978	176	254	371
File4	867	165	234	321

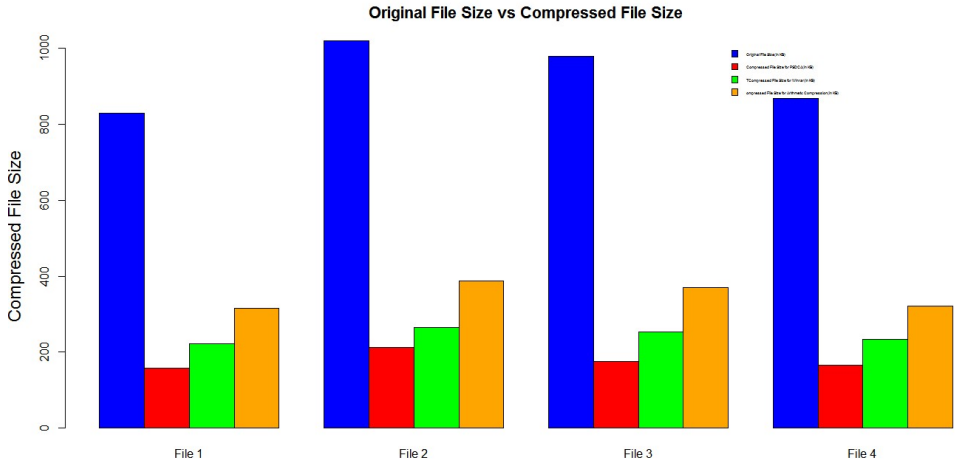

Figure.8. File Size: Original Vs. Compressed & Performance of Various Compression Methods

Figure.8 illustrates the plot depicting original file size to compressed file size ratio. Results yielded by the SDCA demonstrates the possibility of achieving compression file with minimum size in all the four different files. File 1 produces least size 158 whereas its original size are 828 and reduced to 19%. However, in other mechanisms produced higher sizes with proportion of 26% and 39%.

5.2. Results of PSDCA

This work is implemented on the latest version Hadoop's 2.7.1on Ubuntu14.10. In all, many nodes are included in the test cluster wherein one is master, while out of 10 data nodes, 7 are similar and 3 disparate, creating 4 mixed systems. Table.3 shows the configuration of the cluster used in this study.

Table.3.Configuration of the Cluster

Type (Number)	Processor Model	Internal Memory(RAM)	External Memory(RAM)
A(7)	2.5GHzInteli5Processor	4GB	500GB
B (1)	3.2GHzInteli5Processor	16GB	500GB
C(1)	3.6GHzInteli5Processor	8GB	320GB
D(1)	3.6GHzInteli5Processor	32GB	1TB

Table. 4. Comparisons of Compression Performance

File Name	*File Size*	*File Type*	*Parallel Arithmetic Coding*	*Parallel Huffman Coding*	*Parallel LZW Coding*	*Purposed PSDAC*
ERR009295	470811430	PDF	217694610	217694610	224274069	199908837
ERR092485	611354745	PNG	585467042	585467042	588234117	585467042
SRR628456	1075151912	BMP	633317808	633317843	692954495	633317776
SRR624457	603227492	MP3	153113767	156648643	157349618	133078188
SRR624558	217514321	WAV	23062960	26059923	20853461	19259273
SRR624659	1608147955	DLL	774870508	766382271	686669775	679072189
SRR624850	4985746	C++	1052673	1216547	1100900	1014600

Journal of Theoretical and Applied Information Technology
31st July 2023. Vol.101. No 14

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

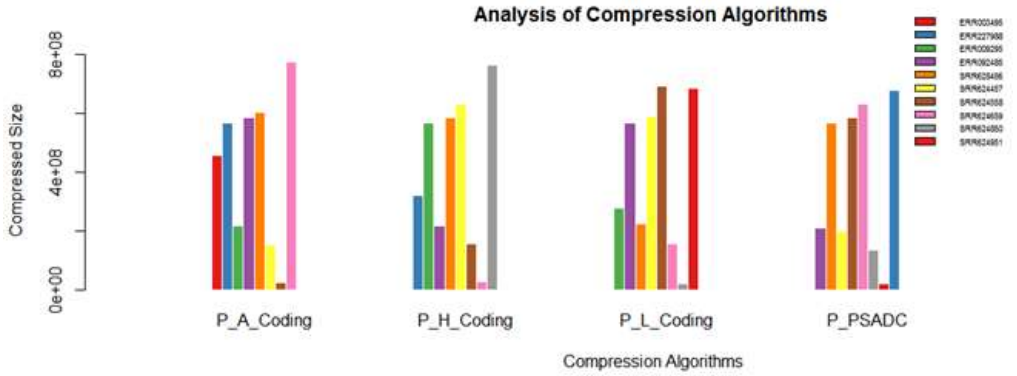

Figure.9.Comparisons of Compression Performance

Table 5 Comparisons of Compression Ratio of Parallel Compression Algorithm

Original Data			*Compressed Ratio*			
File Name	*File Size*	*File Type*	*Parallel Arithmetic Coding*	*Parallel Huffman Coding*	*Parallel LZW Coding*	*Purposed PSDAC*
ERR003495	1899995323	PDF	100.547	78.763	74.766	68.786
ERR227988	569784647	PNG	98.748	72.594	68.574	62.676
ERR009295	470811430	BMP	88.675	69.134	66.432	60.458
ERR092485	611354745	MP3	88.964	69.264	62.273	59.278
SRR628456	1075151912	WAV	100.333	70.382	68.487	65.487
SRR624457	603227492	DLL	96.045	67.324	62.345	58.367
SRR624558	217514321	C++	99.456	67.345	61.246	57.548
SRR624659	1608147955	Stream	99.849	74.843	72.873	62.875
SRR624850	4985746	TXT	99.467	63.145	60.239	57.235
SRR624951	5045745	TXT	99.873	63.348	60.046	57.143

The proposed work PSDAC is tested with various algorithms like Parallel Arithmetic Coding Parallel Huffman Coding, Parallel LZW Coding and the results is shown in Table 4 and Figure 9. A total of 8 files of different types include BMP, MP3, WAV, DLL,C++,Stream, and TXT and each of different sizes is tested with all parallel compression algorithms. It is observed from the results that overall compression performance of all parallel algorithms proposed produces better results compared to others. In terms of the file size reduction proposed produced better in a way, in case of the file name SRR624659, the original size is reduced from 470811430 to 199908837. However, the other methods reduced to file sizes 217694610,217694610, and 224274069 respectively. These file sizes are high compared to the file size produced by the proposed method PSADC. Similar kind of performance is observed in the other files also.

Journal of Theoretical and Applied Information Technology
31st July 2023. Vol.101. No 14

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

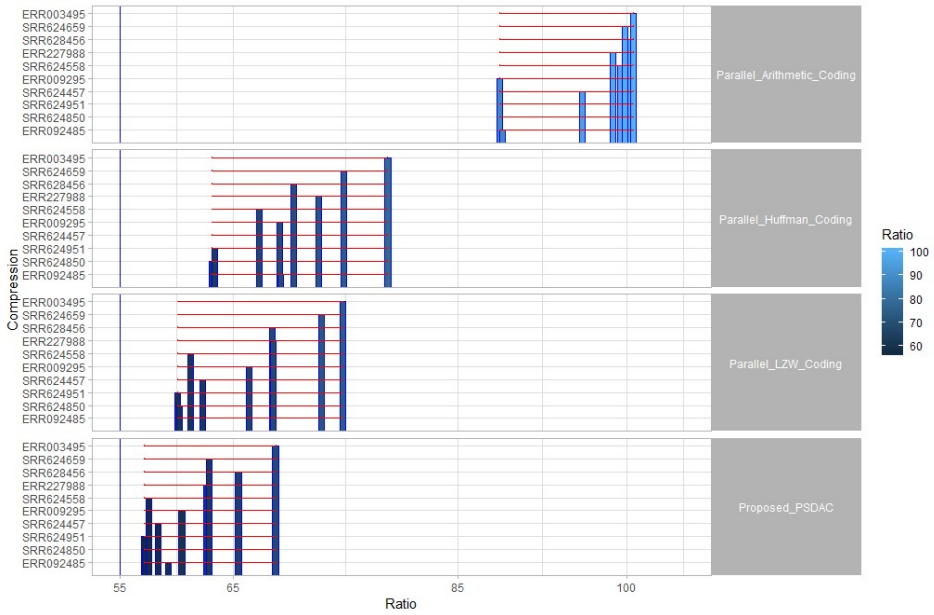

Figure.10. Comparisons of Compression Ratio of Parallel Compression Algorithms

Figure 10 and Table 5, illustrates the compression ratio against among parallel proposed method PSDCA to standard parallel methods Parallel Arithmetic Coding, Parallel Huffman Coding, Parallel LZW Coding respectively. Figure 10 depicts compressed file size: original file size ratio, which is compression ratio. A brief analysis of the result yielded by the PSDCA demonstrates the possibility of achieving better compression ratio in all respective files with different types. The lowest compression ratio 57.143 is produced in the file name SRR624951 of type TXT and in standard parallel methods its ratio to be 99.873,63.348,60.046 respectively. The highest compression ratio 68.786 is produced in the file name ERR003495 of type PDF and in standard parallel methods it ration to be 100.547, 78.763,74.766 respectively.

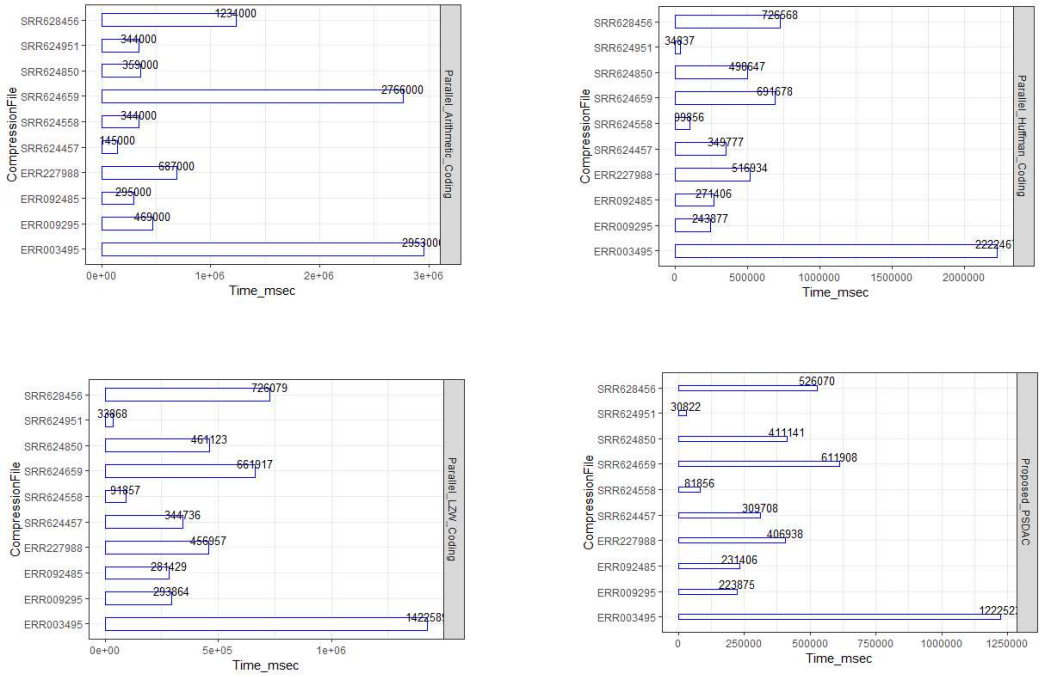

Figure.11. Comparisons of Compression Time of Parallel Compression Algorithms

Journal of Theoretical and Applied Information Technology
31st July 2023. Vol.101. No 14

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Figure.11illustrates the compression time against among parallel proposed method PSDCA to standard parallel methods Parallel Arithmetic Coding, Parallel Huffman Coding ,Parallel LZW Coding respectively . A brief analysis of the result yielded by the PSDCA demonstrates the possibility of achieving least compression time in all respective files with different types. In general, the minimum compression time plays a major role for large data compression methods. Amongst the known parallel compression tools, not too many algorithms can be applied to huge data sizes. However, only PSDCA offers matching performance in terms of compression time speed, variable throughput, and compression ratios.

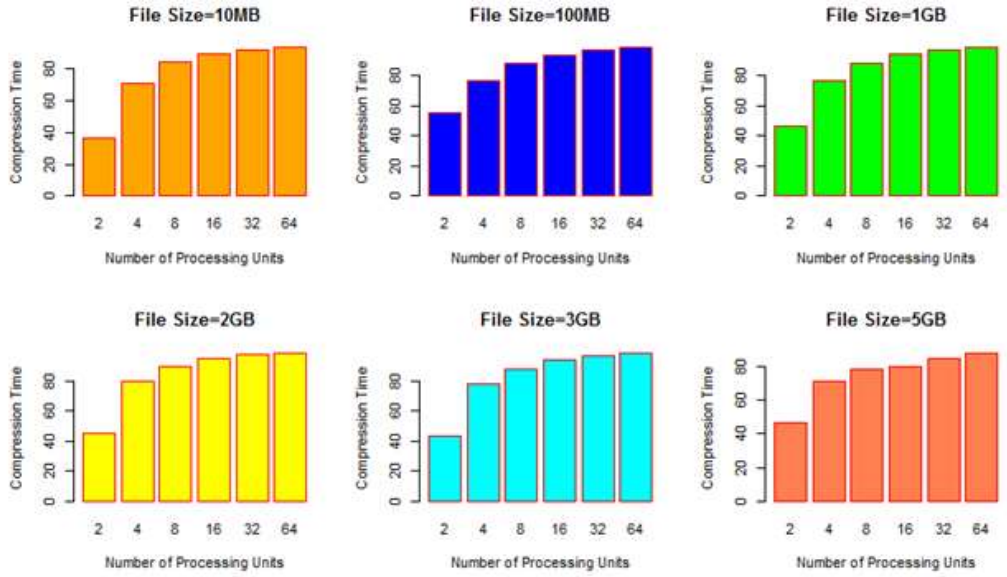

Figure.12. Compression of algorithm performance: Compression Time & File size and No.of Cores

Table 6. Ccompression Of Algorithm Performance: Compression Time & File Size And No.of Cores

Name	Size	Processing Units					
		2	4	8	16	32	64
File1	10MB	36.43	71.37	84.73	89.89	92.49	93.47
File2	100MB	55.34	76.83	88.24	93.44	96.56	98.23
File3	1GB	46.20	76.75	88.23	94.22	97.12	98.46
File4	2GB	45.46	79.83	89.65	94.59	97.39	98.56
File5	3GB	43.67	77.84	87.88	94.45	97.09	98.67
File6	5GB	46.79	70.98	78.57	79.65	84.63	87.77

In Table 6 and Fig. 12, we exhibit reduction in compression time percentage as generated by PSDCA, with nearly 90% reduction in compression time while compressing 10MB,100MB,1GB,2GB,3GB, and5GB data through 64 processes. Furthermore, compression time was reduced by nearly half while compressing 100MB data to 5GB on 2 processes. It is observed that percentage of compression time was reduced with changes quantity of processing units, approximately1/p where p is no. of processors.

6. Conclusion

Cloud computing is the future of data storage, communication, and resource sharing. Cloud computing makes applications efficient in data handling functions, quite affordable to many who cannot afford costly large-scale distributed infrastructure. Compressing large amounts of intense application data for internet transmission saves time and disc space. This study proposed Sparse Data Compression Algorithm (SDCA) and Parallel SDCA, which eliminate memory entries to conserve storage space. The suggested architecture can reduce cloud client access time via cloud servers and cloud storage wastage by using "NULL" values. Due to poor communication channels, matching missing storage values in their production portion may take longer than expected.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

REFERENCES:

[1] J. Shen, T. Zhou, D. He, Y. Zhang, X. Sun and Y. Xiang, "Block Design-Based Key Agreement for Group Data Sharing in Cloud Computing," in IEEE Transactions on Dependable and Secure Computing, vol. 16, no. 6, pp. 996-1010, 1 Nov.- Dec. 2019.

[2] J. Shen, T. Zhou, D. He, Y. Zhang, X. Sun and Y. Xiang, "Block Design-Based Key Agreement for Group Data Sharing in Cloud Computing," in IEEE Transactions on Dependable and Secure Computing, vol. 16, no. 6, pp. 996-1010, 1 Nov.- Dec. 2019.

[3] L. Qing, Z. Boyu, W. Jinhua and L. Qinqian, "Research on key technology of network security situation awareness of private cloud in enterprises," 2018 IEEE 3rd International Conference on Cloud Computing and Big Data Analysis (ICCCBDA), Chengdu, 2018, pp. 462-466.

[4] J. Shen, T. Zhou, D. He, Y. Zhang, X. Sun and Y. Xiang, "Block Design-Based Key Agreement for Group Data Sharing in Cloud Computing," in IEEE Transactions on Dependable and Secure Computing, vol. 16, no. 6, pp. 996-1010, 1 Nov.- Dec. 2019.

[5] Thabit, F., Alhomdy, S., Al-Ahdal, A. H., & Jagtap, S. (2021). A new lightweight cryptographic algorithm for enhancing data security in cloud computing. Global Transitions Proceedings., 2, 91–99.

[6] Yang, P., Xiong, N., & Ren, J. (2020). Data security and privacy protection for cloud storage: a survey. IEEE Access, 8, 131723–131740. special section on emerging approaches to cyber security.

[7] Kanatt, S., Jadhav, A., & Talwar, P. (2020). Review of secure file storage on cloud using hybrid cryptography. International Journal of Engineering Research & Technology (IJERT), 9(2), 16–20.

[8] Poduval, V., Koul, A., Rebello, D., Bhat, K., & Wahul, R. M. (2020). Cloud based secure storage of fles using hybrid cryptography and image steganography. International Journal of Recent Technology and Engineering (IJRTE), 8(6), 665–667.

[9] Ibrahim, D. S. (2019). Enhancing cloud computing security using cryptography & steganography. Iraqi Journal of Information Technology, 9(3), 191–224.

[10] Fateh, M., Rezvani, M., & Irani, Y. (2021). A new method of coding for steganography based on lsb matching revisited. Hindawi Security and Communication Networks, 2021(6610678), 1–15.

[11] Albalawi, A., & Hamza, N. (2020). A survey on cloud data security using image steganography. International Journal of Advanced Computer Science and Applications, 11(1), 645–649.

[12] Kumar, U., & Prakash, J. (2020). Secure fle storage on cloud using hybrid cryptography algorithm. International Journal of Creative Research Thoughts, 8(7), 334–340.

[13] Gabriel, A.J., Adetunmbi, A.O., & Obaila, P. (2020). A two-layer image-steganography systemfor covert communication over enterprise network. Proceedings of ICCSA 2020, LNCS 12254 (pp. 459–470), Springer Nature.

[14] Sivan, R., & Zukerman, Z. A. (2021). Security and privacy in cloud-based E-health system. Symmetry, 13, 742.

[15] Reddy, P. M. (2019). Hybrid crypto system for cloud storage security using MECC and Native bayes classifer A review. International Journal of Computer Sciences and Engineering, 7(6), 87–90.

[16] Akeel, W., & Hashim, A. (2017). Using steganography for secure data storage in cloud computing. International Research Journal of Engineering and Technology, 4(4), 3668–3672.

[17] Singh, A. K., Singh, J., & Singh, H. V. (2015). Steganography in images using LSB technique. International Journal of Latest Trends in Engineering and Technology, 5(1), 426–430.

[18] Karuna Jyothi, K., & Indira Reddy, B. (2018). Study on virtual private network (VPN), VPN's protocols and security. International Journal of Scientifc Research in Computer Science, Engineering and Information Technology, 3(5), 919–932.

[19] Zhang, Z., Chandel, S., Sun, J., Yan, S., Yu, Y., & Zang, J. (2018). VPN: a boon or trap? A comparative study of MPLs, IPSec, and SSL virtual private networks. In Proceedings of the 2018 2nd international conference on computing methodologies and communication (ICCMC), (IEEE), Erode, India (pp. 510–515).

[20] Ikram, M., Vallina-Rodriguez, N., Seneviratne, S., Kaafar, M. A., & Paxson, V. (2016). An analysis of the privacy and security risks of android VPN permission-enabled apps. In Proceedings of the 2016 internet measurement conference, Santa Monica, CA, USA, (ACM) (pp. 349–364).

[21] Mukhopadhyay, B., Bose, R., & Roy, S. (2020). A novel approach to load balancing and cloud computing security using SSL in IaaS environment. International Journal of Advanced Trends in Computer Science and Engineering,