A Comparative Performance Analysis for Compression Techniques in Cloud Storage

[1] V.Sherine Vargheese , [2]Dr. D. Maruthanayagam
[1]Research Scholar, Sri Vijay Vidyalaya College of Arts & Science, Dharmapuri, Tamilnadu, India
[2] Head/Professor, PG and Research Department of Computer Science, Sri Vijay Vidyalaya College of Arts & Science, Dharmapuri, Tamilnadu, India

Abstract: Cloud Storage becomes more efficient utilization for users with storage space and makes user friendly and timely acquire data, which is ***base of all categories of cloud applications.*** In cloud application system, ***a critical issue is analyzed by insufficient available memory affecting the performance and reliability of the device***. A novel solution manages this issue by compressing the blocks of memory even if the system runs out of memory (Insufficient access) and then it decompress the memory when it is required again, therefore the memory space can be automatically reallocated. With the support of data compression handles the size of text data and accumulating the same amount of data in relatively fewer bits, at the ***end of resulting to reduce the data storage space, transmission capacity or resource utilization.*** One of the most challenging tasks is Cloud storage space optimization and it includes different compression algorithms that are available in different formats. ***In this paper, discussed about the different compression techniques and their comparative performances were thoroughly analyzed***.

Keywords: Cloud Storage, Data Compression Techniques, Lempel-Ziv-Welch (LZW) Coding, Huffman Coding, Run Length Encoding Algorithm, and LZ77 algorithm.

I. INTRODUCTION

Cloud computing involves shared resources, different services such as platform, software, infrastructure that has provided on demand service over internet. It provides with the competence of large storage capacity and makes the virtual infrastructure for users with remote computing [1]. ***At present times, several cloud computing service provider are utilized such as Google, Amazon, Microsoft, IBM, Adobe and so forth.*** To utilize the online access for various resource and also for data storage management, for instance **S3 (Simple Storage Service)** built on the Google drive and Amazon EC2 etc. Cloud storage becomes an important feature which manages the IT industry. The application of cloud storage is the most developed part which depends on the distributed file system, cluster application, and grid technology also it affords storage service to user via internet [2]. ***The main benefits of the cloud storage enables users at anywhere and anytime to access the data.*** Cloud storage system automatically analyses user's requirements and locate and then transform the data, which greatly make easy to the users. Cloud computing simply performs with the work-flow with unlimited resources and aids to make efficient frameworks like that storing a large amount of data for additional processing [3]. However, the large amount data requires larger volume of storage space and energy during the transmission time through the network. Because of the information explosion [3], data reduction technologies like compression and de-duplication have been developed to improve the space efficiency in storage systems, including the cloud storage environment. The main requirement of data compression is to reduce the storage space of data. ***Data compression process consists of a file (Audio, Video, and Text) can be transformed to another (compressed or zip) file***, such that the original file may be fully recovered from the original file without any loss of actual information. This process may be useful if one wants to save the storage space. For instance, if the data needs to store a 4MB file, it can be eminent to first compress into smaller size and save the storage space. ***The compressed files are exchanged over the internet during upload and download performs faster.*** The capability to reconstitute the original file processed from the compressed version at anytime. Data compression of encoding rules allows substantial decrease in the total number of bits to transmit or store a file than the original representation with the intention that it obtains less transmission time and less storage space while communicating over a network. Data compression techniques consist of two compression techniques: **lossless data compression and lossy data compression.** In lossless compression techniques, the compressed data can be retrieved accurately to the original data. One of the most popular techniques for **lossless compression is the LempelZiv(LZ)** compression methods [4]. Alternatively in lossy compression system, the decompressed data is not accurate to the original data and it might be little variation in error rates. For instance, JPEG image compression is a recognized lossy compression technique that performs in part by cutting off less important bits from information [5]. In this paper, the performance of compression algorithms evaluated with example cloud data.

II. DATA COMPRESSION TECHNIQUES

Data compression techniques have been utilized in a compatible way for achieving effective Compression Ratio irrespective of file formats and size [6]. During compression time, the suitable representation of data manipulates the performance of a compression technique in different ways of representation exist like tree structure, matrix format, quantization, chain code etc. An unsuitable representation of data may result to larger compressed file than the original file representation. The representing information might give better Compression Ratio and it must be concentrated more in this area. Available memory management and Computational complexity plays a significant task in the design of Data Compression techniques [7, 8]. ***Most of the pre-existing techniques capable of handling high computational complexity issue. For Example,*** a critical issue performed ***in real time applications such as medical imaging, WSN (Wireless Sensor Nodes), Cloud Computing, satellite imaging, etc***. Because the applications are provided energy constrained and delay sensitive, computationally inexpensive techniques required to be developed. Also, compression technique raises the complexity in memory management. Even if the amount of memory needs to execute the compression technique exceeds the available system memory, effective compression cannot be achieved. Although some compression methods achieve better CR, it does not run the available memory efficiently [9]. **Finally, memory management** in compression technique **is another open research area.** Several types of data compression algorithms have been proposed most of them used lossy and lossless data compression algorithm. ***In this paper,*** thoroughly observes the ***performance***

of the Lempel-Ziv-Welch (LZW) Coding algorithm, Huffman Encoding Algorithm, Run Length Encoding Algorithm (RLE) and LZ77. These listed algorithms used for compressing text data were compared and analyzed. [10].

2.1 Lempel-Ziv-Welch (LZW) Coding Algorithm

LZW is considered a universal lossless data compression algorithm [10] invented by Abraham Lempel, Jacob Ziv, and Terry Welch. LZW was issued by Welch in 1984 as a extended implementation of the LZ78 algorithm issued by Lempel and Ziv in 1978. ***LZW provided simple implementation, and also the potential for very high throughput during hardware implementations*** [11]. As the dictionary-based encoding technique referred by LZW. The algorithm makes a data dictionary (also known as a string table or translation table) of data arising in an uncompressed data stream. Patterns of data (substrings) are recognized in the data stream and are equivalent to entries in the dictionary. The 12 bit code representing the sequence is placed in the compressed file instead of the sequence. In the table, the first 256 entries represent the single byte value from 0 to 255, even as the remaining entries correspond to sequences of bytes. ***The LZW algorithm has performed in an efficient mode of producing the code table,*** which depends upon the ***particular data being compressed.*** The following steps describes clearly about LZW:

1. Initiate dictionary to include single entry for each byte. And then the encoded string initialized with the first byte of the input stream.
2. The next byte read from the input stream.
3. If the byte is in state of an EOF (End of Finish) go to step 6.
4. Or else the byte concatenates to the encoded string generates a string which is present in the dictionary:
    - Concatenate the byte to the encoded string.
    - Go to step 2.
5. If concatenating the byte to the encoded string generates a string which is not present in the dictionary:
    - Insert the new string to the dictionary.
    - The code written for the encoded string to the output stream.
    - The encoded string set as equal to the new byte.
    - Go to step 2.
6. Write out code for encoded string and exit.

2.2 Huffman Coding Algorithm

Huffman Coding Algorithm was developed and published by David in the year of 1952. *Paper submits on "A Method for the Construction of Minimum- Redundancy Codes" [13]*. Huffman coding is considered as an entropy ***encoding algorithm for lossless technique***. This coding algorithm refers to the ***utilization of a variable-length code table to encode a source*** symbol. The variable-length code table has been derived based on the expected probability of prevalence for each possible value of the source symbol. Huffman coding utilizes a particular technique for selecting the representation of each symbol, resulting in a prefix code that states the most widespread source symbols using shorter strings of bits than are used for less common source symbols.

Algorithm: Encoding

1. Read Text File.
2. Obtain the probabilities of every character in the file.
3. Sort the characters in descending order in accordance with their probabilities
4. Generate the binary tree from left to right until you have just one symbol left.
5. Read the tree from right to left assigning different bits to different branches.
6. Store the final Huffman dictionary.
7. Encode each character in the file by referring to the dictionary.
8. Send out the encoded code along with the dictionary

Algorithm: Decoding

1. Read the encoded code bitwise.
2. Look for the code in the dictionary
3. If it is a match obtains, its corresponding symbol else read the next bit and repeat Steps 3 & 4
4. Write the decoded text in a file

2.3 Run Length Encoding (RLE) Algorithm

RLE *is the simplest data compression algorithm*. In this algorithm, the consecutive sequences of symbols are recognized as runs and the other sequences are recognized as non runs. ***This algorithm handles some kind of redundancy.*** It checks whether any repeating symbols or not, which depends upon those redundancies and their lengths. Consecutive recurrent symbols are recognized as runs and the entire other sequences are referred as non-runs. For instance, the text “ABCCCCB” is referred as a source to compress, the **first 3 letters are referred as a non-run with length 3**, and then the **next 4 letters are regarded as a run with length 4** because a repetition of symbol is C. The key task of this algorithm is to recognize the runs of the source file; also it is used to record the symbol and the length of each run [14]. This scheme has two potential issues. First issue is that an escape character may actually take place in the file. The answer is to make use of two escape characters to represent it, and that can actually make the output file larger if the uncompressed input file consists of occupying lots of escape characters. The second issue is that a single byte does not identify run lengths greater than 256. This complexity can be handled by using multiple escape sequences to compress one long string. RLE is not very helpful for compressing text files because a typical text file does not have a lot of long, repetitive character strings. ***RLE is helpful for compressing bytes of a monochrome image file,*** which consists of solid black picture bits, or "pixels", in a sea of white pixels, or the reverse. Run-length encoding is frequently utilized as a preprocessor for other compression algorithms.

Algorithm:

1. The character chose from source string.
2. Append the chosen character to the destination string.
3. Count the number of subsequent occurrences of the chosen character and add the count to destination string.
4. Select the next character and consecutively repeat steps 2, 3 and 4 if end of string is NOT reached.

2.4 LZ77

LZ77 algorithms accomplish **compression by replacing continuous volumes of data** with references to one copy of that data pre-existing earlier in the input (uncompressed) data stream. A pair of numbers is known as a ***length distance pair which is to be encoded using a match***, which is equivalent to the statement "each of the next length characters is equal to the characters exactly distance characters behind it in the uncompressed stream". To mark matches, the encoder should maintain track of some quantity of the most recent data, like the last 2 kB, 4 kB, 6 kB or 32 kB. The structure in which this data is apprehended is known as a *sliding window*, so that ***LZ77 is performed as sliding window compression.*** The encoder wants to keep this data to appear for matches, and the decoder wants to remain this data to interpret the matches the encoder refers to. The larger the sliding window compression analyzed that the longer support back the encoder may possibly search for making references. Even if it is not acceptable other than frequently helpful to permit length-distance pairs to locate a length that in point of fact exceeds the distance. As a copy command, this is puzzling: "Go back four characters and copy 10 characters from that position into the current position"[15]. How preserve ten characters be copied over even if only four of them are actually present in the buffer? Tackling one byte at a time, there is no issue of serving this request, for the reason that a byte is copied over and fed over again while the input keeps into the copy command. While the copy-from position creates it to the initial receiver position, so that it has accordingly fed data that was pasted from the starting of the copy-from position. The operation is thus equivalent to the statement "copy the data you were given and repetitively paste it until it fits". Encoding-Pseudo code algorithms are provided as follows [16]:-

- Step1: Initially check whether i look-ahead buffer is empty or not
- Step2: If not empty, look for the longest match in search buffer.
- Step3: If it is found print output as (offset from window boundary, length of match, next symbol in lookahead buffer) and shift window by length+1 else print output as (0,0,first symbol in look-ahead buffer) and shift window by 1.
- Step4: Loop until the look ahead buffer is empty.

III. EXPREMENTAL RESULTS

The experimentation result analyzes RLE, Huffman coding, LZ77 and LZW. The entire chosen lossless algorithms are implemented using Java. The comparative analysis of data compression algorithms selected different types of test text files are shown in Figure 1. The algorithms are executed on Intel(R) Core(TM)2 Duo CPU T6600 @ 2.20GHz, 2.20Ghz with RAM: 4.00GB (3.50 GB usable), Windows 7 (64bit). The performances of the algorithms based on the size of the source file and text patterns and the organization of different symbols in the source file. Hence, the research analyze process completed to include text files of different types like source codes, e-books in pdf files, notepad files, and etc, and different file sizes are utilized as source files. An algorithm provides an acceptable saving percentage with reduced time period for compression and decompression is considered as the most excellent technique. The performance is analyzed using five parameters such as Compression Factor, Compression ratio, Compression time, Decompression Time and Saving Percentage.

- **Compression Ratio:** It is considered as a percentage that outputs from partitioning the compression size (in bits) through the original file size (in bits) after that multiplying the result by 100% [17].
- **Compression Time:** Time taken manages for both compression and decompression have to be taken into examined because in some cases compression time and decompression time to be considered in some cases both of them are required.
- **Entropy:** Entropy determines the measurement of the totality of information in file. It is number that is conclusion from partitioning the compression size in bits by number of symbols in the original file and scale as bits per symbol [4].

Compression Time

The compression times of chosen files for all the algorithms are shown in Figure 1. The compression time is maximized as file size increases. For **RLE Compression time is less as well as be a constant value and not affected by the file size compare than Huffman,LZ77 and LZW**. Compression times are average values for LZ77 approach is smaller than the other algorithm. The LZW algorithm performed with small files other than not for the large files because of the size of the dictionary. ***Compression times of Huffman algorithm is analyzed the lowest than LZ77.***

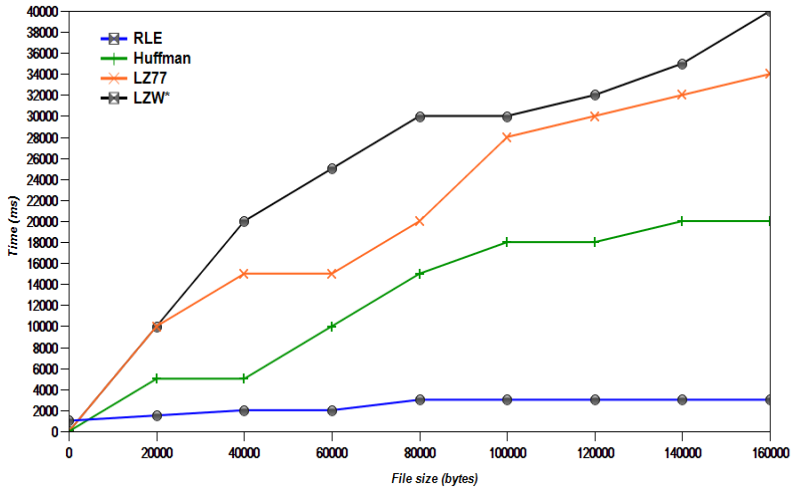

Figure 1: Compression time comparison

Decompression Time

The decompression times of all the algorithms are illustrated in Figure 2. The decompression times of all the techniques are relatively less than 500000 milliseconds except these two techniques the Huffman Algorithm and LZW. Again the decompression times of the Huffman and LZ77 algorithms are approximately similar performances. **The decompression times of the RLE algorithms are comparatively less for the entire file sizes than other algorithms (Huffman, LZ77 and LZW).**

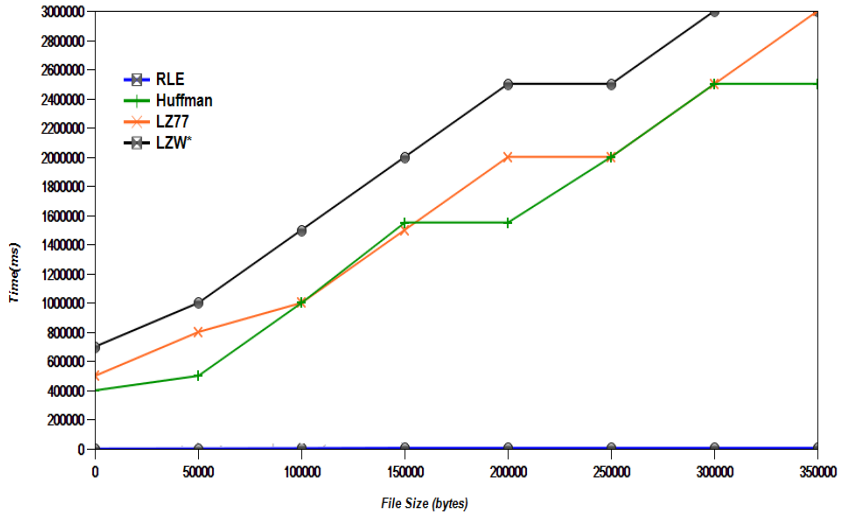

Figure 2: Decompression time comparison

Compression Ratio

The compression Ratio is analyzed with all the algorithms and the results of compressing different types of files. The performance of LZW is found as the best when compared with the parameters Compression Ratio. **LZW provides better compression ratio for highly correlated data than other algorithms (Huffman, RLE, and LZ77)** in Figure 3. However in large file size, larger dictionary is needed causing higher efforts for addressing and administration both at runtime. Also as the file size ***increases the compression efficiency*** of the algorithm increases because of increase in size of dictionary.

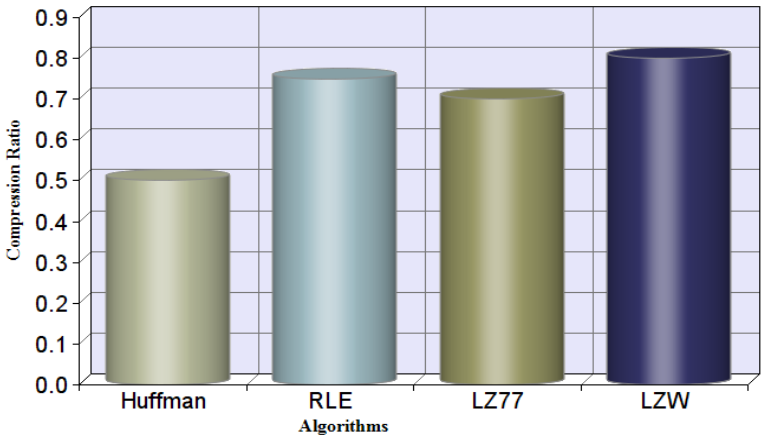

Figure 3: Comparison with compression ratio

Saving Percentage

Saving percentage of the RLE is performed as a very less value illustrated in figure 4. The compressed file sizes of all the Huffman algorithms are similar performances. The compressed file size is maximized in accordance with the original file size except for the LZW technique. This involves that the saving percentage of this algorithm highly depends on the redundancy of the file. **The LZW algorithm illustrates the highest saving percentage than other selected algorithms (Huffman, RLE, and LZ77).** However, it fails when the file size is large so that it cannot be used in such scenarios.

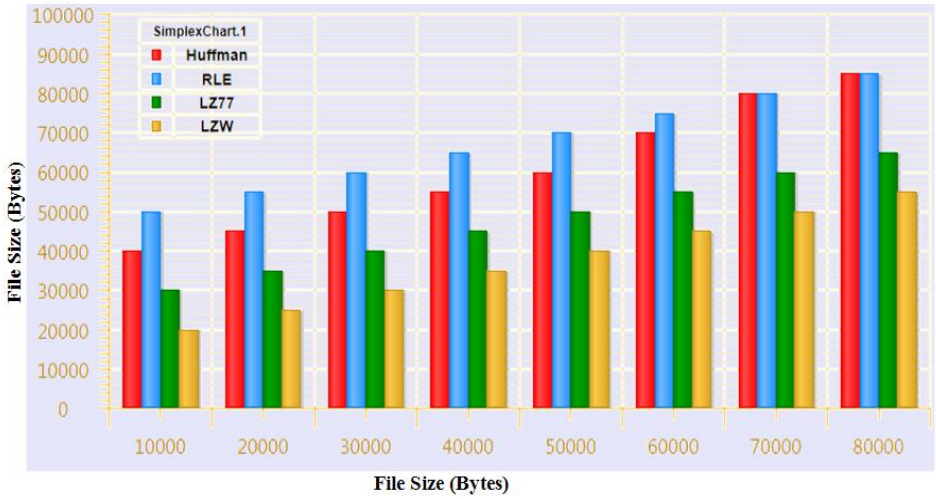

Figure 4: Saving percentage

IV. CONCLUSION

Data Compression techniques play an important role to deal with huge volume of data produced in different forms in digital world. No universal Data Compression approaches have been proposed to efficiently compress different forms of data in diverse applications. ***Several Data Compression algorithms are proposed to compress in different varieties of data such as audio, video, text, images and so forth.*** From ***this paper***, the performance of the data ***compression techniques analyzed and compared RLE, Huffman, LZW, LZ77*** were tested and experimented against standard files, text files, PowerPoint files, document files, and Portable Document Format files are presented. ***From the above results the LZW algorithm provides better compression ratio Compare than other Algorithms like Huffman, RLE and LZ77.*** However in large files it obtains more compression and decompression time than other algorithms. ***Depending on storage and time the RLE algorithm is better than other pre-existing algorithms (Huffman, LZW, and LZ77)***. In future, this current research focused on the common ***novel*** implementation ***to minimize computational complexity of LZW encoding using any new data structures or algorithm.***

V. REFERENCES

[1] Daniel J. Abadi, "Data Management in the Cloud: Limitations and Opportunities", IEEE Data Engineering Bulletin, Volume 32, March 2009, 3-12.

[2] James Broberg, Rajkumar Buyya, Zahir Tari, "MetaCDN: Harnessing "Storage Clouds" for high performance content delivery", Journal of Network and Computer Applications, 1012–1022, 2009.

[3] A. R. Shovon, S. Roy, T. Sharma, and M. Whaiduzzaman, "A restful e-governance application framework for people identity verification in cloud," in International Conference on Cloud Computing. Springer, 2018, pp. 281–294.

[4] R. Naqvi, R. Riaz, and F. Siddiqui, "Optimized rtl design and implementation of lzw algorithm for high bandwidth applications," Electrical Review, vol. 4, pp. 279–285, 2011.

[5] C. Arcangel, "On compression," Retrieved 6 March 2013, 2013.

[6] Nicolae,B., "High Throughput Data Compression for Cloud Storage", Springer 2012.Vol.11,pp 48-54.

[7] Khalid Sayood, "Introduction to Data Compression", Ed Fox (Editor), March 2000

[8] Ian H H. Witten, Alistair Moffat, Timothy C. Bell , "Managing Gigabytes: Compressing and Indexing Documents and Images", May 1999

[9] S. Porwal, Y. Chaudhary, J. Joshi and M. Jain , " Data Compression Methodologies for Lossless Data and Comparison between Algorithms", International Journal of Engineering Science and Innovative Technology (IJESIT) Volume 2, Issue 2, March 2013.

[10] Roger seeck, "General Principle of RLE"

[11] Draft Lecture Notes,"compression Algorithms: Huffman and Lempel-Ziv-Welch (Lzw)". Last Update: February 13, 2012.

[12] Haroon A, Mohammed A, "Data Compression Techniques on Text Files: A Comparison Study", International Journal of Computer Applications (0975 – 8887) Volume 26– No.5, July 2010.

[13] Vikas Singla, Rakesh Singla and Sandeep Gupta, "Data compression modelling: Huffman and Arithmetic", International Journal of The Computer, the Internet and Management, Vol. 16 No.3, Page(s):64- 68.Sept-Dec, 2008

[14] S.R. Kodituwakku," Comparison of Lossless Data Compression Algorithms for Text DaTA", Indian Journal of Computer Science and Engineering , Vol 1 No 4 416-425.

[15] C:\Documents and Settings\DELL\Local Settings\temp\IM\LZSS (LZ77) Discussion and Implementation.mht.

[16] T.Patel, J. Anjela, P. Choudhary, K. Dangarwala , "Survey of Text Compression Algorithms", ISSN 2278-0181 Vol, 4 Issue 03, March- 2015.

[17] Haroon Altarawneh and Mohammad Altarawneh, "Data Compression Techniques on Text Files:A Comparison Study" International Journal of Computer Applications (0975 – 8887) Volume 26– No.5, July 2011.

About the Authors

.
V.Sherine Vargheese received her M.Phil degree from Periyar University, Salem in 2007. She received her Master's degree from Bishop Heber College affiliated to Bharathidasan University,Tiruchirapalli in 2004. She is pursuing her Ph.D Degree (Part-Time) in Sri Vijay Vidhyala College of Arts & Science, Dharmapuri, Tamilnadu, India. She is working as HOD Cum Assistant Professor in PG Department of Computer Science at Kailash Women's College, Nangavalli, Salem. Her current research interests include Cloud Computing and Computer Networks.

Dr.D.Maruthanayagam received his **Ph.D** Degree from Manonmaniam Sundaranar University, Tirunelveli in the year 2014. He received his **M.Phil** Degree from Bharathidasan University, Trichy in the year 2005. He received his **M.C.A** Degree from Madras University, Chennai in the year 2000. He is working as **Head & Professor**, PG and Research Department of Computer Science, Sri Vijay Vidyalaya College of Arts & Science, Dharmapuri, Tamilnadu, India. He has **18 years** of experience in academic field. He has published **5 books**, **34 papers** in International Journals and **30 papers** in National & International Conferences so far. His areas of interest include Computer Networks, Grid Computing, Cloud Computing and Mobile Computing.