Arithmetic Coding- A Reliable Implementation

Lakshmi Sasilal
CSED, NIT Calicut
Kerala

Dr. V. K. Govindan
CSED, NIT Calicut
Kerala

ABSTRACT
Arithmetic compression scheme is one of the commonly used techniques to represent more amount of information using the available units of resources. It has been known that arithmetic coding has a better coding efficiency than other compression schemes. However, when used in an error prone environment, the poor error resistance property of the method is a severe disadvantage. It is difficult to locate an error when arithmetic coding is used and a large portion of a data must be discarded when an error occurs. In this paper, a novel technique is proposed to improve the error resilience of arithmetic coding, in which the decoder is less affected by the errors caused in the transmission of data over the network. A comparative study with the basic algorithm demonstrates that the time performance of the error resilient arithmetic coder is somewhat comparable to the basic algorithm.

General Terms
Arithmetic Coding, Error Resilient Coder

Keywords
ERAC, Arithmetic compression,

1. INTRODUCTION
By means of data compression, the aim is to reduce the redundancy in the data which is being stored or communicated, and thereby increase the data density. Data compression has important applications in the areas of data storage and transmission. It allows a user to store more information in the memory device available than otherwise possible. There are many data handling applications which require the processing of large amount of data, and the number of such applications is constantly increasing. At the same time, the proliferation of computer communication networks is resulting in massive transfer of data over communication links. Compressing the data to be stored and transmitted helps to reduce the storage and communication costs involved in the process. When the size of data being transmitted is reduced, the capacity or bandwidth of the communication channel is used more efficiently. Similarly, compressing a file to lesser size helps to make use of the storage capacity of the medium more efficiently. Various methods have been suggested by researchers, and the information theory has been made use of in various ways to eliminate the redundancies to achieve better compression performances.

The techniques used to compress data can be broadly classified into two. They are lossless compression techniques and lossy compression techniques. Lossless compression reduces bits by identifying and eliminating statistical redundancy. No information is lost in lossless compression. Lossy compression reduces bits by identifying unnecessary information and removing it.

Arithmetic compression scheme is one of the commonly used techniques to represent more amount of information using the available units of resources. It has been known that arithmetic coding has a better coding efficiency than other compression schemes. [1] discusses arithmetic coding scheme and claims to be superior in most respects to the better-known Huffman method. The present work mainly focuses on improving the error resilience of arithmetic coding.

Arithmetic coding is based on the frequency of symbols, and the whole message is represented by a fractional number between 0 and 1. The exact range (within 0 and 1) in which the number belongs depends on the length of the data, the symbols and their locations and frequency within the data string. The number must be represented without any loss of precision. As the message size becomes larger, the interval in which the number belongs becomes narrower.

The most important advantage of arithmetic coding is its flexibility: it can be used in conjunction with any model that can provide a sequence of event probabilities. Optimality is another important advantage of arithmetic coding.

A major disadvantage of arithmetic coding is its poor error resistance. A single bit error in the encoded file causes the decoder's internal state to be in error, making the remainder of the decoded file corrupted. Universal access to data is one of the major objectives of emerging communication systems. The extension of services offered to mobile users, from traditional voice traffic to complex data sources such as web-pages, images, video and music demands higher quality of services in terms of data accuracy and speed. Arithmetic compressor allows the transfer of large volume of data with limited resources, but with less reliability. While using arithmetic compression, a single bit error may cause the retransmission of the entire data.

2. RELATED WORKS
Arithmetic coding is an active topic of research since for many decades. Some of the major disadvantages of Arithmetic coding are that its implementation is complex and time consuming, and the basic algorithm has no resiliency to coding errors. There are a number of publications dealing with fast implementation and error resilience. Some of these works are reviewed briefly in the following:

One of the important works is that by Witten et al. [1]. They present a good tutorial on arithmetic coding- how it works, its practical implementation and the efficiency of the arithmetic coding. Later in 1991, Alistair et al. [2] proposed a new implementation of arithmetic coding incorporating several improvements over a widely used earlier version. These

improvements include fewer multiplicative operations, and greatly extended range of alphabet sizes and symbol probabilities.

A major drawback of arithmetic coding is its slow speed due to the requirement of multiplications and divisions. A few attempts have been done to avoid multiplications and to improve the efficiency. Langdon and Rissanen [3] proposed a modified scheme for encoding a binary string by using shift-and-add. Rissanen and Mohiuddin [4] proposed a multiplication-free algorithm for encoding general string. The method was further developed by Lei [5]. Howard and Vitter [6] described an efficient bit-level implementation that uses table lookup as a fast alterative to arithmetic operations.

Bassiouni et al. [7] has proposed a variant of arithmetic coding scheme which can be used for data compression in supercomputers. The approach makes use the correlation of adjacent characters to improve the compression performance. The scheme is based on the observation that in many data files, data records exhibit strong locality behaviour of character reference.

Another work that attempts to improve the computational performance is that by Jianjun Zhang and Xingfang Ni [8]. They present a new implementation of bit-level arithmetic coding using integer additions and shifts. The algorithm has less computational complexity and more flexibility, and thus is very suitable for hardware design.

Several preprocessing techniques are discussed in [9], [10], [11]. In [9], they propose RIDBE (Reinforced Intelligent Dictionary Based Encoding). It is a reversible lossless, dictionary based text transformation algorithm. The basic idea of the secure compression they have proposed is to preprocess the text and transform it into some intermediate form which can be compressed with better efficiency and which exploits the natural redundancy of the language in making the transformation.

As mentioned before, arithmetic decoder has poor error resilience property. David W. Redmill et al. [12] shows that without the Error Resilient Entropy Code, arithmetic coding is less resilient than Huffman coding and examines the use of arithmetic coding in conjunction with the error resilient entropy code.

A technique to implement error detection as part of the arithmetic coding process by introducing a small amount of extra redundancy is described by Colin Boyd et al. [13]. Redundancy is introduced by adjusting the coding space so that some parts are never used by the encoder. During decoding, if the number defined by the received encoded string ever enters the restricted region, an error in communication must have occurred.

A joint lossless-source and channel coding approach that incorporates error detection and correction capabilities in arithmetic coding is proposed by G .F. Elmasry in [14]. The encoded binary data representation allows the source decoder to recover the source symbols, even with channel errors. Later G .F. Elmasry in [15] modified his algorithm in such a way that the decoder utilizes Hamming distance, the knowledge of the source statistics, and the self-synchronization property of arithmetic coding to recover the encoded sequence of symbols.

Zhi-Quan Cheng et al. [16] proposed an error resilient arithmetic coding algorithm for coding 3D content inspired by error resilient JPEG 2000 image coding standards. The approach makes use of an error resilient extended multiple quantization coder which divides bit stream into small independent parts, employing periodic termination markers, which permits basic transmission error containment.

Another proposal for error resilient arithmetic coding is the JPEG2000 based one for wireless image transmission by Yipeng et al. [17]. The authors demonstrated the effectiveness of the proposal for improving end-to-end reconstructed image quality. In still another work, H. Morita et al. [18] presented an error resilient variable length arithmetic code employing sync markers of fixed length. The approach found to provide suppression of error rate at the expense of increased redundancy.

The rest of this paper is organized as follows: Section 3 describes the arithmetic coding and its implementation. The proposed improved arithmetic coder with probability Scaling and the error resilient coding algorithm are presented in Section 4. Results and performance comparison of the proposed approach are discussed in Section 5, and finally the paper is concluded in Section 6.

3. ARITHMETIC CODING

This section deals with the working and implementation of the arithmetic coding based on that of Witten et al. [1].

3.1 Basic Algorithm

The algorithm for encoding a file using this method works conceptually as follows [1]:

1. The "current interval" [L,H) is initialized to [0,1).
2. For each symbol that has to be encoded, do the following procedure:
   a) Subdivide the current interval into subintervals, one for each possible alphabet symbol. The size of a symbol's subinterval is proportional to the estimated probability that the symbol will be the next symbol in the file according to the model of the input.
   b) Select the subinterval corresponding to the symbol that actually occurs next in the file, and make it the new current interval.
3. Output enough bits to distinguish the final current interval from all other possible final intervals.

3.2 Implementation

The basic implementation described above has two major difficulties: the shrinking of current interval requires high precision arithmetic and no output is produced until the entire file has been read. Witten et al. [1] has suggested a clever solution for preventing the current interval from shrinking too much when the end points are close to 1/2, but straddle 1/2. In that case nothing is known about the next output bit, but whatever it is, the following bit will have the opposite value; hence merely keep track of that fact and expand the current interval symmetrically about 1/2. This follow-on procedure may be repeated any number of times, so as to keep the current interval size longer than 1/4. This expansion process takes place immediately after the selection of the subinterval corresponding to an input symbol. Repeat the following steps as many times as possible:

a) If the new subinterval is not entirely within one of the intervals [0, 1/2), [1/4, 3/4), or [1/2, 1), stop iterating and return.
b) If the new subinterval lies entirely within [0, 1/2), output 0 and any 1s leftover from previous symbols; then double the size of the interval [0, 1/2), expanding toward the right.
c) If the new subinterval lies entirely within [1/2,1), output 1 and any 0s left over from previous symbols; then double the size of the interval [1/2,1), expanding toward the left.
d) If the new subinterval lies entirely within [1/4, 3/4), keep track of this fact for future output; then double the size of the interval [1/4, 3/4), expanding in both directions away from the midpoint.

4. PROPOSED MODEL

4.1 Improved Arithmetic Coder with probability Scaling

It is useful to divide the process of data compression into two logically disjoint activities: statistical modelling, and coding. One can think of the model as the "intelligence" of a compression scheme, which is responsible for deducing or interpolating the structure of the input, whereas the coder is the "engine room" of the compression system. The efficiency of the coder largely depends on the input provided by the modeller, i.e. how well the symbol probabilities are estimated. This modular division helps to develop a highly independent modeller and coder regions in the implementation.

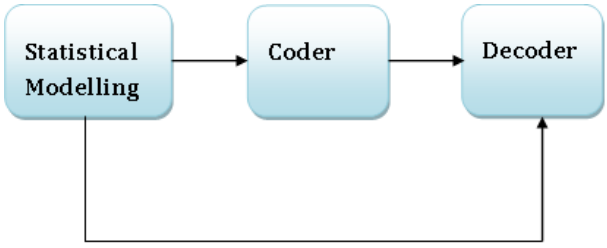

Fig 1: Stages in Arithmetic Coding/Decoding

Given an alphabet with symbols S_0, S_1, ... S_n, the modeller produces the symbol probability table which says that each symbol has a probability of occurrence of p_0, p_1, ... p_n. It is possible to represent each probability, p_i, as a unique non-overlapping range of values between 0 and $\sum p_i$. Using this information the coder encodes the input message and maps it to a range that denotes the entire message.

The mechanism that achieves this operates as follows. Suppose that p_i is the probability of the i^{th} symbol in the alphabet, and that variables L and R are initialized to 0 and 1 respectively. Value L represents the smallest binary value consistent with a code representing the symbols processed so far, and R represents the product of the probabilities of those symbols. To encode the next symbol, which (say) is the j^{th} of the alphabet, both L and R must be refined: L is replaced by $L+R*\sum_{i=1}^{j-1} p_i$ and R is replaced by $R* p_j$, preserving the relationship between L, R, and the symbols so far processed. At the end of the message, any binary value between L and L + R will unambiguously specify the input message. The computational performance largely depends on how best the cumulative probability distribution is calculated, and how best the arithmetic is performed.

Some architectures offer integer multiply and divide instructions in hardware, and execute them in the same time as other operations, for example, on an Intel Pentium Pro (200MHz). Except on those machines, the multiplications and divisions in the basic arithmetic coder is making it less efficient and slow. The simplest way to get rid of this disadvantage is to replace the multiplications and divisions by shift-and-add operations wherever possible.

For this purpose, the probability values estimated by the modeller are scaled to values by means of which the multiplication can be easily replaced by shift-and-add operations. This scaling is in such a way that it will not affect the compression ratio of the arithmetic coder, but at the same time improves its time complexity.

The approximation suggested here is to use $2^{-k(pi)}$, where $k(p_i)$ is an integer representing the bounds of probability range of the symbol p_i in the probability symbol table. Since this value is a power of 2, the multiplications can be easily replaced with a left shift operation and a division operation can be easily replaced with a right shift operation. Therefore, it provides an excellent compromise between good performance and low complexity.

4.2 Error Resilient Arithmetic Coder

The major challenge faced by the emerging communication systems is the rapid, universal access to the data, without a compromise with accuracy. The extension of services offered to mobile users, from traditional voice traffic to complex data sources such as web-pages, images, video and music along with the constraints imposed by the environment are boosting a considerable amount of research in the field of wireless communications. The networks are often loaded with large quantity of data being transferred to various destinations, and it demands a system with enhanced methods to handle data without causing any errors, at the same time making use of available resources in an efficient manner.

Compressing the data that has to be sent across the network is one of the widely used techniques to handle larger amount of data with the available bandwidth of the network. For this purpose a coder has to be present at the source, and the destination side has to be equipped with a decoder. Fig.2 depicts this scenario.

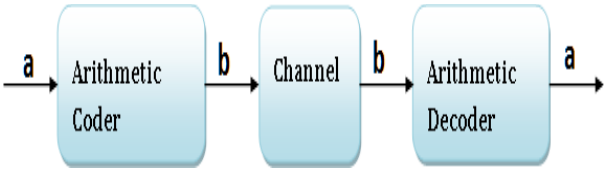

Fig 2: Transmission system block diagram

Currently, arithmetic coders are not widely accepted for this purpose even though they are much better than many other compressors. This is mainly because of the poor error resistance property of arithmetic coders. A single bit error in

the transmission stage affects the decoder badly and results in corrupted output. The bit change might be happening at the earlier stage of the transmission. That single bit change causes the entire message to be decoded into meaningless data. Fig.3 depicts this scenario.

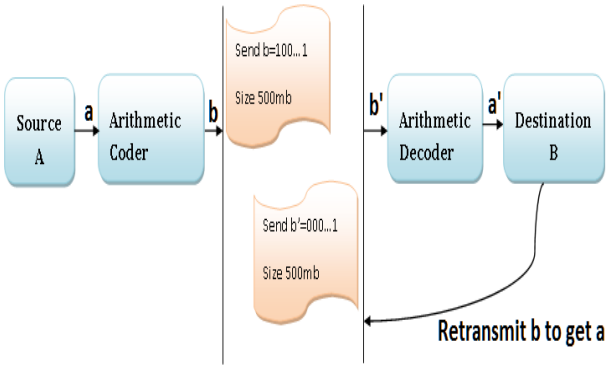

Fig 3: Single bit error causes retransmission of the entire message

In such cases, the retransmission of entire message can be avoided by making use of an error resilient arithmetic coder. A simple method to produce such a coder, Error Resilient Arithmetic Coder (ERAC) is suggested here.

In ERAC, the message or file that has to be transmitted is divided into several blocks. The size of blocks can be varying as well as fixed. In the case of a corrupted output, the number of bytes that has to be retransmitted depends on this block size. There exists a trade-off between the block size and the compression ratio and hence it has to be carefully chosen.

ERAC also makes use of an extra symbol to denote the end of a message block, in case the block size is varying. This helps the decoder to reconstruct the message without any errors.

The ERAC encoding algorithm is as follows:

1. Initialize the variables COUNT = 0, EOF_BLOCK = 256
2. The "current interval" [L,H) is initialized to [0,1).
3. For each symbol that has to be encoded, do the following procedure:
   a) Increment the value of COUNT by 1.
   b) If the value of count>BLOCK_SIZE, do:
      i. Encode the symbol EOF_BLOCK, denoting the end of a message block, so that the current interval denotes the message block that has been encoded so far.
      ii. Reset the value of current interval to [0,1], COUNT as 0
   c) Encode the symbol that has to be encoded, as done in the basic algorithm.
4. Output enough bits to distinguish the final current interval from all other possible final intervals.

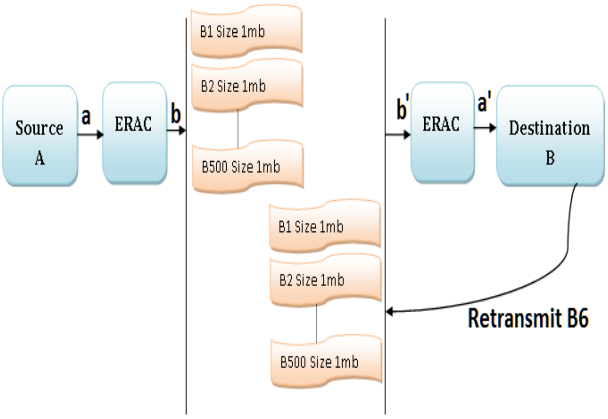

Fig 4: Single bit error causing the retransmission of one message block only.

The ERAC decoding algorithm is as follows:

1. The "current interval" [L,H) is initialized to [0,1).
2. Until end of file symbol is decoded, do the following procedure:
   a) Decode the encoded value into the corresponding symbol.
   b) If the symbol is not EOF_BLOCK, print it.
   c) Else, reset the current interval [L,H) to [0,1) and continue the decoding procedure.

5. RESULTS AND DISCUSSION

In addition to the error resilient property, ERAC provides a way to implement parallelism in basic arithmetic coders and decoders.

The proposed technique yields significant improvement in terms of reliability and reconstruction of the message, while limiting the complexity and compression ratio at a reasonable level. With the block size as 64, a file of size 1GB has an additional overhead of 8mb only, assuming an average of 4 bits for encoding an end of block symbol. The new algorithm has been implemented and compared with the implementation of the basic algorithm to demonstrate the performance. Table I shows the comparison results.

Table 1. Comparing ERAC time performance with naive algorithm

Name	Org Size (mb)	En. Time(basic algo)	En. Time(ERAC)
bible.txt	4	1m39.329s	1m40.781s
E.coli	4.6	1m7.651s	1m9.125s
world192.txt	2.5	1m5.768s	1m7.056s

6. CONCLUSION

A modified version of basic arithmetic compression scheme is proposed in this paper. The suggested model eliminates the poor error resistance property of the basic arithmetic coder and thus makes the decoder error resistant. The suggested ERAC makes the arithmetic compression scheme suitable for the use in network communications. The algorithm has been implemented and the comparison of results with basic scheme demonstrates the advantages of the proposed scheme.

7. REFERENCES

[1] I. H. Witten, R. M. Neal, and J. G. Cleary. "Arithmetic coding for data compression". Commun. ACM. 1987, 30:520-540.

[2] ALISTAIR MOFFAT,RADFORD M. NEAL and IAN H. WITTEN. "Arithmetic Coding Revisited". ACM Transactions on Information Systems, Vol. 16, No. 3, July 1998, Pages 256–294.

[3] G. G. Langdon, and J. J. Rissanen. "A simple general binary source code". IEEE Trans. Inf. Theory. 1982, IT-28:800-803.

[4] J. J. Rissanen, and K. M. Mohiuddin. "A multiplication-free multialphabet arithmetic code". IEEE Trans. Commun.1989, 37: 93-98.

[5] Shaw-Min Lei. "Efficient multiplication-free arithmetic codes". IEEE Trans. Commun. 1995, 43: 2950-2958.

[6] P. G. Howard, and J. S. Vitter. "Arithmetic coding for data compression". Proc. IEEE. 1994, 82: 857-865.

[7] Bassiouni, M.A.; Ranganathan, N.; Mukherjee, A.; "A scheme for data compression in supercomputers," Supercomputing '88. [Vol.1]. Proceedings. , vol., no., pp.272-278, 14-18 Nov 1988

[8] Jianjun Zhang and Xingfang Ni. "An Improved Bit-level Arithmetic Coding Algorithm". Journal of Information and Computing Science Vol. 5, No. 3, 2010, pp. 193-198

[9] S. Senthil, S.J Rexiline and L. Robert. "RIDBE: A Lossless, Reversible Text Transformation Scheme for better Compression" International Journal of Computer Applications (0975 – 8887) Volume 51– No.12, August 2012

[10] Fauzia S. Awan, Nan Zhang, Nitin Motgi, Raja T. Iqbal, and Amar Mukherjee. "LIPT : A Reversible Lossless Text Transform to Improve Compression Performance", Proceedings of the 2001 IEEE Data Compression Conference, IEEE Computer Society Press, Los Alamitos, California. Vol. 481. 2001.

[11] M. Burrows, D. J. Wheeler, M. Burrows, and D. J. Wheeler, "A block-sorting lossless data compression algorithm," Tech. Rep., 1994.

[12] Redmill, David W., and David R. Bull. "Error resilient arithmetic coding of still images." Image Processing, 1996. Proceedings, International Conference on. Vol. 1. IEEE, 1996.

[13] BOYD, C., CLEARY, J.G., IRVINE, S.A., RINSMA MELCHERT, I., and WITTEN, I.H.: 'Integrating error detection into arithmetic coding', *ZEEE Trans. Commun.*, 1997,45, (I), pp. 1-3

[14] ELMASRY, G.F.: 'Arithmetic coding algorithm with embedded channel coding', Electron. Lett., 1997, 33, (20), pp. 1687-1688

[15] Elmasry, G.F., "Embedding channel coding in arithmetic coding," Communications, IEE Proceedings- , vol.146, no.2, pp.73,78, Apr 1999

[16] Cheng, Zhi-Quan, et al. "An Error-Resilient Arithmetic Coding Algorithm for Compressed Meshes." Cyberworlds, 2008 International Conference on. IEEE, 2008.

[17] Sun, Yipeng, et al. "Error resilient arithmetic coding for wireless robust image transmission." Wireless Communications and Signal Processing (WCSP), 2011 International Conference on. IEEE, 2011.

[18] Morita, Hiroyoshi, Ying Zou, and Adriaan J. van Wijngaarden. "Design and analysis of synchronizable error-resilient arithmetic codes." Global Telecommunications Conference, 2009. GLOBECOM 2009. IEEE. IEEE, 2009.