An overview of streaming compression techniques

Benjamin V. Hugo
Department of Computer Science
University of Cape Town
bennahugo@aol.com

ABSTRACT

We provide a summary of current compression techniques and their applicability to high volume streaming data. The following techniques are discussed: RLE, Statistical methods such as Huffman and Arithmetic coding, LZ methods, Wavelet Transforms and Predictive Methods for time series. We believe that the compression ratios provided by predictive methods, LZ methods and RLE with variable-length coding may yield good results when applied in a context where processing capacity is limited due to the shear volume of streaming data received as input.

1. INTRODUCTION

The MeerKAT array outputs very large sets of correlated data (in the area of 1 terabyte per second), making it challenging to distribute and store the data. In light of this, the emphasis of this survey will be placed on fast compression techniques that can achieve moderate compression ratios, whilst keeping performance overheads to a minimum.

The concept of compression can ultimately be thought of as expressing information in a more compact form. For example one can eliminate unnecessary information such as repetition or encode frequently seen elements using fewer bits. Formally it is expressed as limiting statistical redundancy within data. The techniques for reducing redundancy include the following four general methods: run length encoding, statistical methods, LZ methods (dictionary-based) and transforms [10, p. 7].

It is further important to distinguish between algorithms that are lossless vs. lossy, as well as those algorithms operating in streaming mode vs. those operating in blocked mode.

An algorithm is said to be lossless when it only encodes information more optimally but does not remove any information. Lossy algorithms (seen in image compression formats such as JPEG) removes information that cannot easily be perceived, but this introduces error in the data [10, p. 8]. For example, the process of scalar quantization generally takes large numbers and cuts them down to shorter numbers (this can be done by removing some less significant bits) [10, p. 40]. In our context of use we refrain from using these lossy compression techniques due to the nature of the correlated radio astronomy data being compressed.

Algorithms operating in blocked mode such as the Burrows-Wheeler algorithm works on input read in large blocks of bytes. Many encoders, however, work in streaming mode; either a single or a small number of bytes are processed at a time, until the end of file marker is reached [10, p. 10].

We also note the following general performance metrics:

1. Compression ratio. A value of 0.6 indicates that the compressed data occupies 60% of the space of the original, uncompressed data. Values greater than 1 indicates negative compression.

$$\text{Compression ratio} = \frac{\text{size of output stream}}{\text{size of input stream}} \quad (1)$$

2. Compression factor. This is simply the inverse of the compression ratio. Values greater than 1 indicate compression and values less than 1 indicate expansion. A bigger value indicates better compression.

$$\text{Compression factor} = \frac{\text{size of input stream}}{\text{size of output stream}} \quad (2)$$

We will first consider some examples in the four main categories before discussing advance techniques focusing on predictive numerical compression.

2. RUN LENGTH ENCODING (RLE)

In simplest terms run length encoding refers to a simple (and old) scheme of eliminating repetition in runs of characters/bytes: if s = adddddeebbbb then RLE(s) = a@d5ee@b4. Usually a special symbol is used to denote a run of symbols. Instead of writing a repeated symbol, say d out 5 times, one can write d5. In light of this RLE is a good candidate for instances where information is likely to be repeated *locally* (for instance in image compression). [10, ch 1].

Ray et al. [8] investigates several compression techniques for database attribute compression for both text and numerical data. They found that RLE performs well in cases

where there are many redundant characters (as with fixed length string attributes in database tables). However, the compression ratio drops significantly when it is used to compress real-valued data. Abadi et al. [1] notes, when testing 32-bit integers, that RLE performs well on low-cardinality data (where there are few discrete values) and cases where there are larger run lengths (even for high cardinality data). One can, for instance, encode runs of zero bits in a shorter form if the numeric data in a scientific database is skewed toward the less significant bits as Bassiouni [2] suggests.

3. STATISTICAL METHODS

In Information Theory data redundancy is defined as the difference between the largest possible entropy of a symbol set and its actual entropy. Entropy is the measurement of the information contained in a single base-n symbol (as transmitted per time unit by some source). Mathematically redundancy is defined as follows (n is the size of a symbol set and P_i is the probability that a symbol c_i is transmitted from a source)[10, p. 46 - 47]:

$$R = \log_2 n + \sum_{1}^{n} P_i \log_2 P_i \tag{3}$$

One of the most commonly used algorithms is Huffman coding. It is a form of entropy encoding that compresses data by assigning shorter *integral* representations to symbols that occur more frequently [10, ch 2]. Cannane et al. [4] notes that Huffman coding cannot effectively be applied to cases where words, tokens or symbols cannot easily be extracted. These cases includes genome, numeric, scientific and binary formats.

There is another common entropy encoding technique known as arithmetic encoding. It differs from Huffman coding in the sense that it assigns real-valued intervals to each symbol (and can therefore more closely match the desired entropy for a particular symbol). The initial interval of [0,1) is refined with the processing of each consecutive character, by assigning it to a subinterval within the previous interval. The width of the interval is proportional to the probability with which the character currently being processed occurs [10, ch 2][14].

Both approaches lead to variable-size codes. Both techniques have *adaptive versions* which are useful in situations where the probability distributions change or have to be estimated. This approach is also applicable to the situation where the symbol table has to be computed on the fly, because it is impossible to perform multiple passes - as seen with streaming data [10, ch 2][8].

A well known example where arithmetic coding is used is in the JPEG2000 image compression standard (known as the MQ algorithm) [11]. It can, however, also be used for numeric compression because it allows for any probability distribution, as pointed out by Witten et al. [14] They add that arithmetic coding is faster than Huffman coding for adaptive situations. Analysis performed by Ray et al. [8] shows that both the adaptive and non-adaptive versions of arithmetic coding produces better results than their Huffman counterparts for real-valued data over larger collections. However, it must be pointed out that arithmetic-coding decompression is slow and is not tailored to situations where fast access is required [13].

4. DICTIONARY (LZ) METHODS

Unlike statistical methods, dictionary methods are adaptive methods which do not use variable-size codes. Instead they rely on the recurrences of entire phrases. Using a dictionary they encode variable-length phrases to fixed-length codes (hence they resemble entropy encoders in cases where they receive longer input). Two of the primary algorithms are LZ77 and LZ78. There are many variants on each of these, including LZSS and LZW. LZ77 uses a sliding window consisting of a search buffer, as well as a look-ahead buffer. The encoder scans the search buffer backwards trying to locate a match for the characters in the lookahead buffer. The matches are saved as length-distance pairs. LZSS improves upon LZ77 by holding the look-ahead buffer in a circular queue, storing the dictionary in a binary search tree and constructing tokens with two fields rather than the three-field construction used by LZ77. LZ78 abandons the sliding window approach by keeping a dictionary of strings encountered previously (and can therefore use considerable amounts of memory). It is possible to do a two part compression where, for example, LZ77 is followed by a statistical method like Huffman coding (the algorithm is then known as LZH). The general issue with many of the variants of these algorithms is the size of the dictionary. It can be overcome by breaking the input data into blocks. Below is a short list of historically popular applications of LZ methods [10, ch 3]:

1. The UNIX compress utility and the GIF image format build on a simpler, slow adapting variant of LZ78, called LZW.
2. The Microsoft ZIP and Cabinet formats
3. The popular RAR format combines an LZ variant with Huffman encoding.
4. The compression algorithm known as Deflate combines LZ77 with Huffman encoding. The technique is published in the public domain and is widely used: in the popular UNIX utility GZIP, standard protocols including HTTP and PPP, the PNG image format and Adobe PDF.

Bassiouni [2] suggests that LZ methods are primarily used for non-numeric data and are only applicable for header compression in large scientific databases. This notion is supported by Roth et al. [9] They state that LZW achieves very low compression ratios when compressing floating point data (even expansion in the case when the numbers are not of the same magnitude). They do however state that data, composed mostly of integers, compress well at around 50%.

Further analysis by Ray et al. [8] on a largely numerical database shows that LZ compression achieve lower compression ratios than arithmetic coding if not performed at a file level (ie. not on single attributes, records or pages) . However, as previously mentioned, arithmetic coding may be too slow to be employed in our context.

Williams et al. [13] suggests another approach to variable-length representations that work in time-contrained instances where either streaming or random access is required. They show that variable-length byte schemes constantly outperform LZ-based GZIP's compression ratios for large collections of integers (with both high and low entropies). However, GZIP is consistently faster than both variable-length byte and bit schemes.

These points raise the possibility of employing an LZ technique such as LZW if the data consists of integers (or floating point numbers in a very limited range), since it is rather fast and provides support for streaming data.

5. WAVELET TRANSFORMS

A wavelet can, in simple terms, be thought of as an oscillation localized to a finite interval (the amplitude is approximately zero outside the interval). The principle behind wavelet transforms is to transform from a time domain to a frequency domain and back. Discreatised wavelet transforms are used in (but not limited to) image and audio compression. The JPEG2000 and DjVu formats are just 2 of the many uses in image formats. The ability to decompress a large image progressively in the JPEG2000 standard illustrates why these techniques may be useful [10, ch 5][11].

In image compression lossless compression can be achieved by combining the discrete wavelet transform known as a Haar transform with RLE and Huffman coding. The Haar transform consists of calculating the averages and differences between pairs of samples. The wavelet transform has smaller numbers compared to the values of the original pixels and can therefore be encoded using RLE and Huffman coding [10, ch 5]. Tao et al. [12] points out that lossless compression of large collections of scientific floating point data is possible through wavelet transform techniques. However, they are mostly confined to cases where progressive transmission between coarse grained and high resolution versions is needed (such as approximate visualizations for large volumes of data and other hierarchical approaches).

6. PREDICTIVE METHODS

It is duly noted that a lot of research is being done on predictive methods and that only a few examples of these techniques are presented. They are context dependent - particularly on the structure of the input data. Some techniques developed by Engelson et al., Ratanaworabhan et al. and Lindstorm et al. are discussed.

As Engelson et al. [5] point out: general compression technologies such as text, image and signal compression tools are not intended for use on numerical data of time series. There is an opportunity for significant savings, when nearby elements in a time series change only in the least significant bits. Their proposed algorithm not only work on fixed time steps, but varying time steps as well. They employ a 1D extrapolation of previously received data. The algorithm shows effective compression rates when the input data is close to representing some polynomial.

Ratanaworabhan et al. [7] suggests a strategy for 64bit floating point compression in time-constrained environments. Their method does not assume that the consecutive values in a block of data differ only by a small amount as in [5]. Instead it identifies recurring difference patterns using a hash table lookup, which is then used to make a prediction (using a *dfcm* predictor) of the actual value. If the difference between the actual and predicted values are small then the XOR between the two values will contain a large number of leading zeros that can be compressed using a leading zero count. In particular the sign bits, exponent bits and the first few significant bits of the mantissa in such an XOR will be zero. A performance analysis (in terms of speed) of their compressor, (labeled dfcm) is shown in figure 1. It is interesting to note that their tailored algorithm is much faster than the standard LZ derivatives. In latter works [3] they presented an similar algorithm that achieves even higher throughput.

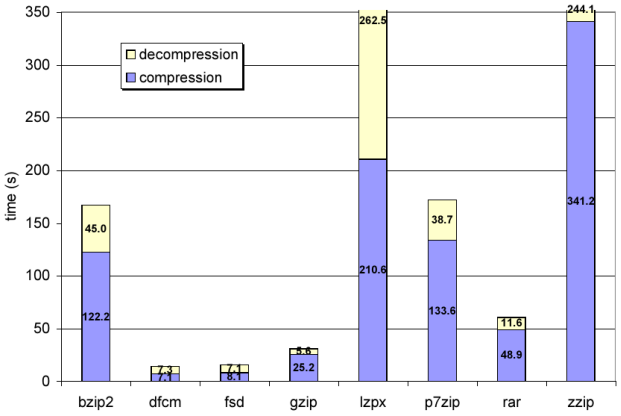

Figure 1: Compression/Decompression speed of Ratanaworabhan, Ke and Burtscher's algorithm (dfcm)[7]

Taking a different approach to prediction, Lindstrom and Isenburg proposes an online, fast and lossless technique for less structured data than that of Ratanworabhan et al. [6] The algorithm uses a *Lorenzo* predictor and partitions residuals into entropy codes. The encoder achieves compression ratios comparable to those by Engelson et al., but at twice the speed with a simpler implementation. They achieve slightly slower compression speeds than those of Ratanaworabhan et al., but with significantly better compression ratios (figure 2).

7. DISCUSSION

In light of the techniques discussed in this literature survey it may be fruitful to try the following techniques: an LZ method such as LZW, predictive methods resembling those proposed by Ratanaworabhan et al. and RLE with variable length coding (if the data values are skewed towards the least significant bits). Those variable-length coding schemes, based on adaptive arithmetic coding, may be too processing intensive for our purposes.

8. REFERENCES

[1] Daniel Abadi, Samuel Madden, and Miguel Ferreira. Integrating compression and execution in column-oriented database systems. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 671–682. ACM, 2006.

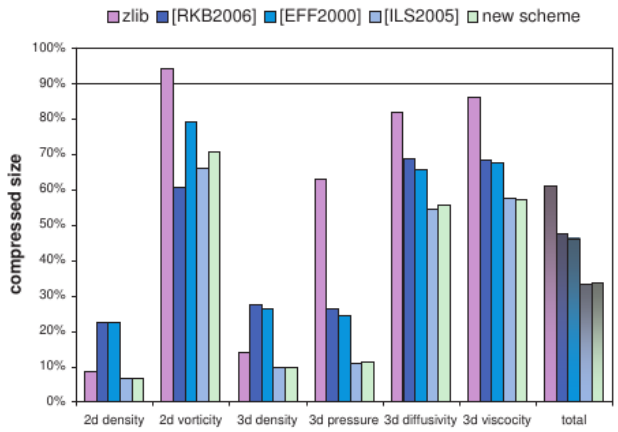

Figure 2: Compression size analysis from lindstrom et al. [6]

[2] M.A. Bassiouni. Data compression in scientific and statistical databases. *Software Engineering, IEEE Transactions on*, SE-11(10):1047–1058, 1985.

[3] M. Burtscher and P. Ratanaworabhan. Fpc: A high-speed compressor for double-precision floating-point data. *Computers, IEEE Transactions on*, 58(1):18–31, 2009.

[4] Adam Cannane and Hugh E. Williams. A general-purpose compression scheme for large collections. *ACM Trans. Inf. Syst.*, 20(3):329–355, July 2002.

[5] Vadim Engelson, Dag Fritzson, and Peter Fritzson. Lossless compression of high-volume numerical data from simulations. In *Data Compression Conference*, pages 574–586. Citeseer, 2000.

[6] Peter Lindstrom and Martin Isenburg. Fast and efficient compression of floating-point data. *Visualization and Computer Graphics, IEEE Transactions on*, 12(5):1245–1250, 2006.

[7] P. Ratanaworabhan, J. Ke, and M. Burtscher. Fast lossless compression of scientific floating-point data. In *Data Compression Conference, 2006. DCC 2006. Proceedings*, pages 133–142, 2006.

[8] Gautam Ray, Jayant R Haritsa, and S Seshadri. Database compression: A performance enhancement tool. In *International Conference on Management of Data*, page 4, 1995.

[9] Mark A. Roth and Scott J. Van Horn. Database compression. *SIGMOD Rec.*, 22(3):31–39, September 1993.

[10] D. Salomon. *Data Compression.: The Complete Reference.* Springer-Verlag New York Incorporated, 2004.

[11] A. Skodras, C. Christopoulos, and T. Ebrahimi. The jpeg 2000 still image compression standard. *Signal Processing Magazine, IEEE*, 18(5):36–58, 2001.

[12] Hai Tao and Robert J. Moorhead. Progressive transmission of scientific data using biorthogonal wavelet transform. In *Proceedings of the conference on Visualization '94*, VIS '94, pages 93–99, Los Alamitos, CA, USA, 1994. IEEE Computer Society Press.

[13] Hugh E Williams and Justin Zobel. Compressing integers for fast file access. *The Computer Journal*, 42(3):193–201, 1999.

[14] Ian H. Witten, Radford M. Neal, and John G. Cleary. Arithmetic coding for data compression. *Commun. ACM*, 30(6):520–540, June 1987.