Evaluation of Basic Data Compression Algorithms in a Distributed Environment

Minhaj Ahmad Khan*

Department of Computer Science, Bahauddin Zakariya Univ. Multan, Pakistan

Abstract: Data compression methods aim at reducing the data size in order to make data transfer more efficient. To accomplish data compression, the basic algorithms such as Huffman, Lempel-Ziv (LZ), Shannon-Fano (SF) and Run-Length Encoding (RLE) are widely used. Most of the applications incorporate different variants of these algorithms.

This paper analyzes the execution times, compression ratio and efficiency of compression methods in a client-server distributed environment. The data from a client is distributed to multiple processors/servers, subsequently compressed by the servers at remote locations, and sent back to the client. Our experimentation has been carried out using Simgrid Framework. Our results show that the LZ algorithm attains better efficiency/scalability and compression ratio, however, it works slower than other algorithms.

Keywords: Distributed Computing, Compression, LZ, Shannon-Fano, RLE, Huffman, Scalability.

1. INTRODUCTION

Distributed computing environments make use of multiple computing resources simultaneously. The resources exist at various locations and are interconnected through high speed networks. A large application with inherent parallelism may be distributed among the computing resources in order to achieve better performance. The speedup or the performance improvement thus achieved mainly depends upon the amount of parallelism. Data compression that aims at reducing the size of data also represents a problem that can be parallelized.

Data compression techniques [1-6] are widely used for various applications including graphics and audio/video. The compressor/de-compressor combination termed as a CODEC reduces the data size using compression and is able to generate original data using de-compression. Lossless compression algorithms generate compressed data in such a form so that the original input data can be later retrieved with no distortion. In contrast, the lossy compression methods discard some of data considered redundant in producing the original or near to original data during de-compression. The lossless compression techniques are therefore suitable for input data in text form, whereas the lossy compression methods are useful for input data in multimedia form such as audio and video [3, 4, 7, 8].

Most of the compression algorithms are variants of the basic compression algorithms including the Huffman [1], Lempel-Ziv (LZ) [5], Shannon-Fano (SF) [6] and Run-Length Encoding (RLE) [8]. These algorithms are normally evaluated using the metric of compression ratio that represents the ratio of the data size after compression to the data size before compression [3, 4]. Similarly, the time taken for compressing data is also of utmost importance especially when working in a real-time environment.

To increase performance, the compression algorithms can be parallelized by distributing input data among several processors with each processor operating on a small part of input data. Consequently, an improved performance is obtained in the distributed environment. For evaluating the performance of a parallel algorithm, the metric *speedup* refers to the ratio of time for sequential execution to the time for parallel execution. Another metric termed as *efficiency* refers to the ratio of speedup to the number of processors. Algorithms with an increase in *efficiency* corresponding to the increase in the number of processors are considered as scalable algorithms [9-11]. For a distributed computing environment, an efficient algorithm is therefore able to exploit the resources in an effective way and must require a small execution time.

This paper performs a comparative analysis of four compression algorithms: Huffman, Lempel-Ziv (LZ77), Shannon-Fano (SF) and Run-Length Encoding (RLE). The algorithms are executed for a distributed environment where the systems are connected *via* a high speed network. A client sends compression requests to multiple servers that are homogenous systems placed at remote locations. The data is compressed by each server and then sent back to the

*Address corresponding to this author at the Department of Computer Science, Bahauddin Zakariya Univ. Multan, Pakistan; E-mail: mik@bzu.edu.pk client. We perform experimentation using GRAS interface of the Simgrid framework [12] and analyze results for execution time, *efficiency* and compression ratio. The rest of the paper is organized as follows. Section 2 provides a succinct description of the working of the considered compression algorithms. Section 3 describes the configurations and the setup used for experimentation. The results are presented and analyzed in Section 4 followed by the conclusion in Section 5.

2. COMPRESSION ALGORITHMS

We consider four data compression algorithms: Huffman, LZ, Shannon-Fano and RLE. The working and implementation details of these algorithms are elaborated as follows.

2.1. Huffman Algorithm

The Huffman algorithm [1] is a lossless data compression technique that makes use of binary tree in a bottom-up approach. Every letter is a part of a tree and the frequency of occurrence of letters in the input data is used to define the ordering. The letters having least frequencies are combined to form a larger tree whose frequency is computed as the sum of the frequencies of the child nodes. The binary tree then continues to grow with the new trees being added at sorted locations. Consequently, a single tree is formed as a combination of different trees and is termed as Huffman tree.

In contrast to other algorithms, the Huffman algorithm is considered to be efficient for the cases where the data is noisy [13]. Furthermore, the optimal code lengths are used by the algorithm when the symbols not appearing in the data are skipped.

2.2. Lempel-Ziv (LZ) Algorithm

The LZ algorithm [5] is a lossless data compression technique similar to the Huffman coding, however it incorporates a sliding window during compression. The input string is parsed to form blocks that are encoded. Subsequent blocks are encoded by reference to previous blocks. The encoding is represented by a tuple having length and offset as its members. The length describes the number of characters that are equal to the characters at the specified offset in the input data. To match occurrences of data, a buffer of data is maintained that is called a sliding window.

In general, due to string matching at each step, the LZ compression technique is considered to be very slow. However, its good compression ratio and straightforward implementation make it a widely used compression technique.

2.3. Shannon-Fano (SF) Algorithm

The SF algorithm [6] is also a lossless data compression technique very similar to the Huffman coding, however, in contrast to the Huffman coding, it is not able to generate an optimal code length. In this coding technique, the symbols are initially arranged in order of their probabilities thereby generating two sets of symbols. The code assignment for the input symbols starts by assigning a '1' or '0' to the symbols and subsequent codes are generated by recursively dividing symbols in the sets into subgroups and adding bits to the codes for the symbols.

In general, the SF algorithm is very easy to implement, however, the generation of suboptimal code lengths makes is less widely used when compared with the Huffman coding technique.

2.4. Run-Length Encoding (RLE)

The RLE algorithm [8] is also a lossless data compression technique and is considered to be the simplest in that the repeated occurrences of a character are replaced by the length of the character followed by the character itself. It is also used in the Graphics Interchange Format (GIF) images.

As RLE works very well for the data in which a character has many repeated occurrences, it is widely used for simple images that use a limited set of colours.

3. EXPERIMENTAL SETUP AND CONFIGURATION

For evaluating the performance of the compression algorithms, we used the Grid Reality and Simulation (GRAS) interface of the Simgrid Framework [12]. The GRAS interface enables to write distributed client and server code. Two configuration files specifying the hardware configurations for the client and the servers are input to the Simgrid Framework that simulates the code on the specified hardware. We have used a diverse number of servers (processors) 1,2,4,8,16 for each run of the implementation code. The clients and servers are connected using star topology *via* a high speed network operating at 1GBps. The client is configured to support a maximum of 30GFlops, whereas each server is configured to support a maximum of 50GFlops.

Table 1. The Benchmarks and the Sizes of Input Data Used for Compression

Benchmark	Size (~MB)	Benchmark	Size (~MB)	Benchmark	Size (~MB)	Benchmark	Size (~MB)
Automake	0.2	evince	0.5	assistant-qt4	1	smbspool	2
Ccmake	3	rpcclient	4	Doxygen	5	fdebuginfo	7
lidb	15	Pdf File 1	20	Pdf File 2	40	Pdf File 3	60
Pdf File 4	80	Pdf File 5	100	Pdf File 6	200	Pdf File 7	500

Each scenario implements a distributed environment with a client sending data to server(s) with each server being capable of executing the compression algorithm and returning result to the client. On the server side, we have used the compression algorithm implementations from the BCL library [13] written in C language. We have used the binary utilities/programs existing on a linux-based systems as described in Table **1**. We also used a large sized Pdf file (~20MB) and replicated its data to generate files of larger size.

4. PERFORMANCE RESULTS

Figure **1** shows the results obtained for the compression ratio for all the algorithms using a single processor. The results of compression ratio are almost the same while using multiple processors. This is due to the fact that in case of multiple processors, each server operates on a small part of data, and consequently, the main compressed data returned to the client by multiple servers is the same as obtained using a single processor.

It is evident from the results that the LZ compression algorithm achieves the best compression ratio which implies that it produces the minimum data size of compressed data. On average, the LZ compression algorithm performs 31.55%, 31.79% and 40.75% better than the Huffman, SF and RLE compression algorithms respectively.

Figure **2** shows the average execution time in seconds for each algorithm using different number of servers (1,2,4,8 & 16). The addition of multiple servers makes the code execute faster significantly in comparison with the single server system. Corresponding to all the processors, the Huffman, LZ, SF and RLE algorithm have average execution times of 0.76, 2.63, 0.75 and 0.42 seconds respectively. The RLE algorithm therefore performs better than all other algorithms. The complex buffer manipulation by the LZ algorithm makes it execute slower than other algorithms.

Figure **3** shows the *efficiency* results obtained for all the algorithms using different number of servers. The *efficiency* increases with two servers, however there is a gradual decrease in *efficiency* as the number of servers grows. In terms of *efficiency*, the largest impact is produced by the LZ compression algorithm which implies that the algorithm is scalable and is able to exploit additional resources effectively in comparison with other algorithms.

5. CONCLUSION AND FUTURE WORK

We compare four data compression algorithms Huffman, LZ, SF and RLE in terms of the compression

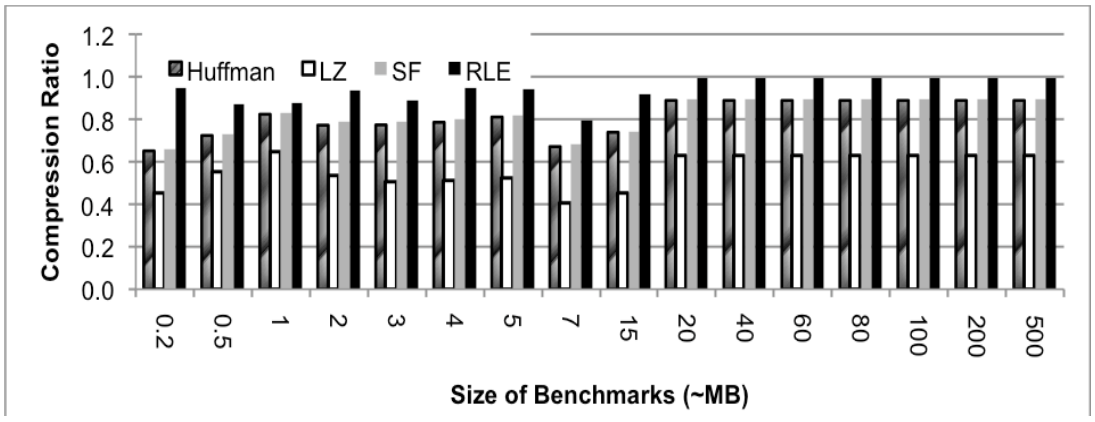

Figure 1: Compression Ratio for benchmarks of different sizes using the compression algorithms.

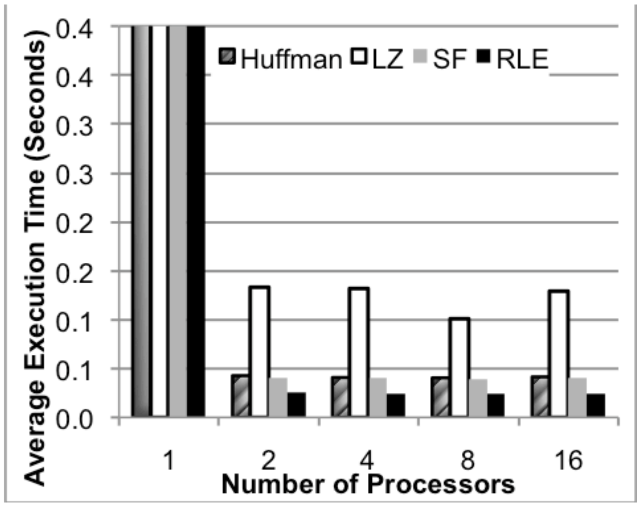

Figure 2: Execution times for benchmarks of different sizes corresponding to different number of processors.

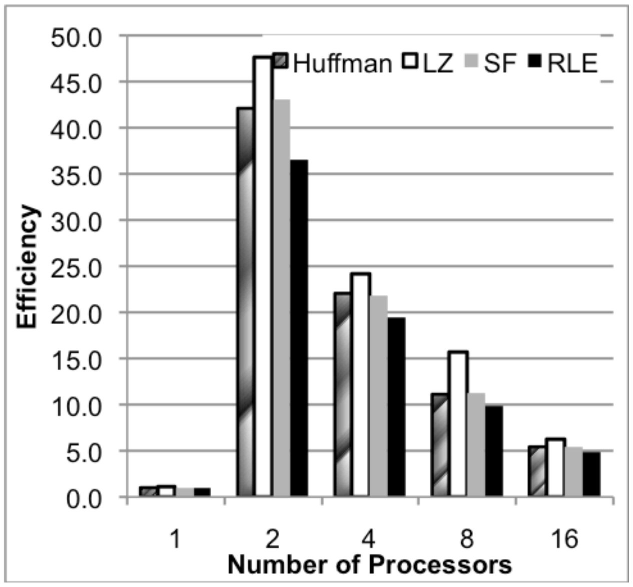

Figure 3: Efficiency results for benchmarks of different sizes corresponding to different number of processors.

ratio, execution time and the *efficiency*. We used as benchmarks several binary utilities in Linux and different Pdf files. The algorithms were set to execute in a client-server based distributed environment with variant number of servers (1, 2, 4, 8 & 16) using Simgrid Framework.

For our scenarios, the LZ algorithm has the best compression ratio, however it works with a large overhead of memory buffer processing thereby executing slower than other techniques. In terms of *efficiency* or scalability the LZ algorithm is better than all other algorithms and reduces significantly its execution time when new servers are added to the system. As future work, we intend to analyze the *efficiency* results on large multi-core systems and modify the LZ algorithm to adapt to the underlying architecture.

REFERENCES

[1] Huffman DA. A Method for the Construction of Minimum-Redundancy Codes. Proceedings of the I.R.E. Sep. 1952; 1098-102.

[2] Vitter JS. Design and Analysis of Dynamic Huffman Codes. J ACM 1987; 34(4): 825-45. http://dx.doi.org/10.1145/31846.42227

[3] Blelloch GE. Introduction to Data Compression. Carnegie Mellon Univ., 2010.

[4] Cormen TH, Stein C, Rivest RL, Leiserson CE. Introduction to Algorithms (2nd ed.). McGraw-Hill Higher Education 2001.

[5] Shannon CE. A Mathematical Theory of Communication. Bell Syst Tech J 1948; 27: 379-23.

[6] Ziv J, Lempel A. A universal Algorithm for Sequential Data Compression. IEEE Trans Inform Theory 1977; 23(3): 337-43.

[7] Ziv J, Lempel A. Compression of Individual Sequences *via* Variable-Rate Coding. IEEE Trans Inform Theory 1978; 24(5): 530-36.

[8] Leurs L. RLE Compression. http://www.prepressure.com/library/compression_algorithms/rle, 2012.

[9] Bondi AB. Characteristics of scalability and their impact on performance. Proceedings of the 2nd international workshop on Software and performance, Ottawa Canada, 2000; pp. 195-203.

[10] Fox G, Johnson M, Lyzenga G, Otto S, Salmon J, Walker D. Solving Problems on Concurrent Processors. Prentice-Hall, Englewood Cliffs, NJ 1988; Vol. 1.

[11] Kumar V, Grama A, Gupta A, Karypis G. Introduction to Parallel Computing - Design and Analysis of Algorithms. The Benjamin/Cummings Publishing Company, Inc., Redwood City, CA, 1994.

[12] Casanova H, Legrand A, Quinson M. SimGrid: a Generic Framework for Large-Scale Distributed Experimentations. Proceedings of the 10th IEEE International Conference on Computer Modelling and Simulation (UKSIM/EUROSIM'08), 2008.

[13] Geelnard M. Basic Compression Library. 2006; Available: http://bcl.comli.eu/.

Received on 12-06-2012 Accepted on 30-06-2012 Published on 04-07-2012

http://dx.doi.org/10.6000/1927-5129.2012.08.02.18