Lossless Compression Techniques in Edge Computing for Mission-Critical Applications in the IoT

T. N. Gia[2], L. Qingqing[2], J. Peña Queralta[2], H. Tenhunen[2], Z. Zou[1] and T. Westerlund[2]

[1] School of Information Science and Technology, Fudan Universtiy, China
[2] Turku Intelligent Embedded and Robotic Systems, University of Turku, Finland
Emails: [1]zhuo@fudan.edu.cn, [2]{qingqli, tunggi, jopequ, hatenhu, tovewe}@utu.fi

***Abstract*—The need of data compression at smart Edge/Fog-based gateways is undeniable as data compression can significantly reduce the amount of data that has to be transmitted over a network. This, in turn, has a direct impact on reducing transmission latency and increasing network bandwidth. In time-critical and data sensitive IoT applications such as healthcare, lossless data compression is preferable as compressed data can be recovered without losing any information. However, it is not an easy task to choose a proper lossless data compression algorithm for IoT applications as each lossless data compression method has its own advantages and disadvantages. This paper focuses on the analysis of lossless data compression algorithms run at smart Edge/Fog gateways. Widely used lossless data compression are run at different hardware which is often used as smart Fog/Edge gateways. The latency of data compression and compression rate in different cases of input data sizes are analyzed. The paper provides guidelines for choosing a proper lossless data compression algorithm for time-critical IoT applications.**

***Index Terms*—Edge, Fog, gateways, IoT, data compression**

I. Introduction

In most IoT applications, sensor nodes collect and send data to a gateway over a wireless network. Then, the gateway forwards the data to Cloud servers via the Internet. This traditional cloud-centric IoT architectures might cause some disadvantages such as high latency when the number of sensor nodes or the volume of data is dramatically high. In some IoT applications based on low-power and low data-rate technologies as LoRa, a significant negative impact on system performance can be expected [1]. In order to reduce the latency, save network bandwidth and avoid problems of transmitting a large amount of data over the network, data can be compressed at Edge or Fog gateways before being sent to Cloud servers [2]. Two basic types of data compression, namely lossless and lossy data compression, can be applied in IoT applications. Lossless data compression algorithms can recover compressed data without losing any information whilst lossy algorithms are not. Therefore, lossless data compression algorithms are preferable, in particular, for data-sensitive and mission-critical IoT applications such as healthcare systems. However, lossless compression methods often provide a low compression rate such as 5:1 or 10:1 which is much smaller than lossy data compression rate i.e., 50:1 or 100:1. Different types of lossless data compression methods have been proposed for IoT applications. However, not all of them are suitable for real time IoT applications. Therefore, in this paper, widely used lossless data compression algorithms are experimented at different hardware which is often used as a smart Fog/Edge gateway. Their compression rate and latency are reported and analyzed. In addition, this paper will give a hint for a system administrator when choosing a suitable lossless data compression method for their Edge/Fog gateways.

TABLE I
Gateway hardware specification

	CPU - Core	Clock	RAM
Rasberry Pi 3Bv2 (Pi)	ARM A53 - 4	1.4 Ghz	1 Gb
Intel UP (I-UP)	ATOM x5-Z8350 - 4	1.92 Ghz	2 Gb
UP Gateway (I-UPG)	i3-8145UE - 4	3.9 Ghz	16 Gb
Intel i5(I-i5)	i5-6200U - 4	2.3 Ghz	6 Gb

TABLE II
LZO's compression latency and rate

Data size	200B	2kB	10kB	20kB
Pi	-	0.158 ms	0.185 ms	0.218 ms
I-UP	-	0.098 ms	0.13 ms	0.156 ms
I-UPG	-	0.208 ms	0.228 ms	0.234 ms
I-i5	-	0.119 ms	0.126 ms	0.136 ms
Compression rate	NA	(1.334)	(6.455)	(12.58)

II. Lossless Data Compression Algorithms

Due to the scope of this paper, only widely used lossless compression algorithms for general purpose data such as LZO [3], LZ4 [4], LZW [5], Huffman coding [6], and Zstandard [7] are discussed. LZO is able to provide a high data compression rate while it does not require a large latency to compress and decompress data. The algorithm can be considered as a solution for time critical IoT application. Besides the source and destination buffers, a decompress process does not need an additional memory.

LZ4 is a fast lossless compression algorithm which is able to achieve a compression speed higher than 500 MB/s per core. LZ4 also has a fast decompression speed up to a few gigabytes per second per core. LZ4 is a simple algorithm with a reasonable compression rate. LZW is a fast lossless data compression algorithm using a table-based lookup method. LZW works more efficiently if the data file contains a lot of repetitive data. Huffman coding is a lossless data compression algorithm developed by David Huffman. The algorithm is based on a binary-tree frequency sorting technique. Zstandard is a real-time lossless data compression algorithm which is also

TABLE III
LZ4'S COMPRESSION LATENCY AND RATE

Data size	**200B**	**2kB**	**10kB**	**20kB**
Pi	0.56 ms	0.84 ms	0.98 ms	1.11 ms
I-UP	0.34 ms	0.49 ms	0.65 ms	0.78 ms
I-UPG	0.61 ms	0.916 ms	1.106 ms	1.298 ms
I-i5	0.6 ms	0.32 ms	0.788 ms	1.136 ms
Compression rate	(0.985)	(1.414)	(6.455)	(11.59)

TABLE IV
LZW'S COMPRESSION LATENCY AND RATE

Data size	**200B**	**2kB**	**10kB**	**20kB**
Pi	6.24 ms	32.12 ms	126.84 ms	234.73 ms
I-UP	1.87 ms	11.89 ms	47.73 ms	90.84 ms
I-UPG	1.72 ms	8.44 ms	34.04 ms	62.54 ms
I-i5	1.535 ms	5.831 ms	16.37 ms	24.71 ms
Encoded symbols	144	944	3028	4860

TABLE V
HUFFMAN CODING'S COMPRESSION LATENCY AND RATE

Data size	**200B**	**2kB**	**10kB**	**20kB**
Pi	0.98 ms	1.386 ms	2.147 ms	2.399 ms
I-UP	0.856	2.624 ms	4.871 ms	11.71 ms
I-UPG	0.81 ms	1.419 ms	4.64 ms	8.56 ms
I-i5	0.35 ms	1.024 ms	4.27 ms	7.561 ms
Compression rate	(0.836)	(1.596)	(1.823)	(1.856)

TABLE VI
ZSTANDARD'S COMPRESSION LATENCY AND RATE

Data size	**200B**	**2kB**	**10kB**	**20kB**
Pi	1.15 ms	1.39 ms	1.85 ms	1.89 ms
I-UP	0.479 ms	1.402 ms	1.916 ms	1.9-2.3 ms
I-UPG	0.941 ms	1.386 ms	2.147 ms	2.399 ms
I-i5	0.824 ms	1.28 ms	1.809 ms	2.034 ms
Compression rate	(1.48)	(2.209)	(10.91)	(21.78)

able to provide a high decompression speed. The algorithm allows to have trade-off between compression speed and compression ratios.

III. EXPERIMENT AND RESULTS

Widely used lossless data compression algorithms including LZO, LZ4, LZW, Huffman coding and Zstandard are implemented in different hardware (e.g., Rasberry Pi 3B, Intel Up, Intel UP Gateway and Intel i5-based board) which are often used as a smart Fog/Edge gateway in IoT applications. The detail information of these boards are shown in Table I.

In an IoT application such as ECG monitoring or co-robot monitoring, each sensor node can collect data with different sizes from a few bytes to a few kilobytes depending on an application. In addition, depending on the number of sensor nodes, an amount of data received at a smart Fog/Edge varies. Therefore, different data sizes are applied in the experiments. Results are shown in Tables II, III, IV, V and VI.

Table II shows that LZO cannot work properly when data input size is small. When the data volume increases, the LZO algorithm can achieve a higher compression rate e.g., 12 times in cases of compressing 20kB data. In addition, the LZO algorithm does not require a large latency in terms of a millisecond for compressing data. Table III shows that LZ4 cannot work efficiently when data input size is too small e.g., 200B in the experiment. When input size increases, the LZ4 algorithm can achieve a higher data compression rate. Similar to the LZO algorithm, LZ4 is a fast algorithm and it does not require a large latency in terms of a millisecond to compress data. Both in the case of LZ4 and LZ0, we have reduced the data entropy in order to simulate more accurately a real application where there are few oscillations in the measured variables. Table IV show that LZW requires a large latency to compress data. The LZW algorithm relies on a dictionary to compress data. Therefore, it works efficiently when data size is large. Table V shows that the Huffman coding cannot provide a high compression rate even though when data size increases dramatically. Table VI shows that the Zstandard algorithm can work more efficiently when data input size increases if there is lower entropy in the data. In addition, the algorithm can achieve a high compress rate (e.g., 21.78 times) in case of compressing 20kB data. The algorithm is able to compress a large amount of data with a low latency in terms of a few milliseconds. In experimented algorithms, LZO, LZ4 and Zstandard algorithms demonstrate their advantages as they are able to provide a high compress rate with a low data compression latency. These algorithms prove that they are suitable for time critical IoT applications.

IV. CONCLUSION AND FUTURE WORK

This paper provides an analysis of widely used lossless data compression algorithms in IoT applications. The paper reviews widely used lossless data compression algorithms on different hardware platforms, which are often used as a smart Fog/Edge gateways in IoT applications. The results showed that LZO, LZ4, and Zstandard are suitable to provide high performance with a high compression rate and low latency. We defend that these algorithms are suitable for time-critical IoT applications such as ECG, EEG monitoring.

ACKNOWLEDGMENT

This work has been supported by NFSC grant No. 61876039, and the Shanghai Platform for Neuromorphic and AI Chip (NeuHeilium).

REFERENCES

[1] T. N. Gia *et al.* Edge AI in Smart Farming IoT: CNNs at the Edge and Fog Computing with LoRa. In *IEEE AFRICON*, 2019.
[2] J. Peña Queralta *et al.* Edge-AI in LoRabased healthcare monitoring: A case study on fall detection system with LSTM Recurrent Neural Networks. In *42nd TSP*, 2019.
[3] Oberhumer. LZO. Available: www.oberhumer.com/opensource/lzo/. Accessed: Aug 2019, Updated: 2017.
[4] LZ4. Available: lz4.github.io/lz4/. Accessed: Aug 2019.
[5] LZW compression. Available: rosettacode.org/wiki/LZW˙compression. Accessed: Aug 2019.
[6] Huffman coding. cs.duke.edu/csed/poop/huff/info/. Accessed: Aug 2019.
[7] Zstandard. Available: facebook.github.io/zstd/. Accessed: Aug 2019.