Compound Compression Method for Gathering Traffic of IoT/CPS Data

Kazuhito MATSUDA
FUJITSU LABORATORIES LTD.
Kawasaki, Japan
m.kazuhito@fujitsu.com

Makoto KUBOTA
FUJITSU LABORATORIES LTD.
Kawasaki, Japan
kubota.makoto@fujitsu.com

***Abstract*—These days, many kinds of sensors are connected to the Internet. They measure the real world from multiple aspects, and the measured data is used to optimize commercial activity and social infrastructure. This trend is called IoT/CPS, which stands for Internet of Things and Cyber Physical Systems. The data are divided into two types: "heavy data" such as multimedia data and "light data" such as numerical values and strings. Demand exists to gather light data to data centers; however, traffic volume becomes a challenge.**

In this paper, we propose an efficient compression method targeting light data. The proposed method utilizes the structure of light data, which have "constant," "variable," and "timestamp" fields, to create an efficient dictionary of symbols and perform column-based compression. By evaluating multiple kinds of light data sources, the proposed method can improve the compression ratio by 24% compared with a zip (Deflate) compression and achieve traffic reduction.

***Index Terms*—Internet of Things, cyber-physical system, data compression**

I. Introduction

These days, many kinds of sensors are connected to the Internet directly or through gateways. They measure the real world from multiple aspects, and that measured data is then used to actuate machines and execute analysis to optimize commercial activity (e.g. Industry 4.0 [1]) and social infrastructure (e.g. Smart City [2], [3]), which is known as Internet of Things (IoT) [4] and Cyber Physical System (CPS) [5].

In IoT/CPS, sensors to measure the real world are spread over wide geographical areas. Although these sensors generate large amounts of data, it is not reasonable to gather all data in data centers because of the large traffic volume. Cisco predicts that traffic volume into data centers will become over 20 Zeta Bytes by 2021 [6]. For a single data gathering system, a number of sensors might be on the order of a million. In addition, when making actuations to the real world with the measured data, the propagation delay becomes critical. Edge computing [7] is one solution to these problems which works by deploying computers at the edge of networks (e.g. multi-access edge computing [8]) and executing some processing tasks such as filtering, compressing, and sending responses to actuators (Fig. 1).

Data generated by sensors is divided into two types. One is large size data called *heavy data* such as pictures, videos, audios, and depth point clouds. The other is *light data* such as geo-coordinates (e.g. GPS), acceleration/temperature/humidity with low sampling rates, the boolean of event detection (e.g. sudden acceleration or breaking of a vehicle), and character strings. Light data is used for statistics and machine learning, for which it has to be gathered in a data center. It is difficult to gather all heavy data because of its size and the bandwidth between the edge and the data center. The data users can only gather locations and metadata and pull or process it when it is actually needed. In this case, the locations and metadata can also be light data. Indeed, light data is actually light-weight, but the traffic volume to gather it can be challenging to handle.

Deflate [9] is a popular compression algorithm for light data (e.g. text or numerical value). However, Deflate, which includes word dictionary for compression, is not efficient because light data in IoT/CPS is generated continuously, and the amount of data per unit time at each edge computer is small. Even if the word dictionary uses multiple groups of entries (called *chunks* in the remaining of the paper), Huffman coding in Deflate is not efficient because the distribution of words in each chunk differs from the others.

In this paper, we propose a novel compression method in a system consisting of a number of edge computers and a data center (Fig. 1), which can be employed to gather light data from edge computers to data centers. We first focus on the data structure of the light data, which has *constant* fields that do not change over time, *variable* fields which change over time, and *timestamp* fields. For the constant fields, the proposed method releases integer symbols immediately and synchronizes the dictionary of the original values and symbols among the edge computers and the data centers. After finishing the synchronization, when the same constant fields appear, they are replaced with the symbols. For variable fields, the proposed method records the frequencies of appearance and releases symbols to the values of the fields that have relatively high frequency. After replacement with the symbols, the constant fields and each variable field are encoded with an algorithm for sequence of integers (i.e. variable byte code), considering each symbolized field as a column. The proposed method continuously adjusts symbol assignments for constant fields as to be close when they have a high probability of appearing in the same chunk. By this adjustment, the average byte length of column compression is made shorter. Note that

978-1-5386-4980-0/19/$31.00

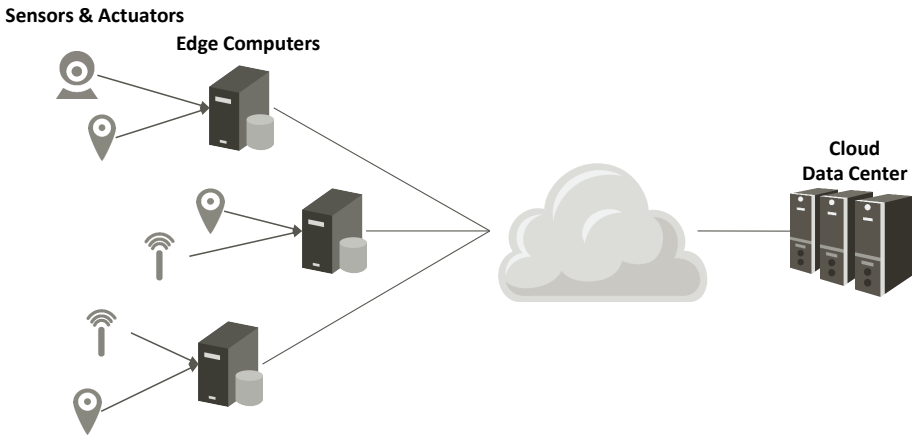

Fig. 1. Edge computing

the proposed method chooses symbols for variable fields to be close to the symbols for the constant fields, then the column compression for the variable field is also efficient.

In a simulation assuming multiple types of light data sources, we evaluated the proposed method and showed that it can improve the compression ratio by 24% in a certain situation compared with the general implementation of Deflate (i.e. zlib).

The remaining of the paper is organized as follows. Section II shows the background of the research and our motivation. In Section III, we explain the proposed method in detail, and in Section IV we present the evaluation results and discussion. We conclude this paper with Section V.

II. Background

As mentioned in Section I, data generated in IoT/CPS can be divided into heavy data (e.g. multi-media data) and light data (e.g. text, numerical, boolean). In this paper, we focus on light data, including locations and meta-data of heavy data. Light data usually has three kinds of fields: 1) *constant* fields which are assigned to each data source uniquely and do not change over time, 2) *variable* fields which change over time, and 3) *timestamp* listed in Table I. The proposed method utilizes this data structure.

Deflate [9] is one of the most popular compression algorithms. In Deflate, a word dictionary is created first by scanning the target data. Variable length symbols are then assigned according to the appearance frequencies (i.e. Huffman coding). However, Deflate is not suitable for light data in IoT/CPS for the following reasons:

- Light data in IoT/CPS is continuously generated and transmitted from the edge computers to the data centers in very short time units (e.g. few seconds) due to demands for real-time analysis. Therefore, the chunk size that is a unit of compression at one time is relatively smaller compared to its word dictionary size.
- Because each data source sends light data periodically, constant fields of a certain pair of data sources appear in the same chunk with high probability. On the other hand, constant fields of a single data source do not appear in a single chunk. Therefore, constant fields cannot be handled as a single word in the word dictionary for the chunk.
- Even if we use the word dictionary over multiple chunks, the frequency distributions of word appearances vary from one chunk to another. Therefore, Huffman coding is inefficient.

Some coding algorithms can reduce the byte length of symbols that do not depend on appearance frequencies, but they can only be applied to sequences of numerical numbers. For example, variable byte code (VBC) is a coding algorithm for sequences of integers. VBC first makes a *gap list*, which is a sequence of differences between two adjoining integers. After that, each gap integer is encoded with a 7 bit payload and a continuation bit which represents that the integer continues to the next byte. It means, for instance, that 0 to 127 can be expressed by 1 byte. However, these algorithms can only apply to columns of restricted variable types.

As described above, not all compression algorithms are suitable for light data in IoT/CPS. Therefore, we invented a novel compound compression method for light data described in Section III.

III. Compound compression method

A. Overview

The proposed method attempts to overcome the weaknesses of Deflate by using the data structure of light data, by maintaining the dictionary between the edge computers and the data center, and by using the VBC encoding. Fig. 2 shows a simple flowchart of the proposed method. In this section, we describe the details of each box in Fig. 2.

Note that the proposed method targets the data which have the structure depicted in Table I, i.e. constant, variable, and timestamp fields and each of the constant/variable fields has more than one key-value. The key is written with arbitrary characters, and the type of value is integer, float, string, or

TABLE I
STRUCTURE OF LIGHT DATA

type	examples	
constant	device unique information	device id, device type
	location data (non-movable device)	GPS/UTM coordinates
variable	sensing data	measured data by each kind of sensors
	location data (movable device)	GPS/UTM coordinates
timestamp	-	-

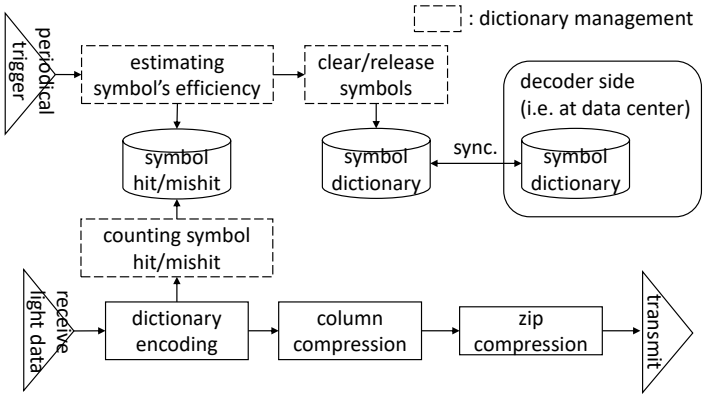

Fig. 2. Procedure overview

TABLE II
SYMBOL DICT. FOR CONSTANT FIELDS

hash	symbol
abcdefg	1
hijklmn	2
...	...

TABLE III
SYMBOL DICT. FOR VARIABLE FIELDS

hash	symbol
nopqrst	3
vwyzab	5
...	...

boolean. We assume that the data provider specifies whether the field is constant or variable.

B. Dictionary encoding

For constant fields, when a light data entry is received, considering whole constant fields of the entry as a single word, the proposed method searches the symbol dictionary for constant fields (Fig. II). Note that the fields convert into the hash value for simplicity. When the word is found in the dictionary, it is replaced with the corresponding integer symbol. When it is not found, a new integer symbol is released for the word, and the pair of the word and the new symbol is appended to a waiting queue to be synchronized. The entries in the queue are periodically synchronized with the dictionary in the data center.

For variable fields, considering each field in an entry as a single word, the proposed method searches the symbol dictionary for variable fields (Table III). When a new symbol for the variable field is needed, a symbol is released with the same value or a value closer to the symbol of the constant field in the same entry. That is, if an entry has a constant field symbol s, the symbols from $k \times s$ to $k \times (s+1) - 1$ can be released to the variable fields. If there are no symbols that are not assigned in this range, no new symbols will be released. The replacement of the word with the released symbol and the synchronization of the dictionary are executed in the same way as for the constant fields.

Table IV is a sample entry applied the dictionary encoding. Note that if there are no available symbols (i.e. not assigned or not synchronized yet), the corresponding key-values are compressed by Deflate while maintaining the order of the entries in the chunk.

C. Column compression

After dictionary encoding, symbols of constant fields in a chunk are compressed by VBC as a column. For example, we assume a dictionary encoded chunk shown in Table V, which is sorted by its constant fields symbols. The integer sequence of constant field symbols of the chunk is $[1, 5, 31, 49, 52]$ and its gap list is $[1, 4, 26, 18, 3]$. If VBC is applied to this gap

TABLE IV
DICT. ENCODED SAMPLE

	original	hash	dict. encoded
constant	source_id: hoge geo: {lat, lng}	abcdefg	1
variable_1	temp: 25.5	nopqrst	3
variable_2	humidity: 70.5	qazwsxe	(none)
timestamp	11223344	-	-

list, it is represented by 5 bytes because any number less than $2^{7\times n}$ can be expressed by n bytes.

Symbols of variable fields are encoded in almost the same way as encoding of constant field symbols. However, the number of variable fields of an entry in the chunk is not always equal to all other entries in the chunk. When there are more than $th_v\%$ entries that have equal to or more than l symbolized variable field in the chunk, we apply VBC to the l column with *zero* padding, which means the field is empty.

Note that variable field symbols cannot be sorted because the entries in the chunk are already sorted by constant field symbols. Therefore, a negative integer can appear in the gap list. We applied a minor change to the original VBC by adding a bit that shows whether the integer is negative or not. In this slightly changed VBC, n bytes express a number from $-2^{7\times n-1}+1$ to $2^{7\times n-1}-1$. We utilize this changed VBC for encoding timestamp fields, too.

D. Dictionary management

As mentioned in Section I, the efficiency of VBC depends on the absolute values of numbers in the gap list. So, we calculate the probability of a certain pair of constant field symbols appearing in the same chunk by observation, then reassign symbols as pairs that appear with high probability have closer symbols than low probability pairs. This simple clustering principle can be effective because the data sources in IoT/CPS transmit light data periodically, or when similar events occur.

For variable field symbols, we estimate traffic reduction of each symbol and clear/release symbols based on the following estimation algorithm:

1) Record usage counts c_{hit} of symbols and mishit counts c_{miss} of variable fields whose symbols are not assigned.
2) Calculate the estimated traffic reduction r by using each symbol with Equation 1, where s_c and s_v are the symbols of the constant and variable fields, l is the symbolized column number explained in the previous subsection, and o is the original size of the variable field key-value:

$$\begin{aligned} b &= \min x \\ &\quad \{x|((s_c-1)\times l - s_v) < 2^{7\times x-1}\} \\ r &= (o-b)\times c_{hit} \end{aligned} \tag{1}$$

3) Periodically clear the symbols whose traffic reduction r is not included in th_{clear} percentile of all symbols for variable fields. Note that in order to suppress burst traffic of dictionary synchronization, we set additional parameter max_{clear}, which is the max number of symbols to release at once.
4) Release new symbols to the variable fields whose c_{miss} are on top of $max_{release}$-th.

After the symbols for constant fields are adjusted as mentioned earlier in this subsection, the r can be a metric to estimate traffic reduction. The parameters th_{clear}, max_{clear}, and $max_{release}$ affect the convergence speed of the dictionary and traffic volume of the dictionary synchronization, so they should be adjusted according to the actual bandwidth between the edge computers and the data center.

IV. Evaluation

A. Settings

For evaluation, we used three types of light data sources: 1) video camera, 2) environmental instruments, and 3) event detector based on movie and/or sensed value. These data sources have common fields: 1) *source_id*: identifier of data source, 2) *geo_lat* and *geo_lng*: geographical location by GPS, 3) *type*: of data source, and 4) *timestamp*. In addition to these common fields, each data source has fields unique for each type written in Table VI. The variable types and assumed distributions are shown in Table VII. Numbers of data sources per each type are 50, and frequencies of data generation of the video camera, environmental instruments, and the event detector are set to 60, 30, and 120 seconds, respectively.

According to Table VII, we generate light data for the evaluation with $3,600$ seconds time span. In the edge computer, the received light data is buffered once during a certain time $t_b = \{2, 5, 10\}$ seconds and compressed by the proposed method. After compression, it is sent to the data center. We also get the results for the case when only zip (Deflate) is used for compression. Although the main metric of the evaluation is traffic volume from the edge computer to the data center, we have also measured the byte length of symbols for constant/variable/timestamp fields and the remaining data that is compressed by zip.

B. Results

Figure 3 shows the traffic volume in the case when $t_b = 5$, where the plot points represent average bytes per second during 10 seconds. In the figure, we plot the results when compressing by the proposed method with or without dictionary synchronization traffic, and only by zip. From the figure, in the bootstrap phase, the traffic volume for dictionary synchronization becomes extremely high compared with the case when compressing is done only by zip. However, this burst traffic only occurs in the bootstrap phase and it can be controlled by the parameters shown in Subsection III-D. After 520 seconds, the proposed method can reduce the traffic volume by 45% compared with the "zip only" case. This indicates that the proposed method can achieve 24% higher compression ratio compared with the zip.

We plot the average byte length of VBC encoded symbols/timestamps and remaining fields compressed by zip in

TABLE V
CHUNK SORTED BY CONSTANT SYMBOL

constant symbol	1	5	31	49	52
variable symbols	1,2	5,6	31,32,33	49	52,53
timestamp	11223344	11223345	11223340	11223332	11223359
remaining fields	none	none	none	a_key: a_value	none

TABLE VI
UNIQUE FIELDS OF EACH SOURCE TYPES

type	unique variable fields
video camera	format, uri
environmental instruments	temperature, humidity
event detector	event

TABLE VII
TYPE AND DISTRIBUTION OF EACH FIELD

field name	variable type and value detail
source_id	string (UUIDv4)
geo_lat/geo_lng	float (num. of decimals = 6)
gen_time	integer (UNIX time)
format	string (”mpeg2”or ”h.264” or ”h.265”)
uri	string (random 30 characters)
temperature/humidity	float (num. of decimals = 2)
event	string (random 20 characters)

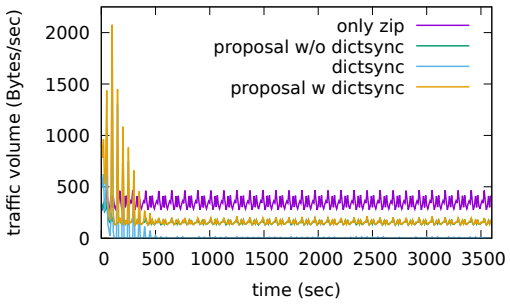

Fig. 3. Traffic volumes ($t_b = 5$)

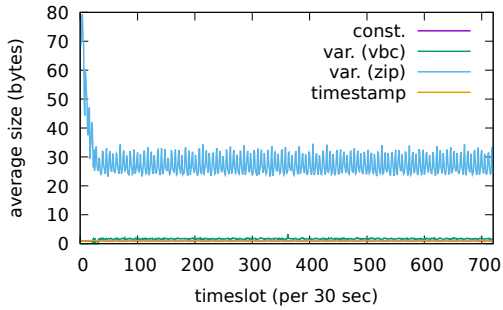

Fig. 4. Average data size of each field ($t_b = 5$)

TABLE VIII
COMPRESSION RATIO WITH VARIED BUFFERING TIME

buffering time t_b (sec)	2	5	10
zip	34%	56%	62%
proposal	63%	80%	85%

TABLE IX
AVERAGE TRAFFIC VOLUME OF EACH TYPE DATA SOURCE TYPE

type of data source	compression ratio	
	zip	proposal
video camera	24%	51%
environmental instruments	52%	78%
event detector	14%	43%

Figure 4. In the evaluation scenario, each data source periodically generates data; once the closer symbols are assigned to the constant fields, it is maintained during the evaluation. So, the length of the constant field symbols can be almost 1 byte (i.e. its gap is less than 128). In the case of timestamps, the length can also be 1 byte due to the buffering time (5 seconds). The average length of symbols for variable fields after 520 seconds is 1.67 bytes. It means that about 33% of all variable symbols are represented by 1 byte and the others are by 2 bytes.

Table VIII shows the results for varied buffering time $t_b = \{2, 5, 10\}$. The results indicate that the proposed method maintains its effectiveness while that of zip compression decreases. This overhead of zip is caused by the dictionary which is included the chunk, which makes the traffic volume relatively larger when the buffering time becomes shorter.

We also show the results in the case where only one type of data sources exists. The buffering time is $t_b = 5$ and the number of data sources is 50. The values in Table IX are the average traffics after 520 seconds in the evaluation. From the table, the effectiveness of compression varies according to the data type. We discuss these results in Subsection IV-C.

C. Discussion

Light data in IoT/CPS has a number of variable types. Naturally, variables that have few value patterns (i.e. boolean and numerical number with few digits) or highly biased distribution can be compressed effectively. On the other hand, when variables have numerous value patterns or unique patterns used only once (i.e. URI including UUID), the effectiveness of the proposed method can be lower. In the evaluation of the paper, we assume the geographical locations as constant fields, then the dictionary encoding can be highly effective.

For the variable fields, such as *type*, *format*, *temperature*, and *humidity* (e.g. environmental instruments in the evaluation) can positively affect the compression ratio of the proposed method, while *uri* and *event* (e.g. video camera and event detector) field cannot be compressed effectively. When we apply the proposed method to practical situations, we need to adjust the proposed method in consideration of these attributes. As for applying VBC to timestamp fields in the proposed method, if the field name and its variable type are given previously, we can adopt a suitable algorithm to the field. It generates a trade-off relationship with the flexibility of the data schema.

V. Conclusion

In this paper, we proposed an effective compression method for light data, such as numerical, string, or boolean, in IoT/CPS. In the proposed method, light data is divided into constant, variable, and timestamp fields. Closer symbols are released to the constant fields that appear in the same chunk frequently, and symbols for variable fields are released based on the constant field symbols of the own entry to maintain the effectiveness of VBC column encoding. We evaluated the proposed method with many types of light data sources in IoT/CPS and showed that the proposed method can reduce traffic volume between the edge computers and data centers and can improve compression ratio by 24% compared with zip (Deflate).

In the future, we plan to invent an adjusting mechanism for the proposed method by automatically dividing light data into constant/variable fields and judging whether or not to apply column encoding individually. We will also develop the tuning mechanism for the parameters.

References

[1] M. Hermann, T. Pentek, and B. Otto, "Design principles for industrie 4.0 scenarios," in *2016 49th Hawaii International Conference on System Sciences (HICSS)*, Jan 2016, pp. 3928–3937.

[2] K. Su, J. Li, and H. Fu, "Smart city and the applications," in *2011 International Conference on Electronics, Communications and Control (ICECC)*, Sept 2011, pp. 1028–1031.

[3] T. Nam and T. A. Pardo, "Conceptualizing smart city with dimensions of technology, people, and institutions," in *Proceedings of the 12th Annual International Digital Government Research Conference: Digital Government Innovation in Challenging Times*, ser. dg.o '11. New York, NY, USA: ACM, 2011, pp. 282–291. [Online]. Available: http://doi.acm.org/10.1145/2037556.2037602

[4] D. Bandyopadhyay and J. Sen, "Internet of Things: Applications and challenges in technology and standardization," *Wireless Personal Communications*, vol. 58, no. 1, pp. 49–69, 2011.

[5] E. A. Lee, "Cyber Physical Systems: Design Challenges," in *Proceedings of IEEE ISORC 2008*, May 2008, pp. 363–369.

[6] Cisco, "Global Cloud Index," available at https://www.cisco.com/c/en/us/solutions/collateral/service-provider/global-cloud-index-gci/white-paper-c11-738085.html.

[7] X. Sun and N. Ansari, "Edgeiot: Mobile edge computing for the internet of things," *IEEE Communications Magazine*, vol. 54, no. 12, pp. 22–29, December 2016.

[8] ETSI, "Multi-access Edge Computing," available at https://www.etsi.org/technologies-clusters/technologies/multi-access-edge-computing.

[9] P.Deutsch, "DEFLATE Compressed Data Format Specification version 1.3," available at https://www.rfc-editor.org/info/rfc1951.