2017 IEEE Third International Conference on Big Data Computing Service and Applications

Data Reduction as a Service in Smart City Architecture

Aseel Alkhelaiwi
Department of Computer Science
University College Cork
Cork, Ireland
aa8@cs.ucc.ie

Dan Grigoras
Department of Computer Science
University College Cork
Cork, Ireland
d.grigoras@cs.ucc.ie

***Abstract*— A wide range of crowdsensing smart city applications utilize cloud computing to send, store and publish data. The amount of data sent to the cloud is relatively large and this introduces bandwidth, network and storage challenges. In this paper, we offer a smart city architecture that includes a data reduction service located in the proximity of the crowd. In this data reduction service, we propose a lossless compression step for single-precision floating-point data received from the crowd, such as accelerometer readings and GPS coordinates. Floating-point compression has proved to reduce the cost of transmitting a large amount of data. Our compression method was evaluated and achieved a good compression ratio.**

Keywords- crowdsensing; compression; floating point; smart city; cloud

I. INTRODUCTION

In recent years, mobile device capabilities have been improving, with greater computational resources, different communication protocols (Bluetooth, Wi-Fi, 3G and 4G, etc.), and multiple sensing units (cameras, accelerometers, GPS, etc.). With these substantial mobile device capabilities, the crowdsensing paradigm has been introduced. Mobile crowdsensing refers to a wide range of sensing applications that employ mobile devices and wireless networks [1]. These include smart city applications that focus on improving citizens' quality of life, such as through better traffic management and effective disability support. Crowdsensing is different to existing sensing solutions that use particular networks of sensors. Crowdsensing makes use of human intelligence to notice, collect and send the sensed data by users' mobile devices. However, due to the limited resources of mobile devices, sensed data are usually offloaded and processed in data centers - clouds. Mobile cloud computing (MCC) is an enabling technology for crowdsensing in that it uploads a large amount of sensed data from mobile phones and other sensors to the cloud to be processed in the cloud to serve different smart city applications [2].

However, the increasing volume of crowdsensing data rises the necessity to adapt local processing in order to send and store these data efficiently in clouds. In other words, sending a large amount of sensed data to the cloud will require a high bandwidth and network traffic and will consume a lot of energy. These are real challenges that can be overcome by processing data at the edge (i.e. local processing). Another challenge lies in the context of the cloud: the more data are sent to the cloud, the greater the size of the data that need to be managed. Sometimes, the free space in the cloud is limited: for example, Google Drive provides 5 GB, then users need to selectively send the data to the cloud as all cloud providers gave users pay as you go storage. Thus, data sent to the cloud need to be reduced in size.

A large number of smart city applications (e.g. regarding potholes, traffic and the environment) require the crowd to send data that contain floating-point values, such as accelerometer readings, GPS coordinates, etc. Large data sets of floating-point values will consume resources in terms of bandwidth and storage.

One of the most common techniques for data reduction is compression. However, general compression algorithms do not always achieve a good compression ratio [3], since they do not take advantage of some of the characteristics of floating-point data sets, such as similarity of consecutive values in scientific data [4], [5].

In this paper, we show the importance of local processing by presenting a smart city architecture that includes a data reduction service as a main service. Instead of locating this data reduction service in the cloud, the service is placed as close to the crowd as possible, in public local servers (similar to cloudlets) that are distributed around the city to serve the community. In these servers, data received from the crowd are filtered and reduced through different steps. The local processing of crowd data has several benefits, especially in reducing the amount of traffic and data filtering. The allocation of a reduction service on local servers distributed around a city is reasonable where the Wi-Fi network is designed to support the community. In this paper, we make the following specific contribution:

- A lossless compression step that is performed on single-precision floating-point data sets and takes advantage of the distributive nature of local servers around a city. This compression is the main reduction step that completes the data reduction

978-1-5090-6318-5/17 $31.00 © 2017 IEEE
DOI 10.1109/BigDataService.2017.24

Table 1: Floating point representation

Precision	Sign	Exponent	Mantissa
Single	1	8	23
Double	1	11	52

Figure 1: Single-precision floating-point format of IEEE 752 standard

service in our cloud-centered smart city architecture.

The rest of this paper is organized as follows. Section II presents the background. Related work is summarized in Section III. Section IV summarizes our smart city architecture. The proposed data reduction method is given in section V and the evaluation results in section VI. Section VII concludes the paper.

II. BACKGROUND

A. Single-precision floating point

IEEE floating-point numbers have three basic components: the sign, the exponent, and the mantissa. The number of bits in single and double precision is shown in Table 1. In this paper, we focus on single-precision floating-point values (Fig. 1).

B. Crowdsensing

Mobile devices and their sensors have led to sophisticated context-aware applications and systems. This area has captured a large amount of interest in how users' mobile phones can contribute sensor data towards enabling, for example, environmental (e.g. air pollution and transportation), community and healthcare awareness.
Context awareness is considered the first generation of sensing and is defined by Pascoe as the ability of devices to sense, interpret and respond to aspects of a user's local environment [6]. Today, crowdsensing, the second generation of sensing, is receiving a lot of attention. Crowdsensing refers to the acquisition of sensor data from multiple (not only one) mobile devices, generally located in the same geographic area. There are some approaches, such as mobile sensing [7] and public sensing [8], that are similar in definition or goals to crowdsensing and are sometimes used interchangeably in the field.

III. RELATED WORK

Although data compression is an attractive topic in the literature, only a small number of studies have focused on the floating-point compression that is needed in big data scenarios, such as crowdsensing, smart city data and scientific data.

A. Scientific Data

In [9], the authors propose a compression method for floating-point data streams. They used a prediction method in order to find similar numbers in a stream and use them as predictions. Then, after performing some operations on the actual number and the predictions, the residuals are compressed using multiway compression. Furthermore, the authors of [10] proposed an algorithm that compresses sequences of IEEE double-precision floating-point values as follows. First, the algorithm predicts each value in the sequence and XORs (Exclusive ORs) it with the true value. If the predicted value is close to the true value, the sign, the exponent, and the first few mantissa bits will be the same. Our work has no prediction method; the compression was performed easily without any prediction limitations (i.e. over-processing).

In [11], a floating-point compression approach (fzip) is proposed. It has two stages: the first uses the coding scheme in Burrows-Wheeler Transform (BWT) compression [12]; the second is value and prefix compression, in which fzip uses arithmetic codes to encode the prefixes. In other words, a different pattern is compressed at each stage. Fzip achieves a good compression ratio but performs badly in terms of runtime. Other work is proposed in [13], in which the authors offer a binary masking technique that partially increases the regularity of high-performance computing data sets in order to create high compressible data sets before applying it for compression. Their work shows a good improvement in compression ratio. However, their masking performs well with data of high similarity (neighbor elements in high-performance computing have close values). Finally, in [14], the authors propose a delta compression algorithm to compress 64-bit floating-point values by storing the higher-order differences between values.

The majority of the works above took advantage of the similarities in some scientific data sets and their work will not be as effective if the data set values are random. In this paper, we deal with both cases: the repeated values in GPS coordinates and the randomness of accelerometer readings.

B. Audio and Image Compression

In terms of audio and image compression, Ghido proposes a compression algorithm for floating-point audio data [15]. The algorithm transforms the floating-point values into integers, maintains the properties of the original values and creates an extra binary stream used for the decompression of the original floating-point values. In our approach, we neither made any transformation nor added an extra binary stream; the exponent part is used for reconstruction in the cloud (in accelerometer readings only; GPS coordinates are reconstructed in the cloud without the exponent or any

additional bit).
Another work that represents the floating point as an integer proposes an addition to the JPEG2000 standard in order to encode floating-point data efficiently with bit-plane coding algorithms [16]. Furthermore, in [17], changes are performed on JPEG2000 to adapt lossless floating-point compression, such as optimizations in the wavelet transformation and beforehand signaling of special numbers. In [18], the authors use a context-based arithmetic coder for single-precision floating-point coordinates. They use the exponent to shift between different arithmetic contexts, while in our situation we use the exponent to represent the exponent number and classify the integer part (see section V for more details).

IV. Overview of Smart City Architecture

A. Producers

In our scenario, a subset of users, from two to what we call a crowd, located in a particular area of a city, deploy and use an Android sensing application on their mobile devices.

B. Public Local Servers

This layer consists of public local servers that are distributed around the city for the purpose of collecting crowd data, storing them for a period of time and processing them locally before sending the data to the cloud. These public servers run other applications, such as traffic or environment monitoring, as well. These servers receive the sensed data, perform processing and send the filtered data to the cloud.

C. Cloud Platform

The cloud computing platform performs annotation on the data received and provides databases in which sensor data are stored. Furthermore, the platform runs a number of smart city applications that process the sensor data received in the cloud.

D. Consumers

Consumers are citizens, or private or public authorities, who consume the sensor data published by the sensor data providers. There could be one, two or a large number of consumers. Consumers need to register in the cloud in order to use (i.e. install) smart city applications that utilize the sensor data stored in the cloud.

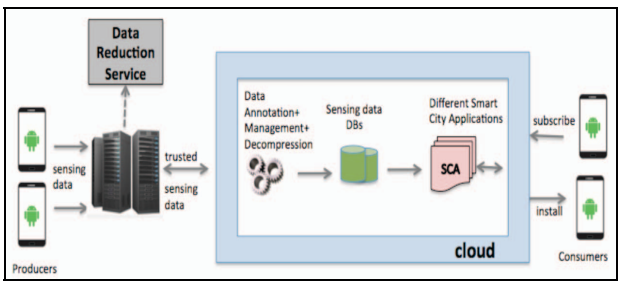

Figure 2: Smart city architecture

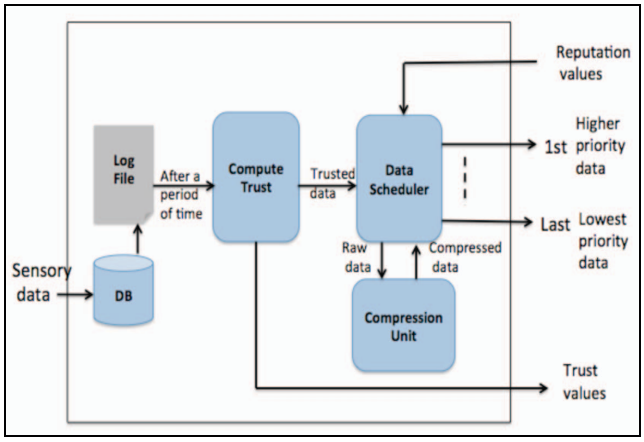

Figure 3: Data reduction service in the public local servers around the city

V. Data Reducton

Data reduction is performed in our proposed architecture by three services running on local servers distributed around the city (Fig. 3), the compression service being our contribution in this paper.
We briefly review the trust and schedule services below before introducing the compression service.

A. Trust and Scheduling Service

The term "trust" refers to building the user's trust and level of reliability in his/her current contribution. On the other hand, "reputation" refers to updating the user's reputation for all of his/her contributions regularly and takes place in the cloud.
When sensor data are received by the local server, trust is calculated using the weighting factors presented in [19]. If the trust value for this specific contribution is below the predefined threshold [19], the contribution will be discarded, otherwise, the contribution is considered trustworthy and will be moved to the scheduling stage.
In the scheduling stage, instead of offloading all trusted sensor data to the cloud, only part of them are sent by the scheduler [20]. There is a priority weight that has, by default, the value of 0 for every user contribution at the beginning. Its value is calculated using different factors, as presented in [20]. The scheduler performs two main jobs:

1) If the local server receives a number of contributions for the same location, the scheduler will select the data that will be sent to the cloud based on the reputation values of the users. Other sensor data are discarded but the number of contributions is considered as a value (similarity weight) that is added to the priority factor.

2) Schedules the sending of useful data to the cloud based on their priority. Data with high priority are sent first.

B. Compression Service

The data size reduction is our main contribution in this paper and will focus on single-precision floating-point data. We took advantage of the distribution of local servers in a smart city and present two compression techniques for location-based data (latitude and longitude) and an accelerometer's three-dimensional data.

1) *Location-based Data Reduction*

In this section, we propose a compression method that takes advantage of our proposed smart city architecture. Public local servers are distributed around the city to serve the citizens. Therefore, the following question was asked: why do we not benefit from the server distribution and apply a compression method to GPS coordinates (latitude and longitude) that are represented as single-precision floating-point numbers (32 bits)?

The servers serve an area that has a diameter not more than 130 kilometers (approximately 130 km is the distance that changes the integer value in the coordinates from x.0000 to y.0000, where x and y are two consecutive numbers). In other words, all contributions that are received in a particular server will have the same sign and integer part. However, some might receive a maximum of two integers and we will show how to handle this.

- Case 1 - servers cover an area that has the same integral part for both latitude and longitude (Algorithm 1).

Latitude and longitude are float numbers that are represented in Fig. 1. For the sake of simplicity, we will discuss latitude and the same will apply to longitude. Therefore, the sign part, the exponent part and the variable number of bits in the mantissa part can be removed, since they are the same for all contributions.

The contribution is then sent to the cloud, in which there is a table that has every server and its corresponding integral number covered in order to decompress. The decompression in the cloud is covered in further work and is out of the scope of this paper.

The number of bits in the mantissa will differ depending on how large the integer is, as shown in Table 2. The number of bits that are removed for compression will be from 10 to 16 (if we assume that the highest integer number is 255). This means that the higher the integer the more bits to compress. The reason for a missing bit is because of the IEEE float number presentation. For example, the number 13 is, in binary, 1101 but after normalization, the most significant bit is hidden and only 3 bits from the mantissa are eligible for our compression method.

- Case 2 - if the server covers an area that has two different integral parts (Algorithm 2).

The same procedure as that for Case 1 will be applied, but with a minor difference: the server needs to add 1 bit (called a decision bit) at the beginning of the compressed value in order for the cloud to determine which integral part is considered for decompression.

For example, if the server always receives GPS coordinates that can have two possible integer parts, x and y, the server will assign 0 for the integer x and 1 for integer y and add it as the most significant bit in order for the cloud to decide which integer is needed. Thus, this case is different only in the decision bit that is added as the most significant bit (see Algorithm 2).

Algorithm 1: GPS coordinate compression (Case 1)

Input x.y (x is the integer part and y is the decimal part)
Output: Result (the compressed number in binary form)

If x == num **then** // num is the integer part that
Result = $(y_0 \ldots y_n)_b$ // is covered by the server
Else
no compression applied, use the 32 bit IEEE float-point presentation

Algorithm 2: GPS coordinate compression (Case 2)

Input x.y (x is the integer part and y is the decimal part)
Output: Result (the compressed number in binary form)

If x == num1 **then** // num1 is the first integer
Result= $(0\ y_0 \ldots y_n)_b$ // covered by the server
Else if x== num2 **then** // num2 is the second integer
Result= $(1\ y_0 \ldots y_n)_b$ //covered by the server
Else
no compression applied, use the 32 bit IEEE float-point presentation

Table 2: The number of bits in the mantissa to compress

Integer numbers	Number of bits to compress in mantissa	Sign + exponent + bits in mantissa to compress
2, 3	1 bit	1+8+1=10
4-11	2 bits	1+8+2=11
8-15	3 bits	1+8+3=12
16-31	4 bits	1+8+4=13
32-63	5 bits	1+8+5=14
64- 127	6 bits	1+8+6=15
128-255	7 bits	1+8+7=16

2) *Accelerometer Data Reduction*

During a single event, an accelerometer will return data for three coordinate axes (x, y and z). These data values are single-precision floating-point numbers. In this section, we propose a compression method that will reduce the size of these float numbers by attempting to remove some bits that can be recovered easily later in the cloud.

We consider the integral number (the integer part of the float number) to be from 0 to 39, based on our experiment in which 39 was the highest integer we got in all axes. However, our method will also work with any number higher that 39.

The sign, exponent and mantissa of the 32-bit floating-point numbers are treated as follows (Algorithm 3):

1) The sign bit is left unchanged in our method, in which 0 corresponds to a positive number and 1 corresponds to a negative number.
2) The exponent is mainly 8 bits. If the exponent is negative, the number is left uncompressed since there is no way to predict the numbers after the float point. On the other hand, if the exponent is positive, then we decrease the number of bits in the exponent to 3 bits. These 3 bits represent the number of bits moved after the float point during the normalization process. For example, if we have the number 13.1857, the mantissa part will look like this before normalization: 1101.0011. After normalization, the number will look like this: 1.1010011×2^3, whereby the point is moved to the most significant 1. Since the number of bits moved after the point is three, then the exponent will only represent number 3. In our integer part range (1-39), the exponent range is from 0 to 5. Therefore, 3 bits are reasonable to present the exponent part and there is no need to add 127 (single-precision bias). The bias addition will take place later in the cloud in the decompression process. Table 3 shows the numbers and their corresponding exponent values that will be stored in the exponent part.

3) In the mantissa, we use the exponent to categorize the integers and decrease the number of bits that represent the integer number (1-39). The categorization of the integers is presented in Table 4. The reduction of the number of bits will take place in two steps:
   3.1) The most significant bit for all integers under the exponents from 1 to 5 is always 1. This bit is going to be removed all the time (the hidden bit in IEEE floating-point). Thus, all the numbers with an exponent from 1 to 5 have one bit removed.
   3.2) After removing the most significant bit "1" from all the numbers, the exponents 1 and 2 will have 1 bit and 2 bits, respectively, to represent their integer numbers (Table 4). Therefore, these integers are left unchanged. On the other hand, the integers covered by the exponents 3, 4 and 5 can be further altered. Starting from numbers covered by exponent 5 (32-39), the new most significant bit after removing "1" will always be "0". Thus, we can also remove this bit and represent the numbers from 32 to 39 only in 4 bits. Numbers covered by the exponent 3 are slightly different. The new most significant bit (after removing 1) can be either 0 or 1. Therefore, we need some indication if we want to remove this bit in our method. Since we represent the exponent in 3 bits and we only have exponents from 0 to 5, we can use the exponent 6 to help us indicate the bit in the decompression. In other words, we will use the exponents 3 and 6 in our compression for the integer values from 8 to 15. Exponent 3 indicates that we remove "0", while exponent 6 indicates that we remove "1". Therefore, numbers from 8 to 15 can be represented in only 2 bits. The same will happen to numbers from 16 to 31, for which we use exponent 4 to indicate that we remove "0" and exponent 7 to indicate the removal of the bit "1". Thereafter, the numbers from 16 to 31 are represented by only 3 bits. For an illustration, see Table 5.

Table 3: Exponent values for every integer number

Number	Exponent
0	Negative exponents
1	$0\ (000)_b$
2 and 3	$1(001)_b$
4-7	$2(010)_b$
8-15	$3(011)_b$
16-31	$4(100)_b$
32-39	$5(101)_b$

Table 4: Integer categories

Exponent	Integer	Binary
0	1	1
1	2 and 3	10 and 11
2	4-7	100-111
3	8-15	1000-1111
4	16-31	10000-11111
5	32-39	100000- 100111

Table 5. Number of bits after steps 3.1 and 3.2

Number	Exponent	Number of bits after step 3.1	Number of bits after step 3.2
2 -3	$1(001)_b$	1 bit	1 bit
4-7	$2(010)_b$	2 bits	2 bits
8-15	$3(011)_b$ and $6(110)_b$	3 bits	2 bits
16-31	$4(100)_b$ and $7(111)_b$	4 bits	3 bits
32-39	$5(101)_b$	5 bits	4 bits

Algorithm 3: Accelerometer data compression

Input x.y (x is the integer part and y is the decimal part)
Output: Result (the compressed number in binary form)

If x == 1 **then**
  Exponent = 000
  Mantissa=$(y_0 \ldots y_n)_b$
Else If x == 2 **OR** x == 3 **then**
  Exponent = 001
  Mantissa=$(x_1)_b (y_0 \ldots y_n)_b$ //x_0 is removed
Else If x >=4 **AND** x<= 7 **then**
  Exponent= 010
  Mantissa=$(x_1x_2)_b (y_0 \ldots y_n)_b$ //x_0 is removed
Else If x >=8 **AND** x<= 11 **then**
  Exponent= 011
  Mantissa=$(x_2x_3)_b (y_0 \ldots y_n)_b$ // x_0 and x_1 removed
Else If x >=12 **AND** x<= 15 **then**
  Exponent= 110
  Mantissa=$(x_2x_3)_b (y_0 \ldots y_n)_b$ // x_0 and x_1 removed
Else If x >=16 **AND** x<= 23 **then**
  Exponent= 100
  Mantissa=$(x_2x_3x_4)_b (y_0 \ldots y_n)_b$ // x_0 and x_1 removed
Else If x >=24 **AND** x<= 31 **then**
  Exponent= 111
  Mantissa=$(x_2x_3x_4)_b (y_0 \ldots y_n)_b$ // x_0 and x_1 removed
Else If x >=32 **AND** x<= 39 **then**
  Exponent= 101
  Mantissa=$(x_2x_3x_4x_5)_b (y_0 \ldots y_n)_b$ // x_0 and x_1 removed
Else
  no compression applied, use the 32 bit IEEE float-point presentation
  Exponent= $(c_0c_1c_2c_3c_4c_5c_6c_7)_b$
  Mantissa= $(x_1 \ldots x_n)_b (y_0 \ldots y_n)_b$
End if
Result=(Signbit)(Exponent)(Mantissa)

VI. EVALUATION

In our evaluation, we assessed the effectiveness of our compression method by first comparing the size of the data before and after performing the compression for both GPS coordinates and accelerometer readings. We then compared our method with some of the existing compression algorithms.

A. Experimental Setup

The compression ratio described above was first evaluated with a data set that contains GPS and accelerometer readings (i.e. both GPS coordinates (latitude and longitude) and three axes (x, y and z) of accelerometer readings). In other words, there are five entries (latitude, longitude, x, y and z) for every sensing contribution. These entries are represented as a single-precision floating point and this is how it is received by the server. The data set is obtained by an Android app that we developed in a previous work [20].
In our evaluation, we had the cloud and one local server that received the sensing data and started performing our compression on the set. The data set consisted of 2,000 readings and was about 1.2 MB (even though the data set was not large, we wanted to examine our method on real sensing data): the GPS coordinates data were 0.51 MB and the accelerometer data was 0.72 MB.

B. Implementing the Compression Method

We implemented and ran the proposed compression method on the data set. The location-based data reduction (step 1) was performed on the GPS coordinates and the accelerometer data reduction was applied to the accelerometer axes. The data set size was reduced to 0.82 MB, which means we removed up to 33% of the data.

C. Comparison

We compared our proposed compression method in terms of compression performance with one general-purpose compression algorithm (zlib) [21] and one floating-point data compression algorithm (Szip) [22]. Zlib offers general-purpose lossless data compression and is an abstraction of the deflate algorithm. Szip is a predictive compression algorithm based on the extended-Rice algorithm that uses Golomb-Rice codes for entropy coding. Table 6 demonstrates that our approach is more effective, as the data size after compression appears to be the lowest.
Furthermore, we performed the comparison of the GPS coordinates and accelerometer readings separately. Table 7 shows the data size after performing each compression algorithm. Our approach and Szip have a similar GPS data size after compression due to the high similarity of the GPS data received on one server. However, our approach shows a better result than the other two algorithms for the accelerometer readings, where the data are random.
Our approach deals with every data entry as a separate entry and does not use the history of the previous entries to predict the next value. Our method shows its effectiveness for similar and random values.

Table 6: Comparison of the data size after compression

Compression algorithm	Data size (MB)
zlib	0.98
Szip	0.91
Our approach	0.82

Table 7: Detailed data size after compression

Compression algorithm	GPS data size (MB)	Accelerometer data size (MB)
zlib	0.292	0.691
Szip	0.265	0.645
Our approach	0.261	0.562

VII. CONCLUSION

We proposed a smart city architecture that includes a data reduction service. In our architecture, we discuss the importance of local processing in the context of smart city and crowd sensing in order to filter the large amount of data received before being sent to, for example, a cloud. The pre-processing of data in local servers saves bandwidth and communication energy.

The data reduction service is located as close to the crowd as possible, in public local servers. These servers are distributed around the city to receive raw crowd data and, after processing, send trusted, filtered and compressed data to the cloud. We believe our proposed architecture will serve well in the context of big data produced by crowdsensing.

Our floating-point compression method for GPS and accelerometer data worked well. The approach showed very good results compared to other techniques, especially with compressing random data.

ACKNOWLEDGMENT

Aseel Alkhelaiwi's research is funded by King Saud University in Saudi Arabia.

REFERENCES

[1] R. K. Ganti, F. Ye, and H. Lei, "Mobile crowdsensing: current state and future challenges," IEEE Commun. Mag., vol. 49, no. 11, pp. 32-39, Nov. 2011.

[2] M. Rahimi, J. Ren, C.H.Liu, A.V. Vasilakos, N. Venkatasubramanian , "Mobile cloud computing: a survey, state of art and future directions," Mobile Network and Applications, vol. 19, no. 2, pp. 133-143, 2014.

[3] D. Saloman and G. Motta, Handbook of Data Compression, 5th ed. London: Springer, 2010.

[4] K. Kourtis, G. Goumas, and N. Koziris, "Improving the performance of multithreaded sparse matrix-vector multiplication using index and value compression," in Proceedings of the International Conference on Parallel Processing (ICPP), Sep. 2008, pp. 511-519.

[5] M. Burtscher and P. Ratanaworabhan, "FPC: a high-speed compressor for double-precision floating-point data," IEEE Transactions on Computers (TC), vol. 58, no. 1, pp. 18-31, Jan. 2009.

[6] J. Pascoe, "Adding generic contextual capabilities to wearable computers," 2nd International Symposium on Wearable Computers, pp. 92-99, 1998.

[7] D. Estrin, "Participatory sensing: applications and architecture [internet predictions]," IEEE Internet Computing, vol. 14, no. 1, pp. 12-42, 2010.

[8] D. Christin, A. Reinhardt, S. S. Kanhere, and M. Hollick, "A survey in mobile participatory sensing applications," Journal of System and Software, vol. 84, no. 11, pp. 1928-1946, 2011.

[9] S. Liu, X. Huang, Y. Ni, H. Fu, and G. Yang, "A versatile compression method for floating-point data stream", Fourth International Conference on Networking and Distributed Computing, Los Angeles, CA, 2013, pp. 141-145.

[10] P. Ratanaworabhan, Jian Ke, and M. Burtscher, "Fast lossless compression of scientific floating-point data," Data Compression Conference (DCC'06), 2006, pp. 133-142.

[11] K. R. Townsend and J. Zambreno, "A multi-phase approach to floating-point compression," IEEE International Conference on Electro/Information Technology (EIT), Dekalb, IL, 2015, pp. 251-256.

[12] M. Burrows and D. J. Wheeler, "A block-sorting lossless data compression algorithm," Systems Research Center, Technical Report 124, May 1994.

[13] L. A. B. Gomez and F. Cappello, "Improving floating point compression through binary masks," IEEE International Conference on Big Data, Silicon Valley, CA, 2013, pp. 326-331.

[14] V. Engelson, D. Fritzson, and P. Fritzson, "Lossless compression of high-volume numerical data from Simulations", Data Compression Conference, 2000, pp. 574-586.

[15] F. Ghido, "An efficient algorithm for lossless compression of IEEE float audio," Data Compression Conference, 2004, pp. 429-438.

[16] B. E. Usevitch, "JPEG2000 extensions for bit plane coding of floating point data", Data Compression Conference, 2003, pp. 451-461.

[17] M. N. Gamito and M. S. Dias, "Lossless coding of floating point data with JPEG 2000 Part 10," Applications of Digital Image Processing XXVII, 2004, pp. 276-287.

[18] M. Isenburg, P. Lindstrom, and J. Snoeyink, "Lossless compression of floating-point geometry," CAD2004, 2004, pp. 495-502.

[19] A. Alkhelaiwi and D. Grigoras, "The origin and trustworthiness of data in smart city applications," IEEE/ACM 8th International Conference on Utility and Cloud Computing, 2015, pp. 376-382.

[20] A. Alkelaiwi and D. Grigoras, "Scheduling crowdsensing data to smart city applications in the cloud", Special Session on High Performance Grid Middleware, in conjunction with IEEE 12th International Conference on Intelligent Computer Communication and Processing, September 8-10, 2016.

[21] "zlib." [Online]. Available: http://www.zlib.net

[22] P.-S. Yeh, W. Xia-Serafino, L. Miles, B. Kobler, and D. Menasce, "Implementation of ccsds lossless data compression in hdf," Earth Science Technology Conference, 2002