Design of a Data Collection System with Data Compression for Small Manufacturers in Industrial IoT Environments

Chunju Tsai, Wen-Yueh Shih, Yi-Shu Lu, Jiun-Long Huang and Lo-Yao Yeh
Department of Computer Science, National Chiao Tung University, Hsinchu, Taiwan
Network and Information Security Division, National Center for High-performance Computing, Taichung City, Taiwan
E-mail: laura830615@gmail.com, wyshih@cs.nctu.edu.tw, barryrodkimo@gmail.com, jlhuang@cs.nctu.edu.tw, lyyeh@narlabs.org.tw

***Abstract*—With advance of IoT (Internet of Things) technology, many manufacturers install several sensors to monitor the status of machines and the health of the whole manufacturing process. In addition, the sensed data are usually transmitted to a back-end database for further analysis. However, the dramatic volume of data sensed by the sensors causes the problem of huge storage requirement and network traffic for the small medium manufacturers which have limited resource and budget in IT (Information Technology). To deal with this problem, we design a two-layered architecture using compression technique to reduce the network traffic. In addition, we use MongoDB, a NoSQL database, to store the compressed data due to MongoDB's excellent scale-out ability and cost-efficiency. We conduct several experiments to measure the performance of the proposed architecture with several compression methods. Experimental results show that with proper lossless compression method, the reduction ratio of the volume of the data is around 80% at the cost of slight increase in execution time.**

***Index Terms*—Data storage, Compression, Industrial IoT, IoT**

I. Introduction

With the advance IoT (Internet of Things), the concept of IIoT (Industrial IoT) [2] has been proposed and widely adopted by many manufacturers. Manufactures install several sensors to monitor the status (e.g., electricity, temperature, vibration and rotation speed) of machines to understand the health of the machines and the whole manufacturing process. In addition, the sensed data are usually transmitted to a back-end database for further analysis. Therefore, adopting IIoT technology unavoidably results in dramatic increase in storage requirement and network traffic. Take the data collected from a real factory in our field trial as an example. When the sample rate of each sensor is set to 20 Hz, the size of the data collected from four machines is about 10 GB per month. The huge requirement in storage and network traffic is a severe problem for small manufacturers which have limited resource and budget in IT(Information Technology).

To solve this problem, we design a two-layered architecture to aggregate the sensed data into a batch and then reduce the size of each batch by compression. Then, we store the compressed batch into a database. Instead of using traditional relational databases such as MySQL, we propose to use MongoDB, a NoSQL database, to store the data due MongoDB's excellent scale-out ability and cost-efficiency. Therefore, the small manufacturers can adopt the propose architecture to store the huge data with low cost with the aid of compression.

In this paper, we adopt both lossless and lossy compression methods. The lossless methods include ZLIB, GZIP, BZIP2, LZMA (Lempel-Ziv-Markov chain algorithm), and LZO (Lempel-Ziv-Oberhumer algorithm). And the lossy methods includes FFT (Fast Fourier Transform), DCT (Discrete Cosine Transform) and DWT (Discrete Wavelet Transform). In Section IV, we compare all of these compression methods on compression speed, space savings, and loss percentage for lossy methods. The information of the comparison will help the users to make a decision on the selection of the compression algorithm.

The rest of this paper is organized as follows. Section II introduces some the related works. Then, Section III introduces the design of our architecture. The experimental results are shown in Section IV. Finally, Section V concludes this paper.

II. Preliminaries

In [3], Jiang et al. introduce a framework to store the data gathered from lots of different types of devices. The proposed data storage framework stores the un-structural data in HDFS. The structural data (e.g., order information) which requires ACID (Atomicity, Consistency, Isolation, Durability) function are stored in a relational database, while the log data (e.g., sensor readings) are stored in a NoSQL database. Unfortunately, it does not deal with the problem of rapidly increasing data volume. In addition, the proposed data storage framework employs three storage systems (HDFS, a relational database and a NoSQL database), and such design unavoidably increases the maintenance cost. Due to the limited IT budget of small manufacturers, we argue that the proposed data storage framework is not suitable for them.

In [4], Mai et al. conduct several experiments to compare several database systems from different aspects such as query latency with different bulk sizes, number of threads, number of records, and so on. Although not the best at all aspects, MongoDB outperforms the others at most cases. In addition, the database size of MongoDB is relatively small when lots of records of data are stored. According to the advantages mentioned above, we choose MongoDB as our back-end database.

Fog computing [5] is an extension of cloud computing. In contrast to cloud computing, fog computing aims to keep data and computation close to end users at the edge of the network. By doing so, fog computing can reduce the network traffic load, and has some advantages such as low latency, high bandwidth, geo-distribution [6]. After a fog computing node completes its tasks, it can send the data to the cloud server which has a higher computation capability and can perform complex computation [1].

III. Proposed System and Implementation

A. Proposed System

Figure 1 shows the architecture of the proposed system. In our proposed system, each sensor connects to a Layer 1 Gateway and periodically sends the sensed data to the corresponding Layer 1 Gateway. When receiving a data record, the Layer 1 Gateway performs the following procedure to handle the data record.

- Step 1: Store the data record into its local storage.
- Step 2: Check whether the number of the data records stored in its local storage is greater than a user-defined threshold. If not, terminate the procedure.
- Step 3: Aggregate the stored data records into a small batch.
- Step 4: Send the small batch to the corresponding Layer 2 Gateway.
- Step 5: Remove the stored data records from the local storage.

To reduce the cost, we assume that users will use simple embedded systems (e.g., Arduino) as Layer 1 Gateways. Thus, Layer 1 Gateways do not perform compression in Step 3.

When receiving a small batch from the Layer 1 Gateway, the Layer 2 Gateway handles the received batch by the following procedure.

- Step 1: Store the small data batch into its local storage.
- Step 2: Check whether the number of the data records stored in its local storage is greater than a user-defined threshold. If not, terminate the procedure.
- Step 3: Aggregate the stored data records into a big batch.
- Step 4: Compress the big batch by the underlying compression method.
- Step 5: Send the compressed batch to the data server.
- Step 6: Remove the stored data records from its local storage.

Since the data records stored in the NoSQL database are compressed, several Web APIs are implemented to provide the applications a simple way to query data records of interest.

B. Data Compression

For small manufacturers, the budge of IT is not ample. Thus, to reduce the cost of storage, Layer 2 Gateways compress the received data records and send the compressed data to the data server. Our system employs the following compress methods.

- Lossless compression methods: ZLIB, GZIP, BZIP2, LZMA, LZO
- Lossy compression methods: FFT, DCT, DWT

In our system, the data are sent from Layer 1 Gateways to Layer 2 Gateways are in key-value format. Lossless compression methods can be directly used to compress key-value format data. Unfortunately, the lossy compression methods can only be used to compress numerical data. Because of the limitation, when using lossy compression methods to compress key-value format data, some preprocessing effort has to be done before compression. For example, the keys (usually in text format) and the values (usually in numerical format) have to be handled and stored separately. Moreover, in addition to the compressed the data records, the capture time of the first record in this batch is also stored in the database.

When compressing data, the lossy compression methods usually produce a matrix of coefficients which is usually the same size as the original data. The value and the number of coefficients significantly influence the accuracy of reconstructed data. Keeping more coefficients will make the decompressed data more accurate. Therefore, the percentage of the discarded coefficients affects not only the saving in space, but also the loss of information. Consider the original coefficient matrix $C_{m,n}$ shown in Equation (1), where m indicates the number of records, and n indicates the number of fields. Let the kept coefficient matrix $C_{p,n}$ be shown in Equation (1), where p indicates the number of kept rows. After compression, the values in matrix $C_{p,n}$ are rounded into three decimal precision and are sent to data server.

$$C_{m,n} = \begin{pmatrix} c_{1,1} & c_{1,2} & \cdots & c_{1,n} \\ c_{2,1} & c_{2,2} & \cdots & c_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{p,1} & c_{p,2} & \cdots & c_{p,n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m,1} & c_{m,2} & \cdots & c_{m,n} \end{pmatrix} \tag{1}$$

$$C_{p,n} = \begin{pmatrix} c_{1,1} & c_{1,2} & \cdots & c_{1,n} \\ c_{2,1} & c_{2,2} & \cdots & c_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{p,1} & c_{p,2} & \cdots & c_{p,n} \end{pmatrix} \tag{2}$$

For decompressing stage, the matrix $C_{p,n}$ is retrieved from the database. As shown in Equation (3), the discarded coefficients are set to zeros to form $C'_{m,n}$. Finally, the corresponding decompression method is employed to decompress the data on the basis of $C'_{m,n}$.

$$C'_{m,n} = \begin{pmatrix} c_{1,1} & c_{1,2} & \cdots & c_{1,n} \\ c_{2,1} & c_{2,2} & \cdots & c_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{p,1} & c_{p,2} & \cdots & c_{p,n} \\ 0 & 0 & \ddots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ddots & 0 \end{pmatrix} \tag{3}$$

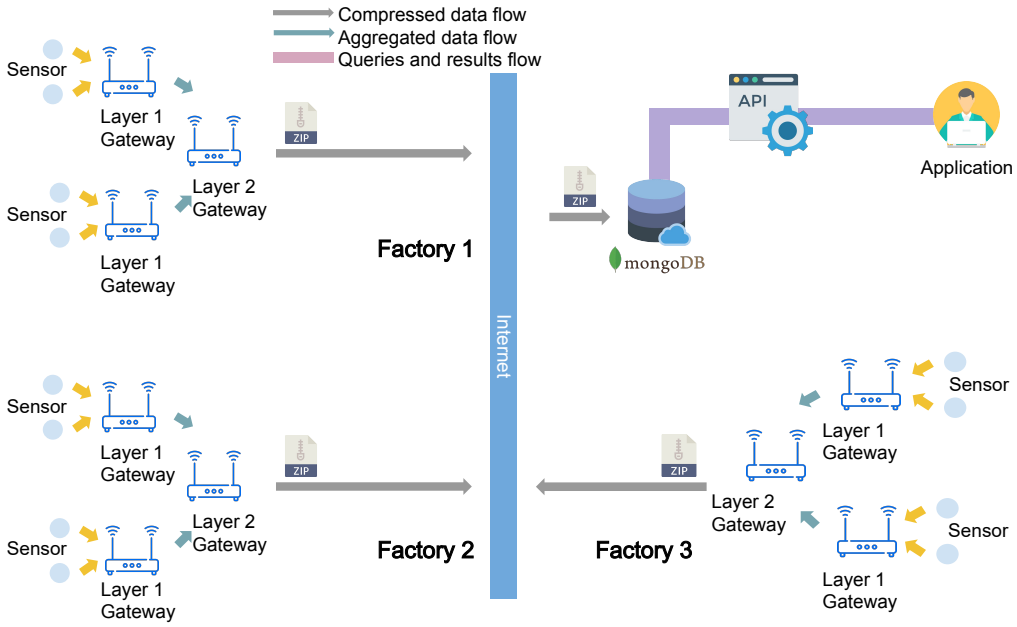

Fig. 1. Architecture

IV. Evaluation

A. Dataset and Performance Metrics

Our dataset is collected from the sensors deployed in a real factory. The dataset consists of 136,067,155 records, and the duration of the dataset is about two months. Each record is composed of 15 columns. Only one column is DateTime type, while the others are all numeric type. The size of the dataset is about 20 GB.

In order to demonstrate the data compression ratio clearly, we use space saving ratio, as shown in Equation (4), as our metric of data compression.

$$Space\ Saving\ Ratio = 1 - \frac{CompressedSize}{UncompressedSize} \quad (4)$$

Using lossy compression methods avoidably produces loss of information since the values of the decompressed data will not be equal to the original values. We use information loss ratio to measure the degree the loss of information. Suppose each batch consists of n records and each record consists of f numerical fields. Let r_i be the i-th record and $f_{i,j}$ be value of the j-th field of the i-th record. Also let $\hat{r}_i$ be the decompressed i-th record and $\hat{f}_{i,j}$ be decompressed value of the j-th field of the i-th record. Information loss ratio is defined as the average of the mean absolute percentage error (MAPE) of each record in a batch, and can be formulated as Equation (5).

$$Information\ Loss\ Ratio = \frac{1}{n}\sum_{i=1}^{n} MAPE(r_i), \quad (5)$$

where

$$MAPE(r_i) = \frac{1}{f}\sum_{j=1}^{f}\left|\frac{\hat{f}_{i,j} - f_{i,j}}{f_{i,j}}\right|. \quad (6)$$

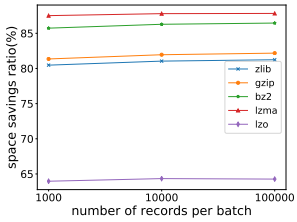

Fig. 2. Space saving with different number of records

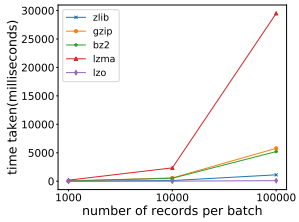

Fig. 3. Compression time taken with different number of records

B. Results of Lossless Compression Methods

As shown in Figure 2, the space saving ratio of all lossless compression methods are not significantly influenced by the batch size. The space saving ratio of LZO is much less than the others because the design of LZO aims to offer fast compression with the cost of low data compression ratio. Thus, as shown in Figure 3, LZO always spends the least time in compression. In contrast, although outperforming the other methods in terms of space saving ratio, LZMA always has poor performance in compression time. In addition, the compression time of LZMA increases drastically as the batch size increases. In our experiment, when the batch size is set to 100,000 records, the compression time of LZMA is at least 6 times longer than the other methods. We can observe from Figures 2 and 3 that BZIP2 is able to strike a good balance between space saving ratio and compression time.

C. Results of Lossy Compression Methods

The compression ability of lossy compression methods can be tuned by the setting of the percentage of discarded coefficients. The space saving ratio, compression time and

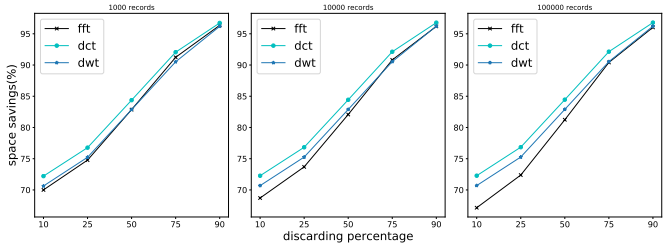

Fig. 4. Space Savings with Multiple Discard Percentages

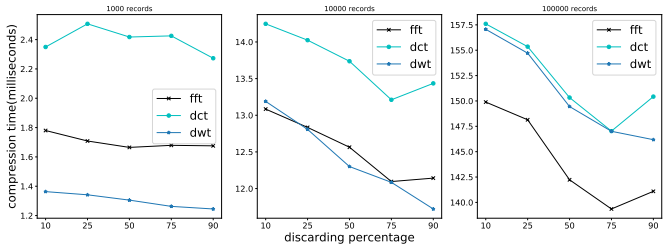

Fig. 5. Compression Time with Multiple Discard Percentages

information loss ratio of all lossy compression methods with the percentage of discarded coefficients varied are shown in Figures 4, 5, and 6, respectively. As shown Figure 4, DCT performs the best on space saving ratio. However, DCT spends the most time in compression in most cases. As the amount of data grows up, FFT performs worse and worse on space saving ratio, while DCT and DWT perform steadily on space saving ratio. We can observe from Figures 4, 5, and 6 that DWT is able to strike a balance among space saving ratio, compression time and information loss ratio.

V. Conclusion

In this paper, we proposed a data collection system for Industrial IoT environments with the aide of two-layered gateways and a NoSQL database (MongoDB). To decrease the storage cost, before sending data to MongoDB, Layer 2 Gateways perform compression to reduce the data size. Based on our experimental results, we have the following observations. When information loss is not tolerable, lossless compression methods should be employed. Our experimental results show that BZIP2 is a good choice due to the balance between space saving ratio and compression time. When some slight information loss is acceptable, lossy compression methods are recommended due to their high space saving ratio and fast compression. Our experimental results show that DWT is a good choice because of its high space saving ratio, low compression time and low information loss.

References

[1] F. Bonomi, R. Milito, P. Natarajan, and J. Zhu. Fog Computing: A Platform for Internet of Things and Analytics. In *Big Data and Internet of Things: A Roadmap for Smart Environments*, pages 169–186. Springer, 2014.

[2] H. Boyes, B. Hallaq, J. Cunningham, and T. Watson. The Industrial Internet of Things (IIoT): An Analysis Framework. *Computers in Industry*, 101, 2018.

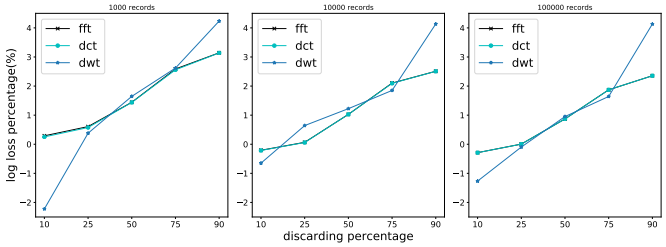

Fig. 6. Information Log Loss with Multiple Discard Percentages

[3] L. Jiang, L. Da Xu, H. Cai, Z. Jiang, F. Bu, and B. Xu. An IoT-Oriented Data Storage Framework in Cloud Computing Platform. *IEEE Transactions on Industrial Informatics*, 10(2):1443–1451, May 2014.

[4] P. T. A. Mai, J. K. Nurminen, and M. Di Francesco. Cloud Databases for Internet-of-Things Data. In *Proceedings of IEEE International Conference on Internet of Things (iThings)*, pages 117–124, 2014.

[5] N. Mohan and J. Kangasharju. Edge-Fog Cloud: A Distributed Cloud for Internet of Things Computations. In *Proceedings of the 2nd Cloudification of the Internet of Things (CIoT)*, pages 1–6, 2016.

[6] S. Yi, Z. Hao, Z. Qin, and Q. Li. Fog Computing: Platform and Applications. In *Proceedings of the 3rd IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb)*, pages 73–78, 2015.