# A Scalable Data Chunk Similarity Based Compression Approach for Efficient Big Sensing Data Processing on Cloud

Chi Yang and Jinjun Chen

**Abstract**—Big sensing data is prevalent in both industry and scientific research applications where the data is generated with high volume and velocity. Cloud computing provides a promising platform for big sensing data processing and storage as it provides a flexible stack of massive computing, storage, and software services in a scalable manner. Current big sensing data processing on Cloud have adopted some data compression techniques. However, due to the high volume and velocity of big sensing data, traditional data compression techniques lack sufficient efficiency and scalability for data processing. Based on specific on-Cloud data compression requirements, we propose a novel scalable data compression approach based on calculating similarity among the partitioned data chunks. Instead of compressing basic data units, the compression will be conducted over partitioned data chunks. To restore original data sets, some restoration functions and predictions will be designed. MapReduce is used for algorithm implementation to achieve extra scalability on Cloud. With real world meteorological big sensing data experiments on U-Cloud platform, we demonstrate that the proposed scalable compression approach based on data chunk similarity can significantly improve data compression efficiency with affordable data accuracy loss.

**Index Terms**—Big sensing data, cloud computing, data chunk, data compression, similarity model, scalability, MapReduce

## 1 Introduction

It is becoming a practical requirement that we need to process big data from multiple sensing systems. That is, we enter into the time of data explosion which brings about new scientific challenges for big sensing data processing. In general, big data [1], [2], [37], [38] is a collection of data sets so large and complex that it becomes extremely difficult to process with on-hand database management systems or traditional data processing tools. It represents the progress of the human cognitive processes, usually includes data sets with sizes beyond the ability of current technology, method and theory to capture, manage and process the data within a tolerable elapsed time [1], [2], [31], [34]. According to literature [1], [2], since 1980s, generated data doubles its size in every 40 months all over the world. In the year of 2012, there were 2.5 quintillion ($2.5 \times 1018$) bytes of data being generated every day. Hence, how to process big data has become a fundamental and critical challenge for modern society.

A very important source of big data is sensing systems, including camera, video, satellite, meteorology, connectomics, earthquake monitoring, traffic monitoring, complex physics simulations, genomics, biological study, medical research, gene analysis and environmental research [1], [2], [16], [17], [18], [19], [20], [43], [44], [45], [46] etc. The big sensing data from different kinds of sensing systems is high heterogeneous, and it has typical characteristics of common real world big data. They are five 'V's, Volume, Variety, Velocity, Veracity and Veracity.

To overcome the processing difficulties caused by five 'V's, of big sensing data, the trend to deploy big data processing on Cloud is getting popular day by day. Cloud computing provides a promising platform for big data processing with its powerful computation capability, storage, scalability, resource reuse and low cost, and has attracted significant attention in alignment with big data. In Amazon's recent real world big data processing on Cloud projects [41], [42], most of big data sets come from sensing systems. However, to process big sensing data can still be costly in terms of space and time even on Cloud platform. To reduce the overall time and space cost for big data, especially big sensing data processing on Cloud, different techniques have been proposed and developed [37], [38], [39]. But due to the size and speed of big sensing data in real world, the current data compression and reduction techniques still need to be improved. It has been well recognized that big sensing data or big data sets from mesh networks such as sensor systems and social networks can take the form of big graph data. To process those big graph data, current techniques normally introduce complex and multiple iterations. Iterations and recursive algorithms may cause computation problems such as parallel memory bottlenecks, deadlocks on data accessing, algorithm inefficiency [30]. In other words, under some circumstances, even with Cloud platform, the task of big data processing may introduce unacceptable time cost, or even lead to processing failures.

- *C. Yang is with the Unitec Institution of Technology, New Zealand. E-mail: chiyangit@gmail.com.*
- *J. Chen is with the Swinburne University of Technology, Australia. E-mail: jinjun.chen@gmail.com.*

*Manuscript received 30 July 2015; revised 4 Dec. 2015; accepted 25 Jan. 2016. Date of publication 18 Feb. 2016; date of current version 27 Apr. 2017. Recommended for acceptance by T. Li.*
*For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.*
*Digital Object Identifier no. 10.1109/TKDE.2016.2531684*

To further improve the data size reduction, reduce the processing time cost and release the iterations in processing big sensing data, in this paper, we propose a novel technique based on data chunk partitioning for effectively processing big data, especially streaming big sensing data on Cloud. With this novel technique, big sensing data stream will be filtered to form standard data chunks at first based on our predefined similarity model. Then, the coming sensing data stream will be compressed according to the generated standard data chunks. With the above data compression, we aim to improve the data compression efficiency by avoiding traditional compression based on each data unit, which is space and time costly due to low level data traverse and manipulation. At the same time, because the compression happens at a higher data chunk level, it reduces the chance for introducing too much usage of iteration and recursion which prove to be main trouble in processing big graph data.

The contents of this paper are organized as follows. In Section 2, we review related work and conduct problem analysis. In Section 3, a data similarity model will be defined and introduced. With that similarity model, the formation process of standard data chunks will be offered by training initial data stream. Then, we will introduce our streaming sensing data compression according to the standard data chunks. In Section 4, all the related scalable algorithms are offered, including scalability with Mapreduce, standard data chunk generation algorithm and scalable compression algorithm. In Section 5, the experimental results will be analyzed to show significant data compression performance gains. In addition, the accuracy loss will also be discussed in relation to compression effectiveness. In Section 6, we will conclude the paper with a brief outlook of future work.

## 2 Related Work and Problem Analysis

Some techniques have been proposed to process big data with traditional data processing tools such as database, traditional compression, machine learning, or parallel and distributed system [1], [2]. In the following Section 2.1, those current popular techniques for big data processing on Cloud will be introduced and analyzed.

### 2.1 Related Work: Big Data Processing on Cloud

Nowadays, lots of big data sets or streams come from sensing systems which are widely deployed in almost every corner of our real world to assist our everyday life [42], [43], [44]. In order to cope with that huge volume big sensing data, different techniques can have been developed [42], online or off-line, centralized or distributed. Naturally, the computational power of Cloud comes into the sight of scientist for big sensing data processing [23], [24], [25], [26].

Cloud computing provides comprehensive computing and storage resources enabling a pay-as-you-go business model by offering IT resources as services [5], [6], [7], [8], [9], [10]. As a result, Cloud provides a promising scalable platform for big data storage, dissemination and interpreting [3], [4]. At present, some research has been done about how to process big data with Cloud. For example, Amazon EC2 infrastructure as a service is a typical Cloud based distributed system for big data processing. Amazon S3 supports distributed data storage. MapReduce [8], [9], [10], [11], [12], [13], [14], [15], [40] is adopted as a programming model for big data processing with Cloud. MapReduce [27], [28] has been widely revised from a batch processing framework into a more incremental one for analyzing huge-volume of incremental data on cloud. It can sort petabytes of big data in only a few hours. The parallelism also provides some possibility of recovering from partial failure of servers or storage during the operation. In our work, MapReduce also acts as a base for parallel processing on Cloud.

A significant amount of research has been done on the processing of incremental data on cloud. Kienzler et al. [8] developed a "stream-as-you-go" approach for accessing and processing incremental big sensing data on cloud via a stream-based data management architecture. The extension of traditional Hadoop framework [11] was made to develop a novel framework named Incoop by incorporating several techniques like task partition and memorization-aware schedule. Olston et al. [9] present a continuous workflow system called Nova on top of Pig/Hadoop through incremental data processing.

Sensor-Cloud [21], [22], [41] is a unique sensing data storage, visualization and remote management platform that leverages powerful cloud computing technologies to provide excellent data scalability, fast visualization, and user programmable analysis. Sensor-Cloud platform [28], [38], [39] has been developed including its definition, architecture, and applications. However, the Sensor-Cloud has less consideration for the big data in complex network topology. Due to the features of high variety, volume, and velocity, big data is difficult to process using on-hand database management tools or traditional Sensor-Cloud platform. The typical examples of big sensing data of complex networks are social network and large scale sensor networks. Under the theme of those complex network systems, it may be difficult to develop time-efficient detecting or trouble-shooting methods for big data processing in complex network systems in real time [21], [22], [23], [27]. Current typical techniques such as MapReduce may introduce high computation cost [29], [30], [31], [32], [34] when encountering big graph sensing data. More work is still expected to improve the effectiveness and efficiency in terms of big graph data processing on Cloud. Therefore, we aim to offer an optimal solution for real-time streaming big graph data compression for applications on Cloud.

Specifically, to reduce the volume of big data sets, different data reduction methods have been proposed recently. For example, in paper [29], the work considers compressed sensing for sparse and low-rank tensors. Low-rank tensors were synthesized as sums of outer products of sparse loading vectors, and a special class of linear dimensionality-reducing transformations that reduce each mode individually. It was proved that interesting "oracle" properties exist. The proofs naturally suggest a two-step approach for processing big sensing data on Cloud. However, the extension and improvement are required to face new issues of big data and Cloud.

In paper [31], a data quality (DQ)-centric big data infrastructure for federated sensor service clouds was proposed. The paper explores the advantages and limitations of current big data technologies in building various components of the platform. In paper [32], the work is focused on state-of-the-art analysis and open research issues in the context of Cloud-enabled largescale sensor networks, which naturally complement the emerging big sensing data paradigm. It focuses in particular on the issue of representing

and managing Big Data, with emphasis on analytics over Big Data, as well as processes and architectures working with such data, with emphasis on Wireless Sensor Networks (WSNs), and draws future directions in this field. In paper [33], Recovery algorithms are developed in compressive sampling (CS). Specifically, to speed up the least-squares module, the matrix-inverse-update algorithm is adopted. That developed algorithm has the potential to be used for compressing big sensing data on Cloud. But cannot be used directly due to the new requirement such as extreme high data speed, distributed environment and scalability.

In paper [34], an anomaly detection technique was used for through-wall human detection to demonstrate the big sensing data processing effectiveness. This technique is totally based on compressive sensing. The results showed that the proposed anomaly detection algorithm could effectively detect the existence of a human being through compressed signals and uncompressed data. In paper [35], an adaptive data gathering scheme by compressive sensing for wireless sensor networks was developed. By introducing autoregressive (AR) model into the reconstruction of the sensed data, the local correlation in sensed data is exploited and thus local adaptive sparsity is achieved. Up to about 8 dB SNR gain can be achieved over conventional CS based method. There is also technique focusing on parallel data storing over large-scale distributed storage stock of Cloud platform. The stored big graph data or stream data sets will be queried and evaluated as the model of distributed data-base in Cloud, such as "Hydoop" [40] and its related "Hive", "HBase", "Zookeeper", and so on.

In paper [30], [36], a spatial and temporal compression model is designed for compressing big sensing data with significant performance gains. The approach in [30] consists of two main technique parts. The first one focuses on reducing the data size over Cloud platform with spatiotemporal compression. A clustering algorithm is developed based on spatial similarity between multiple time series or streams of data. It compares data streams according to the topology of streaming data graph. However, that techniques work at the level temporal or spatial data correlation which could be further improved if other data correlation or relationship can be exploited.

Based on the above literature, current big sensing data processing or compression algorithms work at sampling or data unit navigation level. However, due to the huge volume of the big sensing data, the only data size reduction at that level is not enough. Novel data compression techniques should be developed to dramatically reduce the stored data size and time cost for data manipulation. Instead of compressing data unit one by one at sampling stage and traditional compression techniques which compare data units at a low level, some compression based on huge data blocks in big sensing data should be developed. At the same time, the computation power and the scalability feature of Cloud computing should be further exploited within the framework of MapReduce.

## 2.2 Problem Analysis

As offered in Section 2.1, big sensing data normally streams in with high speed in real time. It may take the form of graph data sometimes [27], [30]. Traditional data compression and current popular data compression techniques work at low basic data unit levels show their weakness and inefficiency when encountering big sensing data sets in the following aspects. (1) Most of traditional data compression techniques require whole data set navigation which costs huge amount of space and time during the compression and decompression [30]. Normally, compression algorithms work at a level which counts each basic data unit for their relationship. (2) Traditional data compression algorithm cannot make full use of scalability of Cloud. The centralized compression and decompression algorithms should be organized on the scalable platform such as Cloud in a more efficient way.

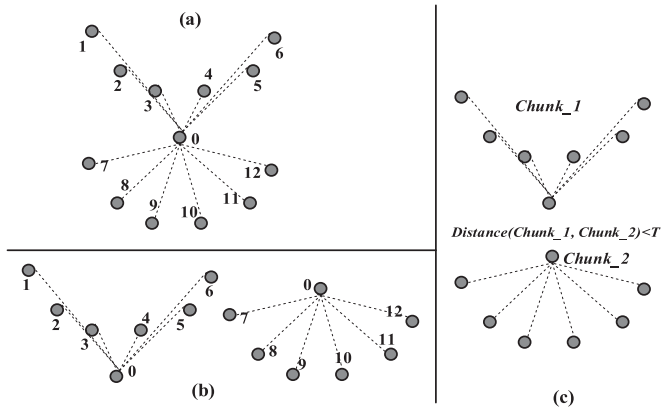



Fig. 1. Influence of two types of geometry similarities.

To compare different strategies for data compression, the example in Fig. 1 is used. As shown in Fig. 1a, a sensing system is deployed for collecting data from 12 different data sources. All the collected sensing data will be transmitted back to node 0. With a traditional data storage strategy, the data collected from sensing node 1 to sensing node 12 will be stored one by one with a certain predefined order. Under this storage theme, the traditional compression happens at each sensing node level. For instance, the data collected in node 1 will be analyzed and used for possibly compressing the data from node 3. However, the above compression approach working at sensing node level can be inefficient especially when the data volume and velocity is high. If more attention is paid to the topology of the sensing system in Fig. 1a, it can be found that the there are two similar sub-clusters under sensing node 0 as shown in Fig. 1b. If the data from node1 to node 6 and node 7 to node 12 has certain correlations, the two data structures in Fig. 1b can be used to calculate each other, we can use a similarity model to describe this relationship between two sub-structures. As shown in Fig. 1c, because the two data chunks *Chunk_1* and *Chunk_2* have a certain similarity which is described with *Distance(Chunk_1, Chunk_2)<T*. In other words, if the compression model based on data chunks can be adopted as shown in Fig. 1c, the efficiency of big sensing data compression can be dramatically increased due to data manipulation based on large data blocks and clusters.

With the above analysis, if the compression can be conducted at the level of data blocks or sensing clusters, it will greatly reduce the size of sensing data storage. At the same time, because lots of sensing clusters and large data blocks can be recovered by decompression process, huge amount of time for data searching can be saved. Motivated by those space and time saving, this paper has the following contributions. (1) Our main algorithm is used to process big sensing

data. So, some features of big sensing data will be studied and analyzed. (2) To carry out compression, the similarity between two different data chunks should be defined. So, how to define and model the similarity between data chunks is a primary requirement for data compression. (3) After the definition for the above similarity model for data chunks, how to generate those standard data chunks for future data compression is also a critical technique which we designed. (4) A novel compression algorithm is developed and designed based on our similarity model and standard data chunk generation. (5) Real meteorological big data from sensing systems will be used to test the different aspects of our proposed big sensing data compression algorithm.

## 3 Data Chunk Similarity and Compression

First, the similarity models for our compression and clustering will be developed. The similarity model is critical and fundamental for deploying the data chunk based data compression because the similarity model is used for generating the standard data chunks.

### 3.1 Similarity Model

Currently, there are five types of models are commonly used including common element approach, template models, geometric models, feature models and Geon theory. However, the following proposed models are related to geometric model and common element approach in terms of numerical data and text data respectively. Our similarity models work on two types of data sets, multi-dimensional numerical data and text data.

#### 3.1.1 Similarity Model for Numerical Data

Suppose there are two numerical data vectors $\vec{X} = (x_1, x_2, x_3, \ldots, x_n)$ and $\vec{Y} = (\mathrm{y}_1, \mathrm{y}_2, \mathrm{y}_3, \ldots, \mathrm{y}_n)$. We can calculate that the matrix norm $\|\vec{X}\|$ and $\|\vec{Y}\|$ with formula (1) and (2)

$$\|\vec{X}\| = \sqrt[2]{x_1^2 + x_2^2 + x_3^2 + \cdots + x_i^2 + \cdots + x_n^2}, \tag{1}$$

$$\|\vec{Y}\| = \sqrt[2]{y_1^2 + y_2^2 + y_3^2 + \cdots + y_i^2 + \cdots + y_n^2}. \tag{2}$$

The geometric approach is the representation of similarity relationships among the members of a set of objects. Geometric similarity is given by distance between objects in this geometric space; the closer together two objects are, the more similar they are. Normally, the similarity is described with a $\cos\theta$ between two vectors and fraction between two matrix norms $\|\vec{X}\|$ and $\|\vec{Y}\|$. Based on the above analysis, we offer two similarity definitions as follows. From formula (3) to (5), the numerical data similarity of $\theta$ is defined and denoted as $\mathrm{Sim}_{\mathrm{n1}}(\vec{X}, \vec{Y})$,

$$\mathrm{Sim}_{\mathrm{n1}}(\vec{X}, \vec{Y}) = \cos\theta, \tag{3}$$

$$\cos\theta = \frac{x_1y_1 + x_2y_2 + \cdots + x_ny_n}{\left\|\vec{X}\right\| \times \left\|\vec{Y}\right\|}, \tag{4}$$

$$\cos\theta = \frac{x_1y_1 + x_2y_2 + \cdots + x_ny_n}{\sqrt[2]{x_1^2 + x_2^2 + \cdots + x_n^2} \times \sqrt[2]{y_1^2 + y_2^2 + \cdots + y_n^2}}. \tag{5}$$

From formula (6) to (7), the numerical data similarity of matrix norms is defined and denoted as $\mathrm{Sim}_{\mathrm{n2}}(\vec{X}, \vec{Y})$,

$$\mathrm{Sim}_{\mathrm{n2}}(\vec{X}, \vec{Y}) = \frac{\left\|\vec{X}\right\|}{\left\|\vec{Y}\right\|}, \tag{6}$$

$$\mathrm{Sim}_{\mathrm{n2}}(\vec{X}, \vec{Y}) = \frac{\sqrt[2]{x_1^2 + x_2^2 + \cdots + x_n^2}}{\sqrt[2]{y_1^2 + y_2^2 + \cdots + y_n^2}}. \tag{7}$$

Considering that in most of real world applications, the preferences of applications are different. So, different weight should be assigned to different attribute for calculating the Similarities as following (8) and (9),

$$\mathrm{Sim}_{\mathrm{n1}}(\vec{X}, \vec{Y}) = \frac{w_1^2x_1y_1 + w_2^2x_2y_2 + \cdots + w_n^2x_ny_n}{\sqrt[2]{w_1^2x_1^2 + \cdots + w_n^2x_n^2} \times \sqrt[2]{w_1^2y_1^2 + \cdots + w_n^2y_n^2}}, \tag{8}$$

$$\mathrm{Sim}_{\mathrm{n2}}(\vec{X}, \vec{Y}) = \frac{\sqrt[2]{w_1^2x_1^2 + w_2^2x_2^2 + \cdots + w_n^2x_n^2}}{\sqrt[2]{w_1^2y_1^2 + w_2^2y_2^2 + \cdots + w_n^2y_n^2}}. \tag{9}$$

Based on the above definition and analysis, now we explain the influence of the above two similarities for compression.

For calculating $\mathrm{Sim}_{\mathrm{n1}}(\vec{X}, \vec{Y})$ of $\theta$, it is critical to measure the similarity between two vectors, when $\mathrm{Sim}_{\mathrm{n1}}(\vec{X}, \vec{Y}) \to 1$, the two vectors are more similar to each other. However, when $\mathrm{Sim}_{\mathrm{n2}}(\vec{X}, \vec{Y}) \to 1$, it cannot be concluded that $\vec{X}, \vec{Y}$ are similar because the two vector can be totally different.

As shown in Fig. 2a, for Similarity $\mathrm{Sim}_{\mathrm{n1}}(\vec{X}, \vec{Y})$ of $\theta$, if the $\mathrm{Sim}_{\mathrm{n1}}(\vec{X}, \vec{Y})) \to 1$, it means the $\theta \to 0$. In other words, the two vectors can be inferred with each other because their each attribute is similar. If the $\theta$ is bigger, it means the two vectors are more different as shown in Fig. 2b.

However, when it comes to $\mathrm{Sim}_{\mathrm{n2}}(\vec{X}, \vec{Y})$ of the matrix norms $\vec{X}, \vec{Y}$, even the norms of two vectors are quite similar to each other, their attributes could be totally different as shown in Fig. 2d. Whereas, even $\mathrm{Sim}_{\mathrm{n2}}(\vec{X}, \vec{Y})$ are quite different or smaller than 1, if the $\theta \to 0$ can be calculated, the two vectors can be inferred to each other as shown in Fig. 2c.

The above similarity computation can be categorized as a typical geometry data similarity finding process. It is designed from the common cosine similarity model. The cosine similarity between two vectors (or two documents on the Vector Space) is a measure that calculates the cosine of the angle $\theta$ between them. To measure similarity between two vectors $x$ and $y$, a popular similarity function is the inner product including the cosine similarity, Pearson correlations, and OLS coefficients. The inner product is unbounded. One way to make it bounded between $-1$ and 1 is to divide by the vectors' norms. It is called the cosine similarity. In this paper, we choose the cosine similarity model because it can measure the big data chunk similarity more accurately under our big sensing data feature assumption. Compare to other similarity checking model, such as similarity based on pure euclidean distance, the cosine similarity not only measures the length distance but also the angle distance between two vectors. This checking is more objective and brings more accuracy guarantee for describing the similarity according to our data assumption.

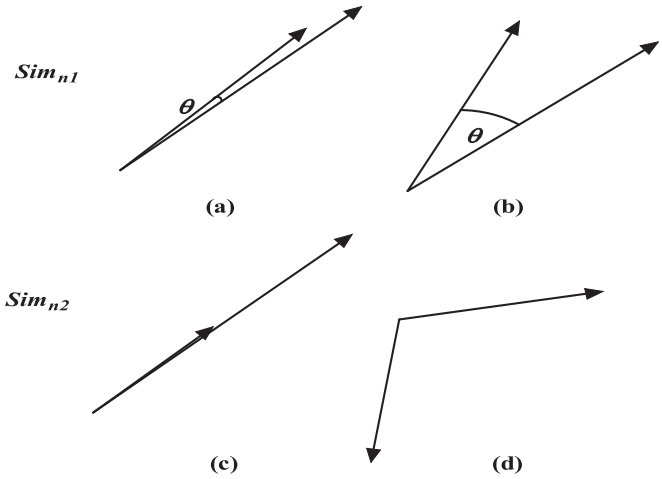



Fig. 2. Influence of two types of geometry similarities.

#### 3.1.2 Similarity Model for Text Data

For string type and text type similarity, a dual variable length hidden Markov model is used and updated in our work for calculating similarity between text data.

Suppose there are a string pair $p(str_1, str_2)$, and a time stamp series $t = t_1, t_2, \ldots, t_n$. We can define the joint probability $PR$ of each pair by the state time stamp series with formula (10). In (10), $p_i$ stands for the paper of text string with the similar time series stamp $t_i$ where $i$ is the state,

$$PR(\mathrm{p}, \mathrm{t}|\Theta) = \pi_{\mathrm{t}1} \prod_{\mathrm{i}=1}^{\mathrm{l}-1} \tau_{\mathrm{t_i t_{i+1}}} \prod_{\mathrm{i}=1}^{1} O_{\mathrm{t_i p_i}}, \tag{10}$$

$$\Theta \equiv (\pi, \{\tau_{\mathrm{s}}\}_s, \{\{\mathrm{O_s}\}_s\}). \tag{11}$$

Formula (11) is the parameters consisting of states of the initial, transition, and output probabilities,

$$MAX_{t\in\tau(\mathrm{p})} PR(\mathrm{p}, \mathrm{t}|\Theta). \tag{12}$$

Generally there are multiple state transitions that produce a given pair of strings. If the set of state sequences that produces a pair $p$ is denoted as $\tau(p)$. Then the string similarity of the pair $p$ is defined as the maximum alignment probability (12). With the transitions and states calculated with (10) to (12), the model can describe the string similarity. This compression technique can be considered as a type of common element similarity computation techniques.

#### 3.1.3 Similarity of Topology

In our data chunk based compression, data sets should be compression block by block. For big graph data and lots of network data, the topology and structure information has big influence for data processing and it should not be ignored. Because we assume the data has a topology of leaf nodes and cluster head, the similarity has following features.

For two tree topology based graphs, $T_1 <V_1, E_1>$ and $T_2 <V_2, E_2>$, their topology similarity is basically determined by the number of the leaf nodes. Based on the cluster head topology, the $T <V, E>$ has n $v$ and n-1 $e$.

If $T_1 <V_1, E_1>$ has m $v$ and m-1 $e$, and $T_2 <V_2, E_2>$ has n $v$ and n-1 $e$, the comparison value of similarity is calculated with following formulas (13) and (14),

$$\mathrm{Sim_V}(\overrightarrow{T_1}, \overrightarrow{T_2}) = \frac{|m-n|}{Max(m,n)}, \tag{13}$$

$$\mathrm{Sim_E}(\overrightarrow{T_1}, \overrightarrow{T_2}) = \frac{|m-n|}{Max(m-1,n-1)}. \tag{14}$$

In formulas (13) and (14), |m-n| calculates the different of $v$ between two trees. On the contrary, Max selects the bigger number between (m, n) and (m-1, n-1). When comparing the two vectors $\mathrm{Sim_{n1}}(\vec{X}, \vec{Y})$, the calculated $\mathrm{Sim_V}(\overrightarrow{T_1}, \overrightarrow{T_2})$ and $\mathrm{Sim_E}(\overrightarrow{T_1}, \overrightarrow{T_2})$ will be added into as new attributes with a weight offered by application requirement or users to calculate overall similarities.

### 3.2 Data Chunk Generation and Formation

With the above definition of similarity model, we will give the techniques for data chunk generation. In the problem analysis, we have introduced the basic idea of data chunk based compression. Under that theme, the data will not be compressed by encoding or data prediction one by one. It is similar to high frequent element compression. The difference is that the frequent element compression recognizes only simple data units; whereas our data chunk based compression recognizes complex data partitions and patterns during the compression process. Similar to chess games, variations and patterns are well studied and predefined, and most of operations will happen at variation level.

Suppose that a data unit in a big data set S is denoted as $\mathrm{X_i}$, and the first data unit should be denoted as $\mathrm{X_1}$. In other words, S is a stream data series, denoted as $\mathrm{S}=\{\mathrm{X_1}, \mathrm{X_2}, \ldots, \mathrm{X_m}\}$, where m is the number of data units streaming in. If the following data unit vectors $\mathrm{X_2}$ and $\mathrm{X_1}$, can be used to calculated $\mathrm{Sim_{n1}}(\vec{X_1}, \vec{X_2})$ and $\mathrm{Sim_{n2}}(\vec{X_1}, \vec{X_2})$. At the same time, if it is text type data, the dual variable length hidden Markov model will be used to calculate similarity. Then, the topology similarities, $\mathrm{Sim_V}(\overrightarrow{T_1}, \overrightarrow{T_2})$ and $\mathrm{Sim_E}(\overrightarrow{T_1}, \overrightarrow{T_2})$ will be calculated and added into original vectors as new attributes. With the big data set preprocess, we aim to generate a standard initial set S′ which is used in the following compression process. According to our predefined similarity model, we need to generate the first initial standard data chunk set S′ for the big data set S. We denote each time slot for calculating a new standard data chunk as t, and $\mathrm{t} \in (1, +\infty)$. The selection process of S′ forms a selection path.

As shown in Fig. 3, at $\mathrm{t} = 1$, $\mathrm{X_1}$ is selected and set as the first standard data chunk and added into S′. Following that, at $\mathrm{t} = 2$, $\mathrm{X_2}$ will be compared to $\mathrm{X_1}$ based on their similarity and the application specified threshold T. Here, we use a distance function $\mathrm{Dis}(\overrightarrow{X_1}, \overrightarrow{X_2})$ is used to describe the difference between any two vectors. If the result of $\mathrm{Dis}(\overrightarrow{X_1}, \overrightarrow{X_2}) > \mathrm{T}$ can be calculated, it means that the two chunks, $\mathrm{X_1}$ and $\mathrm{X_2}$ are quite different two each other. Then a newly generated data chunk $\mathrm{X_1} + \mathrm{X_2}$ should be added to S′ as the second standard data chunk. The above ' + ' operator between $\mathrm{X_1}$ and $\mathrm{X_2}$ means an adding operation between two vectors. At the time stamp, S′ has two elements, denoted as $\mathrm{S'} = \{\mathrm{X_1}, \mathrm{X_1} + \mathrm{X_2}\}$. On the contrary, at $\mathrm{t} = 2$, if $\mathrm{Dis}(\overrightarrow{X_1}, \overrightarrow{X_2}) \leq \mathrm{T}$ can be got, it means $\mathrm{X_1}$ and $\mathrm{X_2}$ are close enough and can be used to infer each other. Then, $\mathrm{X_2}$ will be

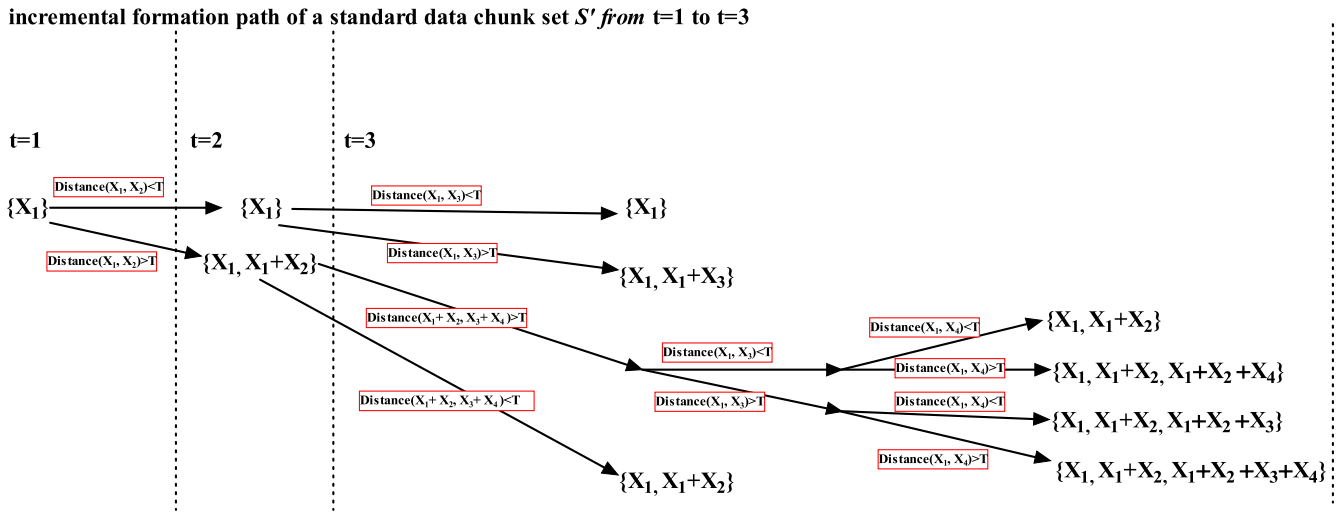



Fig. 3. The initial formation path for a standard data chunk set S'.

discarded and all the $X_2$ will be inferred from $X_1$ during the big data set processing. So, at the end of t = 2, there are two possible states for S′ including $\{X_1\}$ and $\{X_1, X_1 + X_2\}$. In other words, there two paths connecting the possible states at t = 1 to states at t = 2. At t = 3, there are seven possible states and related paths evolved from the states at t = 2. To find a similarity data chunk, we always use the largest data chunk for comparison at first. For example, in Fig. 3, at t = 3, at data block $X_3 + X_4$ is selected first to be compared with block $X_1 + X_2$. If $\mathrm{Dis}(\overrightarrow{X_1} + \overrightarrow{X_2}, \overrightarrow{X_3} + \overrightarrow{X_4}) < \mathrm{T}$ can be calculated, the data block of $X_3$ and $X_4$ will be compressed and represented by $X_1$ and $X_2$. If $\mathrm{Dis}(\overrightarrow{X_1} + \overrightarrow{X_2}, \overrightarrow{X_3} + \overrightarrow{X_4}) > \mathrm{T}$ is calculated, the sub-components of data block $X_3$ and $X_4$ will be calculated to be compared with the standard data chunks already generated at t = 2. Suppose that the length of a coming data block is denoted as $L$. The length of the data block of $X_3$ to $X_4$ $L_{3-4} = 2$. There are $C_2^1 = 2$ sub-blocks $X_3$ and $X_4$. The sub-blocks $X_3$ and $X_4$ will be compared to $X_1$. With this data sub-block comparison, at following t > 3, the data decomposing and sub-block will be conducted recursively for calculating any standard data chunk at t = i as shown in Fig. 4.

Specifically in Fig. 4, there are a series of possible standard chunks at t = i. The original length of coming data is $L$ = i. In the similarity comparing process, If any "Dis()> T" is calculated, there are totally $C_L^1 + C_L^2 + C_L^3 + \ldots + C_L^{L-1} + C_L^L = 2^L - 1$ sub-blocks which probably should be compared to the standard data chunks generated at t = i − 1 in a recursive style.

The last problem is the termination of the above standard data chunk generation process. With the processing of incoming data, suppose that the original selected data length from L at time t = i. If after r rounds, when it comes to t = i + r, there is no new change added to the S′, the S′ is used as the final standard data chunks set for data compression in future. The time round 'r' is set or offered by outside application requirement. The size of the standard data chunk is controlled by selecting the parameter 'r' which determines when to terminate the recursive process of generating new data chunks. As a matter of fact, the generated standard data chunks after initial selection have different size to each other. In this data chunk generation, the recursive process will increase the size of selected standard data chunks step by step. If the terminating condition is not reached, the algorithm will continuously find new standard chunks with bigger size compared to any standard chunk which has already been selected. When the terminating condition is reached according to the given 'r', the final selected standard data chunk is always the largest one in terms of size.

### 3.3 Data Chunk Based Big Data Compression

With the generated standard data chunks, we develop a new data compression technique which recursively compresses in-coming data from big data set S according to generated S′. Suppose that the ith vector in S is denoted as $u_i$ and the jth vector in S′ is denoted as $v_j$.

As shown in Fig. 5, to compress the big data set S from vector $u_j$, there is no need for the compression algorithm to navigate $u_j$ one by one. Whereas, the standard chunks stored in S′ will be used to compress the in-coming vectors series $u_j$ chunks by chunks. For example, with the generated standard chunks set S′, a whole block of data $u_j$ to $u_{j+r}$ will be compared to $v_r$ in S′ first. If the distance between $v_r$ and $u_{j+r}$, $\mathrm{Dis}(\mathrm{vr}, \{u_j, \ldots, u_{j+r}\}) > \mathrm{T}$ can be calculated, the $u_{j+r}$ will be recursively decomposed with the sequence of subsets from $\{u_j, u_{j+1}, u_{j+2}, \ldots, u_{j+r}\}$. Totally, there are $C_r^1, C_r^2, C_r^3, \ldots, C_r^{r-1}, C_r^r$, different subsets based on the vector set $\{u_j, u_{j+1}, u_{j+2}, \ldots, u_{j+r}\}$. These subsets will have opportunity to be compared with $v_{r-1}$ to $v_1$ repectively to detect some similar data chunks in the data block $\{u_j, \ldots, u_{j+r}\}$. This recursive processing for computing $\mathrm{Dis}(v_k, \{u_j, \ldots, u_{j+r}\})$ will only happening under the following conditions.

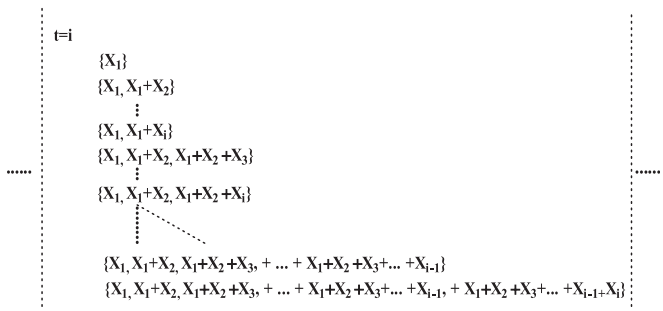



Fig. 4. The possible data chunk states at $t = i$.

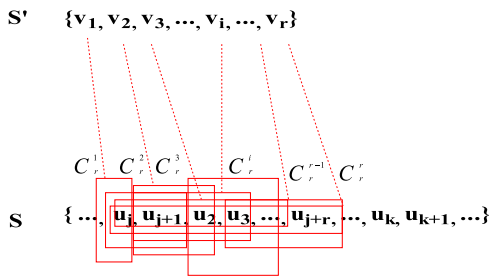



Fig. 5. The possible data chunk states at $t = i$.

(1) $Dis(v_r, \{u_j, \ldots, u_{j+r}\}) < T$: It means that $v_r$ and $\{u_j, \ldots, u_{j+r}\}$ are similar to each other and $v_r$ can be used to compress the data block $\{u_j, \ldots, u_{j+r}\}$.

(2) $Dis(v_1, \{u_1\}) < T$, $Dis(v_k, \{u_k, \ldots, u_{j+l}\}) < T, \ldots,$ $Dis(v_p, \{u_p, \ldots, u_{j+p}\}) < T$; $\{v_1 \cup \cdots \cup v_k \cup \cdots \cup v_p\} = S'$ & $\{u_1\} \cup \cdots \cup \{u_k, \ldots, u_{j+l}\} \cup \cdots \cup \{u_p, \ldots, u_{j+p}\} = S$: It means that all the subsets in $\{u_j, \ldots, u_{j+r}\}$ can be compressed by vectors in $S'$.

(3) $Dis(v_1, \{u_1\}) > T$, $Dis(v_k, \{u_k, \ldots, u_{j+l}\}) > T, \ldots,$ $Dis(v_p, \{u_p, \ldots, u_{j+p}\}) > T$, $Dis(v_1, u_{j+1}) > T$, $Dis(v_2, u_{j+2}) > T, \ldots,$ $Dis(v_k, u_{j+k}) > T, \ldots$ $Dis(v_r, u_{j+r}) > T$: It means that in the set and subsets of data block $\{u_j, \ldots, u_{j+r}\}$ there is no similarity data chunk which can be found in $S'$. In other words, no compression happens here and data block should be stored.

(4) $Dis(v_1, \{u_1\}) > T$, $Dis(v_k, \{u_k, \ldots, u_{j+l}\}) > T, \ldots,$ $Dis(v_p, \{u_p, \ldots, u_{j+p}\}) > T$, $Dis(v_1, u_{j+1}) > T$, $Dis(v_2, u_{j+2}) > T, \ldots,$ $Dis(v_k, u_{j+k}) > T, \ldots$ $Dis(v_r, u_{j+r}) > T$; $Dis(v_2, \{u_2\}) < T$, $Dis(v_{k+1}, \{u_{k+1}, \ldots, u_{j+l}\}) < T, \ldots,$ $Dis(v_{p+1}, \{u_{p+1}, \ldots, u_{j+p+1}\}) < T$, $Dis(v_2, u_{j+2}) < T$, $Dis(v_2, u_{j+2}) < T, \ldots,$ $Dis(v_{k+1}, u_{j+k+1}) < T, \ldots$ $Dis(v_{r-1}, u_{j+r-1}) < T$: It means that some vector subsets of data block $\{u_j, \ldots, u_{j+r}\}$ can be compressed with the elements $v_i$ from $S'$; and others should be stored because there is no similar $v_i$ being matched in $S'$.

With the formed $S'$ and a recursive process, the big sensing data stream will be compressed for both space and time efficiency.

## 4 Scalable Algorithms for Data Chunk Similarity Based Compression on Cloud

To deploy our proposed big sensing data compression on Cloud, two important stages, standard chunk generation and chunk based compression are essential. So the algorithms are developed respectively to conduct the related data processing for the above two stages.

At the first stage, the standard data chunks are generated. The algorithm for selecting those chunks can be performed before the real data compression by centralized computer systems. So, a centralized algorithm is offered for describing the whole process of standard data chunk generation. At the second stage of big data compression, the storage and time saving is mainly achieved by chunk based compression and scalability of Cloud. The chunk based compression is introduced by the algorithm itself, and the scalability is introduced by designing the compression algorithm with MapReduce. In other words, the compression algorithms conclude two parts, "Mapper" side algorithm and "Reducer" side algorithm. In following content of this Section 4, all the above algorithms will be offered and analysed.

### 4.1 MapReduce and Compression Algorithm

To guarantee the scalability of the proposed data compression algorithm based on data chunks, MapReduce programming model and Hadoop platform are adopted for implementation.

#### 4.1.1 MapReduce

MapReduce is a framework for processing parallelizable and scalable problems across huge datasets using a large number of computers (nodes), collectively referred to as a cluster (if all nodes are on the same local network and use similar hardware) or a grid (if the nodes are shared across geographically and administratively distributed systems, and use more heterogenous hardware). Computational processing can occur on data stored either in a filesystem (unstructured) or in a database (structured). MapReduce can take advantage of locality of data, processing data on or near the storage assets to reduce data transmission. "Map" function: The master node takes the input, divides it into smaller sub-problems, and distributes them to worker nodes. A worker node may do this again in turn, leading to a multi-level tree structure. The worker node processes the smaller problem, and passes the answer back to its master node. "Reduce" function: The master node then collects the answers to all the sub-problems and combines them in some way to form the output – the answer to the problem it was originally trying to solve. MapReduce allows for distributed processing of the map and reduction operations.

#### 4.1.2 Principle for Designing MapReduce Algorithm

However, traditional MapReduce is very strict, which limits its application in complex systems, such as meteorology sensing systems. Sometimes, it is hard to directly use one MapReduce to solve a data processing issue perfectly. In other words we have to revise the MapReduce or use its functionalitis alternatively.

Based on our knowledge for MapReduce and its wide applications, three technical changes are commonly adopted to transform the targeting problem for applying MapReduce on our proposed data chunk compression algorithms. With those transformation techniques, a data processing issue can be changed or partially changed into several scalable or centralized parts. The MapReduce programming model will be applied on those scalable parts.

(1) Original algorithm $->$ (embedded in) **Map()/Reduce()**.

(2) Partition the task flow of algorithm $->$ Identify which part of the task flow to generate a MapReduce job $->$ MapReduce generated result returns back to the task flow.

(3) Complete MapReduce design $->$ control flow parallelization/ data parallelization/ flow scalability/ data scalability.

Based on the analysis of the above three strategies and the complicated flow of our data chunk compression algorithms, in our implementation, we adopt different MapReduce strategies in terms of different control flow, data navigation, data comparison for data compression and storage process.

In our work, the compression algorithm is embedded into the Mapper and Reducer, and to check the soundness of the semantics of the newly generated scalable data chunk compression algorithm with MapReduce. Specifically, the flow of our compression MapReduce programming model can be

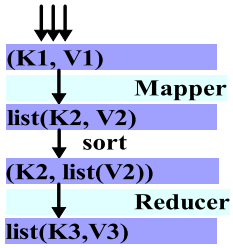



Fig. 6. Programming model of MapReduce.

described with KV pairs as shown in Fig. 6. The output of Mapper goes through shuffle and sort; then it becomes the input of the Reducer. So, the compression algorithms should be embedded in both Mapper and Reducer stages. In addition, the shuffle and sort should be further reformed to fit the data chunk partitioning and compression algorithms. With the above analysis, we offer the following detailed algorithms for data chunks generation and data compression.

### 4.2 Standard Data Chunks Generation Algorithm

Before the start of data compression, a standard data chunks set will be generated based on our defined similarity model. This process is a preprocessing before the compression which normally does not require huge computation resource. So, a centralized algorithm is offered first. After this centralized algorithm, the scalable compression algorithm based on MapReduce will be offered.

In the standard data chunks generation algorithm, there are two important inputs big sensing data set S and the maximum limitation 'r' for data generation control. The output of this algorithm is a data chunk set S′ which is a subset of S containing all the generated standard data chunks for future data compression.

From line (1) to line (4), the initialization process is conducted including S′ and its first element $v_1$, a combined data type $X$ which is used for temporal storing vector data elements from S. From line (5) to line (9), the similarity mode is calculated and selected according to application requirement. Specifically, the algorithm from line (5) to line (6) is used for choosing the processing model of numerical data type vectors. The algorithm between from (7) to line (8) is used for choosing the processing model of text data type vectors. In line (9) the topology information of a selected data vector is calculated and attached into the data vector as numerical attributes with the transformation computation parameter offered in Section 3.1.3. In line (10), the first element, $x_1$ in big data set S is selected as the first element in the standard data chunks set S′. At the same time, the length of the S′ (number of elements in S′) $l$ is set as 1 in line (11). From line (12) to line (28), the algorithm is designed for recursively calculating standard data chunks.

Specifically, line (12) is the maximum rounds limitation which controls the ending condition for generating new standard data chunks recursively. In line (13), the temporal variable $X$ is set at the first time. In line (14), the similarity model is first calculated to compare the similarity between $X$ and $v_i$. If the two vectors, $X$ and $v_i$ are similar enough, $X$ will be replace with $v_i$; whereas if the distance between $X$ and $v_i$ are larger an offered threshold, $X$ will be decomposed recursively to compare with other elements in standard data chunks set S′. From line (18) to line (23), the recursive similarity comparison function $C(v_i, X)$ is called. The details of $C(v_i, X)$ starts in line (29) as follows. In line 30, the termination condition of a for() loop is offered. This loop will terminate only when there is no similar $v_i$ in S′ can be found, or $X$ is decomposed by similar chunks in S′. Between line (32) and line (35), the recursive function $C(v_i, X)$ is called to find any data subset within $X$ which could be similar to any $v_i$ in S′. After the recursive function call of $C(v_i, X)$, the algorithm return to line (24) and line (25) which generate a new standard data chunk $v_i$ and add it into the standard data chunks set S′. Suppose that $L$ is the size of the formed standard data chunk S′. The worst case complexity of the algorithm O ($L^3$) can be calculated due to recursive decomposition and similarity checking in several rounds. However, $L$ is a very small and ignorable value compared to the size of incoming big sensing data set and it will not change in the following compression process. So, in the following compression algorithm, it can be viewed as a constant when analyzing the algorithm complexity.

**Algorithm.** Standard Data Chunks Generation

**input**: a streaming big sensing data set $S = \{x_1, x_2, \ldots, x_n\}$;
maximum time threshold for chunk evolvement: $r$
**Output**: Standard data chunk set S′ *which is a subset of* $S$; $S' = \{v_1, v_2, \ldots v_k\}$

(1) public void main ($S = \{x_1, x_2, \ldots, x_n\}$, int $r$) throws IOException {
(2) initialize $S' = \emptyset$;
(3) initialize $(v_1, x_1)$//$v_1 = x_1$ *is used for initializing* $v_1$;
(4) initialize $X$; // *a temporary variable for storing recursively selection from* $\{x_i, \ldots, x_j\}$;
(5) if(mode.equal(numerical_data))
(6) {Distance()} = {$\text{Sim}_{n1}$(,), $\text{Sim}_{n2}$(,)};
(7) if(mode.equal(text_data))
(8) {Distance()} = {$\text{Sim}_{n1}$(,), $\text{Sim}_{n2}$(,), Θ};
(9) Attach(Distance(), $\text{Sim}_V$(,), Θ, $\text{Sim}_E$(,));
(10) $S' = x_1 \rightarrow S'$; //$x_1$ *is the first elemet in* S′;
(11) initialize $l = 1$; // *the maximum elemet length in* S′ *is* $l$;
(12) for (int *round* $= 1$; *round* $<= r$; *round* $++$) {
(13) $X$ = select (*xi*, *l*);
(14) if (Distance(*vi*, $X$)<Threshold)
(15) $i = i + l$;
(16) else{
(17) *i*– –;
(18) for(;i! = 0&&Distance($v_i$, $X$)>Threshold){
(19) Distance(,) = C($v_i$, $X$);
(20) if (Distance(,)<Threshold)
(21) break;
(22) extract($X'$, $X$);
(23) }
(24) $v_{i+1} = v_i + X'$;
(25) $S' = v_{i+1} \rightarrow S'$;
(26) }
(27) }
(28) }

**Recursive Similarity Comparison Function:**

(29) $C(v_i, X)$ {
(30) for(int i=1;(Distance($v_i$,$X$)<=Threshold)&&length(X)>1;){
(31) *i*–;
(32) for(int j = 1;j>1;j–){
(33) X = $C_X^{X-j}$;
(34) C($v_j$, $X$);
(35) }
(36) }
(37) }

#### 4.2.1 Compression Algorithm: "Map()" Side

With the generated standard data chunks set S′ in the above Section 4.2, the scalable compression algorithm based on MapReduce programming model is offered as follows. The algorithm is divided into two components including Mapper side compression algorithm and Reducer side compression algorithm. First, we introduce our Mapper side algorithm.

"Map()" Side Algorithm. Scalable Compression with Data Chunk Similarity

```
(1)  public static class Mapper extends TableMapper <……, ……> {
(2)  public Mapper() {}
(3)  @Override
(4)  public Datatype map(Datatype S = {x1, x2, ..., xn}, Datatype S′ = {v1, v2, ... vk}.)
(5)  throws IOException {
(6)    ImmutableBytesWritable value = null;
(7)    initialize X; // a temporary variable for storing recursively selection from {xi, ..., xj};
(8)    if(mode.equal(mumerical_data))
(9)      Compression.set(Sim_n1(,), Sim_n2(,));
(10)   if(mode.equal(text_data))
(11)     Compression.set(Sim_n1(,), Sim_n2(,), Θ);
(12)   Compression.set(Sim_n1(,), Sim_n2(,), Θ, Sim_E(,), Sim_V(,));
(13)   L = MaxElementSizeof(S′); int start = 0;
(14)   for(; S! = ø; start = start + L){
(15)     X = S.getlement(start, L);
(16)     for(int j = L; j>0; j–){
(17)       C(v_j, X);
(18)       tag(X.Distance<Threshold);
(19)     }
(20)   }
(21)   return S; // a tagged data set S for final compression;
(22)   for (int i = 0; i < S′.length; i + +) {
(29)   try {
(24)     context.write(compressionID,value);
(25)   } catch (InterruptedException e) {
(26)     throw new IOException(e);
(27)     }
(28)   }
(29) }
```

The Mapper side algorithm takes the S and S′ as its input. The output of Mapper side algorithm is a data set S which its data element tagged. All the tagged elements are able to be compressed and decompressed based on S′. Specifically, the Mapper function is the extension of TableMapper as shown in line (1) and line (2). In line (4)), the map() function is initialized and defined. It has S and S′ as its inputs. Line (5) is the IO exception processing. In line (6) and line (7), some variables are initialized. From line (8) to line (12, the compression data model is selected and configured. The algorithm from line (8) to line (9) is used for choosing the processing model of numerical data type vectors. The algorithm between from (10) to line (11) is used for choosing the processing model of text data type vectors. In line (12), the topology information of a selected data vector is calculated and attached into the data vector as numerical attributes with the parameter introduced in Section 3.1.3. In line (13), the total number of elements in the standard data chunks set, S′ is calculated and stored in $L$. Because during the compression process, the data vectors from S is selected chunk by chunk with the length $L$, the algorithm needs to record the starting point of the data element, denoted as start $= 0$ in line (13). From line (14) to line (20), the recursive similarity comparison function is called again to tag any data element in S to find any $x_i \in$ S which could be compressed. After line (20), the data elements which could be compressed are tagged in map() function. In line (21), the tagged big data set S is returned and separated within map() function for distribution. From line (22) to line (28), the IO exceptions and errors are processed and captured for debugging. The worst case complexity of the algorithm is O ($n \times L^3$), $L \ll n$ where $n$ is the size of the big data set, $L$ is a small number for the size of the standard data chunk set. As introduced in Section 4.2, after the formation of standard data chunk set, $L^3$ can be treated as a constant during the compression process. The worst case algorithm complexity is O ($n$).

#### 4.2.2 Compression Algorithm: "Reduce()" Side

After the processing of map() function of our proposed algorithm, the tagged big data set S should be compressed and calculated for final data compression result.

"Reduce()" Side Algorithm. Scalable Compression with Data Chunk Similarity

```
(1)  public static class Mapper extends TableReducer <……, ……> {
(2)  public Reducer(){}
(3)  @Override
(4)  public void reduce(Datatype S)
(5)  throws IOException {
(6)    ImmutableBytesWritable value = null;
(7)    Compression.set(Sim_n1(,), Sim_n2(,), Θ, Sim_E(,), Sim_V(,));
(8)    for(int i = 0; S.getelement.tag()! = ø; i + +){
(9)      if(S.getlement(i).tag()! = ø){
(10)       S.getlement(i).compress();
(11)       S.update(storage); S.index(decompression path);
(12)     }
(13)   }
(14)   S.combine(); S.consistencycheck();
(15)   return S;
(16)   for (int i = 0; i < S.length; i + +) {
(17)   try {
(18)     context.write(elementID,value);
(19)   } catch (InterruptedException e) {
(20)     throw new IOException(e);
(21)     }
(22)   }
(23) }
```

Our "Reduce()" side scalable compression algorithm extends the *TableRedcuer<>* of MapReduce programming model as shown in the algorithm line (1) to line (3). In the algorithmic line (4), the reduce() function is initialized and defined. The redcue() function takes tagged big data set, S as its input as shown in line (4). Line (5) is the IO exception and error processing. In line (6) and line (7), variable initialization and compression model selection are conducted. From line (8) to line (10), any tagged data element in S is compressed by the compress() function of any element in S. After each function call of compress(), the storage should be

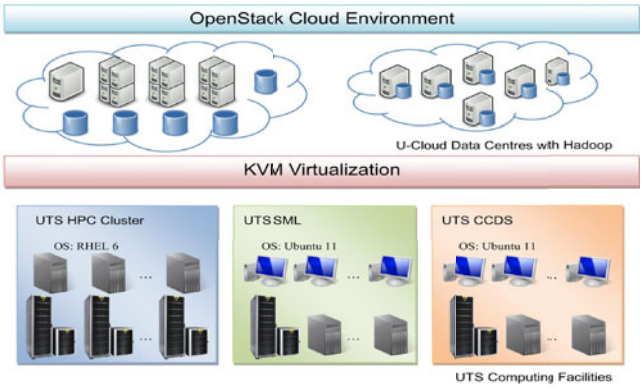



Fig. 7. U-cloud system overview.

updated by a function of update() and the compression path should be indexed for future decompression by a function of index() as show in line (11). In line (14), the combine() function is called for centralization. Furthermore, the consistency of the compressed data is also checked. Finally, the compressed data set S is returned by the algorithm in line (15). Form line (16) to line (22), the IO errors and exceptions are detected and captured.

With the above three algorithms, we implement a big sensing data compression system based on calculating similarity between data chunks. With that system, we can process real big sensing data sets to test the effectiveness and efficiency of the above algorithms.

## 5 Experiments

To verify the time efficiency and the effectiveness of our approach for compressing big sensing data on Cloud, experiments are conducted on U-Could (Cloud computing environment at the University of Technology Sydney) [11], [12], [13], [14], [30]. There are three purposes for this experiment. 1) Demonstrate that the significant storage saving is achieved due to compressed data blocks. 2) Demonstrate that the significant time saving is achieved because lots of real big data blocks can be inferred instead of real search and navigation. 3) Compared to significant time and space performance gains, only tiny data loss is introduced in terms of accuracy.

### 5.1 Experiment Environment and Process

The U-Cloud system is set up as shown in Fig. 7. On top of hardware and Linux operating system, KVM virtualization software is installed for the virtualization of infrastructure and providing unified computing and storage resources. To create virtualized data centres, we install OpenStack Cloud environment which is responsible for virtual machine management, resource scheduling, task distribution and user interaction. Furthermore, Hadoop [40] is installed to facilitate MapReduce computing paradigm and big data processing.

#### 5.1.1 Meteorology Big Data Set

In our experiment, the world meteorology big sensing data sets are used. In the civil network meteorology data set, there are four types of commonly used data formats, including GRIB, BURF, HDF and NetCDF as shown in Table 1. Due to the different data formats, before conducting our experiment, a series of parsers are implemented to preprocess meteorological big sensing data sets from the following civil open data source. After our preprocessing, all the meteorology data sets with different data formats are transformed into our uniform data format for further data compression. Specifically, four kinds of meteorology data sets are accessed in open civil meteorology sources in different data format as follows.

TABLE 1
Meteorology Data Formats

| Format | versions | coding type | feature |
|---|---|---|---|
| GRIB | multiple | binary | self described, compressed |
| BURF | 1 | binary | self described |
| HDF | multiple | multi-objects | scientific data, independent |
| NetCDF | 1 | multi-attributes variables | 6 data types, data files |

(1) **Sea Surface Temperature Data Sources (SST)**:
*ftp://polar.ncep.noaa.gov/pub/cdas/eng.YYYYMMDD;*
*ftp://polar.ncep.noaa.gov/pub/sst/rtg_sst_grb_0.5.YYYYMMDD;*

(2) **Satellite Coverage Rate Data Sources**:
*ftp://ftpprd.ncep.noaa.gov/pub/data/nccf/com/gfs/prod/gdas.YYYYMMDD/gdas1.t06z.1bmhs.tm00.bufr_d;*
*ftp://ftpprd.ncep.noaa.gov/pub/data/nccf/com/gfs/prod/gdas.YYYYMMDD/gdas1.t12z.1bamua.tm00.bufr_d;*
*ftp://ftpprd.ncep.noaa.gov/pub/data/nccf/com/gfs/prod/gdas.YYYYMMDD/gdas1.t12z.1bhrs3.tm00.bufr_d;*
*ftp://ftpprd.ncep.noaa.gov/pub/data/nccf/com/gfs/prod/gdas.YYYYMMDD/gdas1.t12z.1bhrs4.tm00.bufr_d;*

(3) **Wireless Electrical Mask Satellite Observatory Data Sources**:
*ftp://ftpprd.ncep.noaa.gov/pub/data/nccf/com/gfs/prod/gdas.YYYYMMDD/gdas1.t00z.gpsro.tm00.bufr_d;*
*ftp://ftpprd.ncep.noaa.gov/pub/data/nccf/com/gfs/prod/gdas.YYYYMMDD/gdas1.t06z.gpsro.tm00.bufr_d;*
*ftp://ftpprd.ncep.noaa.gov/pub/data/nccf/com/gfs/prod/gdas.YYYYMMDD/gdas1.t12z.gpsro.tm00.bufr_d;*
*ftp://ftpprd.ncep.noaa.gov/pub/data/nccf/com/gfs/prod/gdas.YYYYMMDD/gdas1.t18z.gpsro.tm00.bufr_d;*

(4) **Satellite wind observatory data sources:**
*ftp://ftpprd.ncep.noaa.gov/pub/data/nccf/com/gfs/prod/gdas.YYYYMMDD/gdas1.t00z.satwnd.tm00.bufr_d.unblok*
*ftp://ftpprd.ncep.noaa.gov/pub/data/nccf/com/gfs/prod/gdas.YYYYMMDD/gdas1.t06z.satwnd.tm00.bufr_d.unblok*
*ftp://ftpprd.ncep.noaa.gov/pub/data/nccf/com/gfs/prod/gdas.YYYYMMDD/gdas1.t12z.satwnd.tm00.bufr_d.unblok*

More specifically, from the open big meteorology sensing data sources, the big meteorology data set collected around East Longitude 151°12′31′′ and South Latitude 33°52′06′′ is used. The coverage radius is around 50 km. The time for data sets stamps is traced back to last 20 years. Totally, around 10 terabytes of meteorology sensing data were gathered and downloaded for testing our data chunks similarity based compression algorithms.

Whatever the data format is, with our data parsers and their offered normalization, four types of data attributes in the above data sets are extracted and organized again into a universal data format including temperature, atmosphere pressure, humidity and wind speed. Specifically, the

numerical temperature is measured with Celsius degree. The atmosphere pressure is measured with kPa. The humidity is measured by relative humidity to calculate a percentage value in '%'. The wind speed is measured by two parameters. One is the m/s, the other is angle for wind direction. Each meteorological sensing data set has the approximate size of 2.5 terabytes respectively.

#### 5.1.2 General Comparison and Analysis

First, according to the above experimental data analysis, on average, each type of meteorological data has the size of around 2.5 terabytes. So, after processed by our compression algorithm, different compression ratios will be shown in terms of different meteorological data sets. By analyzing and concluding the data features of those data sets, we aim to show the advantages and disadvantages of our proposed data chunk compression algorithm. In addition to the compression ratio brought in each data type, the overall compression effectiveness over the whole data set (10 terabytes) will also be analyzed to demonstrate the performance gains of our data chunks similarity based data compression.

Second, under the theme of big data storage on Cloud platform, to reduce the data size also means the time saving for navigating and decompressing those data units. Instead, fast computation can be used for data restoration can complex data manipulation such as distributed join and block data operation. Through the experiment, we also want to demonstrate that the time and space saving can dramatically contribute to big sensing data processing performance on Cloud.

Third, even the main designing target of our compression algorithm based on data chunks similarity is to reduce the data size and volume, we also consider the data quality and fidelity loss after deploying our proposed compression and decompression. Because the compression is based on a certain data similarity model, it is unavoidable to bring errors and approximation to original data sets. However, those approximation and errors should be kept within an acceptable range in terms of most application requirement.

### 5.2 Space and Time Saving from Compression

The main development purpose of our compression algorithm based on data chunks similarity is to reduce the volume of data, hence to save data storage and related data operation time cost on Cloud. As shown in Fig. 8, four data types are used for testing our compression algorithm. Specifically, in Fig. 8(1a), the limitation rounds R for generating standard data chunks set is offered as 50. With the time flow from 0 to 24 hours, the compressed temperature data size increase from 0 terabyte to around 1.2 terabytes. However, when R increases to 100 rounds, the compression ratio of the temperature data experiment reaches to around 1.7 terabytes after 24 hours processing as shown in Fig. 8(1b). It means that with the increase of R, more standard data chunks are generated, higher opportunity exists to compress more data blocks in the testing big temperature data set.

In Fig. 8(2a), the limitation rounds R for generating standard data chunks set is offered as 50. With the time flow from 0 to 24 hours, the compressed pressure data size increase from 0 terabyte to around 1.6 terabytes. However, when R increases to 100 rounds, the compression ratio of the pressure data experiment reaches to around 2.0 terabytes after 24 hours processing as shown in Fig. 8(2b). It means that with the increase of R, more standard data chunks are generated, higher opportunity exists to compress more data blocks in the testing big atmosphere pressure data set.

In Fig. 8(3a), the limitation rounds R for generating standard data chunks set is offered as 50. With the time flow from 0 to 24 hours, the compressed relative humidity data size increase from 0 terabyte to around 0.6 terabytes. However, when R increases to 100 rounds, the compression ratio of the humidity data experiment reaches to around 1.1 terabytes after 24 hours processing as shown in Fig. 8(3b). It means that with the increase of R, more standard data chunks are generated, higher opportunity exists to compress more data blocks in the testing big humidity data set. In Fig. 8(4a), the limitation rounds R for generating standard data chunks set is offered as 50. With the time flow from 0 to 24 hours, the compressed wind speed data size increase from 0 terabyte to around 1.0 terabytes. With the time flow from 0 to 24 hours, the compressed wind speed data size increase from 0 terabyte to around 1.0 terabytes. However, when R increases to 100 rounds, the compression ratio of the wind speed data experiment reaches to around 1.3 terabytes after 24 hours processing as shown in Fig. 8(4b). It means that with the increase of R, more standard data chunks are generated, higher opportunity exists to compress more data blocks in the testing big wind speed data set. Based on the above experiment result comparison, it can be got that with the increase of R, the compression ratio increases under all data experiments whatever data type is. It should be noticed that when applying our compression algorithm, more effectiveness or larger compression ratio can be achieved in processing the temperature and pressure data sets compared to the humidity and wind speed data sets. The reason is that the temperature and pressure data set have relatively predictable data trends in their time series; whereas the humidity and wind speed data sets have more unpredictable changes in their time series. R is also has great impact on the data inference for data decompression. In principle, a much larger value is set for R, more standard data chunks will be selected. Hence, more data will be compressed. At the same time, during the process of decompression.

In terms of time saving, it can be inferred indirectly from the compression experiment according the results with the increase of R from 50 to 100 rounds limitation in Fig. 8. Specifically, after setting a data compression size target, we compare the time cost for suppressing that data size when using different data processing strategies. Under our proposed compression, more time can be saved when successfully compressing the same amount of meteorological sensing data including temperature, pressure, humidity and wind speed data as shown in Fig. 8. In other words, the compression and big data processing time cost can be dramatically reduced. With the result analysis of Fig. 8, it can be concluded that significant performance gains are achieved in terms of space and time cost saving.

In Fig. 9, the overall compression effectiveness of our proposed data compression is demonstrated. The $X$ axis stands for the incoming big sensing data size for testing. The $Y$ axis stands for the compression achieved by deploying our compression algorithm. There are two important testing findings should be indicated. (1) With the increase of R from 10 to 90 rounds, the compression ratio increases whatever the testing data size is from 1 to 10 terabytes. In other words, the larger R brings more compression ratio

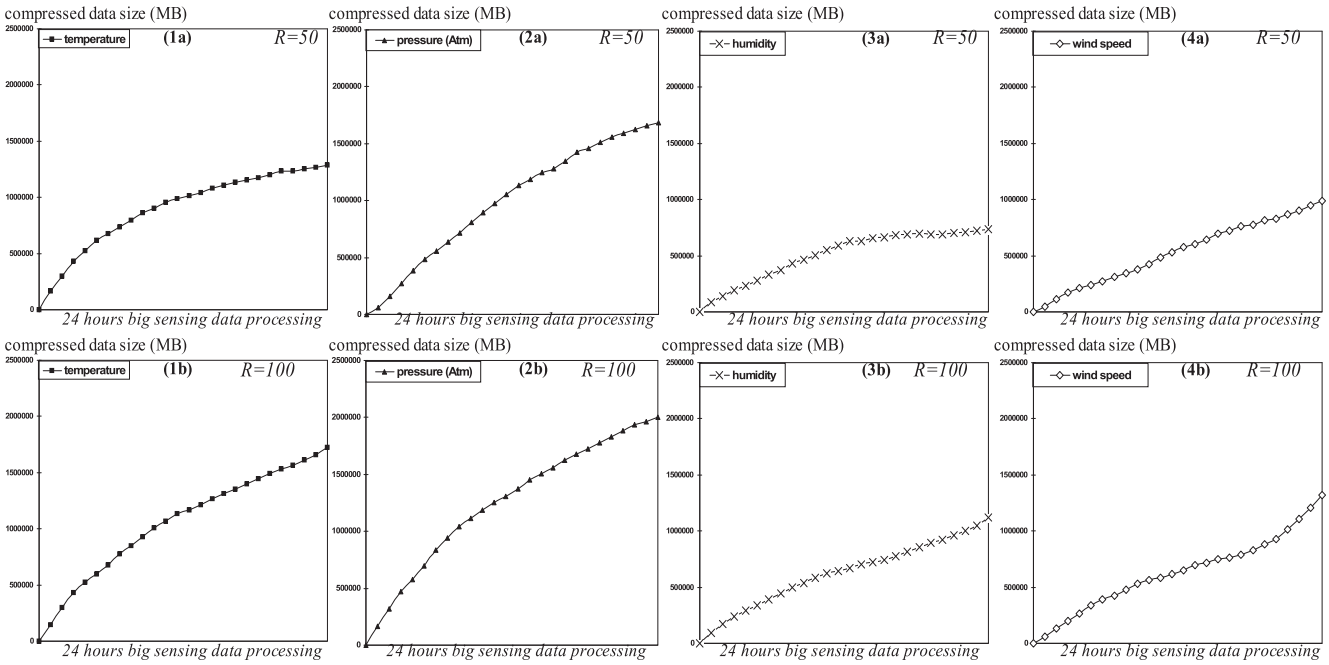



Fig. 8. Data size compressed within 24 hours test.

and performance gains to our compression algorithm. (2) It can be noticed that with th'e increase of testing data size, the compression ratio decreases whatever the value of R is. This result indicates that with more data gathered for testing, more heterogeneous data blocks could be found in meteorological big sensing data set. In other words, more new data blocks which are not compressible with the standard data chunks set could be detected.

### 5.3 Data Accuracy Analysis

In this section, we present data accuracy analysis. First the accuracy definition is briefly described according to the work [30] based on measuring the similarity between two vectors, one from real big data graph G and the other from G′ filtered data as the service provided by Cloud. There are two vectors at a certain time stamps. To describe the similarity between two nodes, correlation coefficient model is used as shown in [30]. Suppose X from G and Y from G′ are two vectors. With Correlation Coefficient method, we can calculate the similarity between them by formula (15)

$$\mathrm{sim}(\mathrm{X},\mathrm{Y}) = \mathrm{r}(\mathrm{X},\mathrm{Y}) = \frac{\mathrm{cov}(\mathrm{X},\mathrm{Y})}{\sqrt{\mathrm{cov}(\mathrm{X},\mathrm{X})\cdot\mathrm{cov}(\mathrm{y},\mathrm{y})}}. \tag{15}$$

From (15) we can find that this similarity resembles to the "cos" similarity computation. sim(X,Y) has a data range [−1, 1]. The calculation of "cov(vector1, vector2)" is as following formula (16) to (18)

$$\mathrm{cov}(\mathrm{X},\mathrm{Y}) = \frac{1}{\mathrm{n}-1}\sum_{\mathrm{i}=1}^{\mathrm{n}}(\mathrm{X_i}-\bar{\mathrm{X}})(\mathrm{Y_i}-\bar{\mathrm{Y}}), \tag{16}$$

$$\mathrm{cov}(\mathrm{X},\mathrm{X}) = \frac{1}{\mathrm{n}-1}\sum_{\mathrm{i}=1}^{\mathrm{n}}(\mathrm{X_i}-\bar{\mathrm{X}})^2, \tag{17}$$

$$\mathrm{cov}(\mathrm{Y},\mathrm{Y}) = \frac{1}{\mathrm{n}-1}\sum_{\mathrm{i}=1}^{\mathrm{n}}(\mathrm{X_i}-\bar{\mathrm{X}})(\mathrm{Y_i}-\bar{\mathrm{Y}}). \tag{18}$$

So, the similarity between two vectors can be calculated with following formula (19)

$$\mathrm{sim}(\mathrm{X},\mathrm{Y}) = \frac{\sum_{\mathrm{i}=1}^{\mathrm{n}}(\mathrm{X_i}-\bar{\mathrm{X}})(\mathrm{Y_i}-\bar{\mathrm{Y}})}{\sqrt{\sum_{\mathrm{i}=1}^{\mathrm{n}}(\mathrm{X_i}-\bar{\mathrm{X}})^2(\mathrm{Y_i}-\bar{\mathrm{Y}})^2}}. \tag{19}$$

As we only need to correlate accuracy and similarity, only [0, 1] data range is selected. The original data range [−1, 1] can be normalized to [0, 1] for representing the accuracy from 0 to 100 percent. As shown in formula (20), sim(X,Y)′ is calculated instead of formula (19). sim (X,Y)′ ∈ [0, 1]

$$\mathrm{sim}(\mathrm{X},\mathrm{Y})' = \frac{\sum_{\mathrm{i}=1}^{\mathrm{n}}(\mathrm{X_i}-\bar{\mathrm{X}})(\mathrm{Y_i}-\bar{\mathrm{Y}})}{\sqrt{\sum_{\mathrm{i}=1}^{\mathrm{n}}(\mathrm{X_i}-\bar{\mathrm{X}})^2(\mathrm{Y_i}-\bar{\mathrm{Y}})^2}}. \tag{20}$$

Hence the accuracy for an edge in G at a time stamp t can be assessed with formula (21)

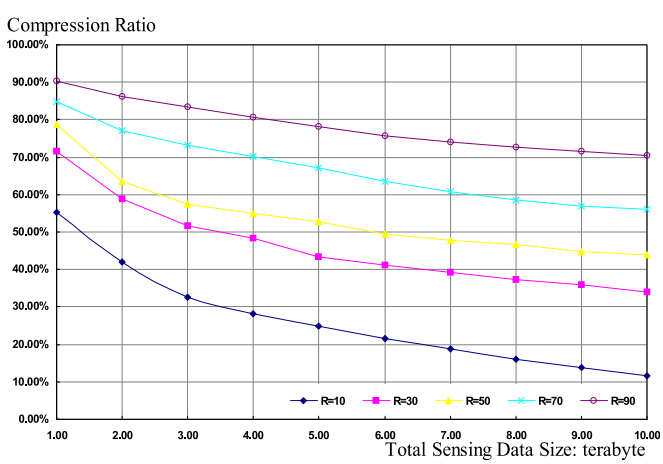



Fig. 9. Compression ratio for different 'r'.

$$\mathrm{sim}(\mathrm{X},\mathrm{Y})' = \frac{\sum_{\mathrm{i}=1}^{\mathrm{n}}(\mathrm{X_i}-\bar{\mathrm{X}})(\mathrm{Y_i}-\bar{\mathrm{Y}})}{\sqrt{\sum_{\mathrm{i}=1}^{\mathrm{n}}(\mathrm{X_i}-\bar{\mathrm{X}})^2(\mathrm{Y_i}-\bar{\mathrm{Y}})^2}}, \tag{21}$$

$$\mathrm{Accuracy} = \mathrm{sim}(\mathrm{X},\mathrm{Y})' \times 100\%, \tag{22}$$

$$\mathrm{Accuracy} = \frac{\sum_{\mathrm{i}=1}^{\mathrm{n}}(\mathrm{X_i}-\bar{\mathrm{X}})(\mathrm{Y_i}-\bar{\mathrm{Y}})}{\sqrt{\sum_{\mathrm{i}=1}^{\mathrm{n}}(\mathrm{X_i}-\bar{\mathrm{X}})^2(\mathrm{Y_i}-\bar{\mathrm{Y}})^2}}. \tag{23}$$

The final accuracy for "Accuracy" between two points within a cluster can be assessed by formula (22). In our example, T = 24 hours is used. The formulas (22) and (23) can be further transformed and calculated with formula (24)

$$\mathrm{Accuracy} = \sum_{\mathrm{t}=0}^{\mathrm{T}} \left( \frac{\left|\sum_{\mathrm{i}=1}^{\mathrm{n}}(\mathrm{X_{it}}-\bar{\mathrm{X}}_\mathrm{t})(\mathrm{Y_{it}}-\bar{\mathrm{Y}}_\mathrm{t})\right|}{\sqrt{\sum_{\mathrm{i}=1}^{\mathrm{n}}(\mathrm{X_{it}}-\bar{\mathrm{X}}_\mathrm{t})^2(\mathrm{Y_{it}}-\bar{\mathrm{Y}}_\mathrm{t})^2}} \right) \Big/ \mathrm{T} \times 100\%. \tag{24}$$

Suppose that in graph data set G(V, E), there are total S edges (with cluster-head structure, edge explosion is avoided) and each edge is indexec with s from [1, S]. We can calculate the Average Accuracy of the Cloud computed Data set 'G' against the original 'G'. This Average Accuracy is used in formula (25) to demonstrate our experiment results in Fig. 10

$$\mathrm{Average\_Accuracy} = \sum_{\mathrm{s}=1}^{\mathrm{S}} (\mathrm{Accuracy})_\mathrm{S}. \tag{25}$$

With the definition of above data accuracy, the data accuracy test is designed and conducted. The testing results are demonstrated in Fig. 10. Specifically, we use as the parameter R from 10 to 100 rounds for conducting accuracy test. As shown in Fig. 10, with the increase of compression ratio from 0 to 80 percent, the data accuracy decreases dramatically. However, it can be found in Fig. 10 that higher the R is, better the data accuracy can be achieved. The reason is that a larger R means more standard data chunks, hence a more refined similarity comparison to guarantee better data accuracy.

At the same time, with the increase of the standard chunks generation limitation round R, whatever the compression is, better data accuracy can be achieved. One important point should be mentioned. The experimental results in Fig. 10 show that for achieving 30 percent compression ratio, we can choose different combination of R and accuracy to realize this goal. In general, if we can set R $>$ 70 rounds and keep a compression ratio around 30 percent, the algorithm will always guarantee the data accuracy larger than 95 percent which will comes to the requirement of lots of real world applications. In other words, our compression algorithm can guarantee the acceptable data accuracy when make significant performance gains in data compression.

## 6 Conclusions and Future Work

In this paper, we proposed a novel scalable data compression based on similarity calculation among the partitioned data chunks with Cloud computing. A similarity model was developed to generate the standard data chunks for compressing big data sets. Instead of compression over basic data units, the compression was conducted over partitioned data chunks. The MapReduce programming model was adopted for the algorithms implementation to achieve some extra scalability on Cloud. With the real meteorological big sensing data experiments on our U-Cloud platform, it was demonstrated that our proposed scalable compression based on data chunk similarity significantly improved data compression performance gains with affordable data accuracy loss. The significant compression ratio brought dramatic space and time cost savings.

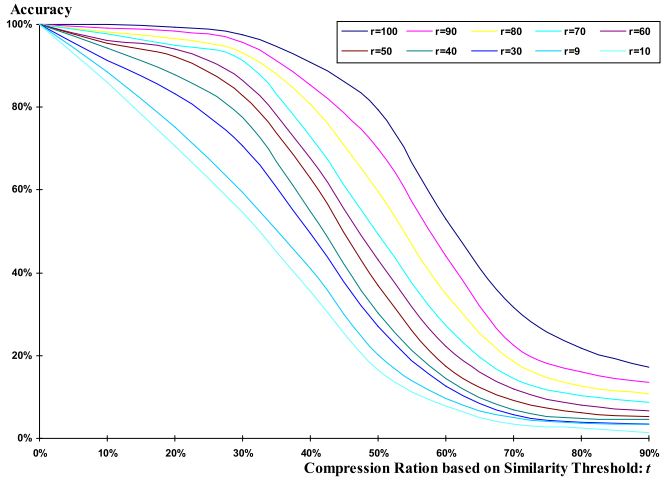



Fig. 10. Relationship between compression ratio and accuracy.

With the popularity of Spark and its specialty in processing streaming big data set, in future we will explore the way to implement our compression algorithm based on data chunks similarity with Spark for better data processing achievements.

## Acknowledgments

This paper is partially supported by Australian Research Council Linkage Project ARC LP140100816 and by STRATUS Project (Security Technologies Returning Accountability, Trust and User-Centric Services in the Cloud) (https://stratus.org.nz). Jinjun Chen is the corresponding author.

## References

[1] S. Tsuchiya, Y. Sakamoto, Y. Tsuchimoto, and V. Lee, "Big data processing in cloud environments," *FUJITSU Sci. Technol. J.*, vol. 48, no. 2, pp. 159–168, 2012.

[2] "Big data: Science in the petabyte era: Community cleverness Required" *Nature*, vol. 455, no. 7209, p. 1, 2008.

[3] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, 2010.

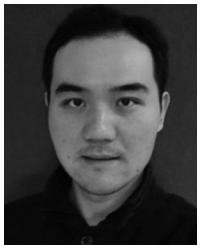
[4] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Gener. Comput. Syst.*, vol. 25, no. 6, pp. 599–616, 2009.

[5] L. Wang, J. Zhan, W. Shi, and Y. Liang, "In cloud, can scientific communities benefit from the economies of scale?," *IEEE Trans. Parallel Distrib. Syst.* vol. 23, no. 2, pp. 296–303, Feb. 2012.

[6] S. Sakr, A. Liu, D. Batista, and M. Alomari, "A survey of large scale data management approaches in cloud environments," *IEEE Commun. Surveys Tuts.*, vol. 13, no. 3, pp. 311–336, Jul.-Sep. 2011.

[7] B. Li, E. Mazur, Y. Diao, A. McGregor, and P. Shenoy, "A platform for scalable one-pass analytics using mapreduce," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2011, pp. 985–996.

[8] R. Kienzler, R. Bruggmann, A. Ranganathan, and N. Tatbul, "Stream as you go: The case for incremental data access and processing in the cloud," in *Proc. IEEE ICDE Int. Workshop Data Manage. Cloud*, 2012, pp. 159–166.
[9] C. Olston, G. Chiou, L. Chitnis, F. Liu, Y. Han, M. Larsson, A. Neumann, V. B. N. Rao, V. Sankarasubramanian, S. Seth, C. Tian, T. ZiCornell, and X. Wang, "Nova: Continuous pig/hadoop workflows," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2011, pp. 1081–1090.
[10] K. H. Lee, Y. J. Lee, H. Choi, Y. D. Chung and B. Moon, "Parallel data processing with mapreduce: A survey," *ACM SIGMOD Rec.*, vol. 40, no. 4, pp. 11–20, 2012.
[11] X. Zhang, C. Liu, S. Nepal, and J. Chen, "An efficient quasi-identifier index based approach for privacy preservation over incremental data sets on cloud," *J. Comput. Syst. Sci.*, vol. 79, no. 5, pp. 542–555, 2013.
[12] X. Zhang, C. Liu, S. Nepal, S. Pandey, and J. Chen, "A privacy leakage upper-bound constraint based approach for cost-effective privacy preserving of intermediate datasets in cloud," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 6, pp. 1192–1202, Jun. 2013.
[13] X. Zhang, T. Yang, C. Liu, and J. Chen, "A scalable two-phase top-down specialization approach for data anonymization using systems, in mapreduce on cloud," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 2, pp. 363–373, Feb. 2014.
[14] W. Dou, X. Zhang, J. Liu, and J. Chen, "HireSome-II: Towards privacy-aware cross-cloud service composition for big data applications," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 2, pp. 455–466, Feb. 2015.
[15] J. Conhen, "Graph twiddling in a mapreduce world," *IEEE Comput. Sci. Eng.*, vol. 11, no. 4, pp. 29–41, Jul./Aug. 2009.
[16] K. Shim, "Mapreduce algorithms for big data analysis," *Proc. VLDB Endowment.*, vol. 5, no. 12, pp. 2016–2017, 2012.
[17] N. Laptev, K. Zeng, and C. Zaniolo, "Very fast estimation for result and accuracy of big data analytics: The EARL system," in *Proc. 29th IEEE Int. Conf. Data Eng.*, 2013, pp. 1296–1299.
[18] X. L. Dong and D. Srivastava, "Big data integration," in *Proc. 29th IEEE Int. Conf. Data Eng.*, 2013, pp. 1245–1248.
[19] P. Mineiro, N. Polyzotis, and M. Weimer, "Machine learning on big data," in *Proc. 29th IEEE Int. Conf. Data Eng.*, 2013, pp. 1242–1244.
[20] A. Aboulnaga and S. Babu, "Workload management for big data analytics," in *Proc. 29th IEEE Int. Conf. Data Eng.*, 2013, p. 1249.
[21] M. Yuriyama and T. Kushida, "Sensor cloud infrastructure," in *Proc. 13th Int. Conf. Netw.-Based Inf. Syst.*, 2010, pp. 1–8.
[22] A. Alamri, W. S. Ansari, M. M. Hassan, M. S. Hossain, A. Alelaiwi, and M. A. Hossain, "A survey on sensor-cloud: Architecture, applications, and approaches," *Int. J. Distrib. Sens. Netw.*, vol. 2013, pp. 1–18, 2013.
[23] C. Ji, Y. Li, W. Qiu, U. Awada, and K. Li, "Big data processing in cloud environemtns," in *Proc.12th Int. Symp. Pervasive Syst., Algorithms Netw.*, 2012, pp. 17–23.
[24] L. Wang, G. Von Laszewski, A. Younge, X. He, M. Kunze, J. Tao, and C. Fu "Cloud computing: A perspective study," *New Gener. Comput.*, vol. 28, no. 2, pp. 137–146, 2010.
[25] X. Yang, L. Wang, and G. Laszewski, "Recent research advances in e-science," *Cluster Comput.*, vol. 12, no. 4, pp. 353–356, 2009.
[26] S. Sakr, A. Liu, D. Batista, and M. Alomari, "A survey of large scale data management approaches in cloud environments," *IEEE Commun. Surveys Tuts.*, vol. 13, no. 3, pp. 311–336, Jul.-Sep. 2011.
[27] S. Lattanzi, B. Moseley, S. Suri, and S. Vassilvitskii, "Filtering: A method for solving graph problems in mapreduce," in *Proc. 23th ACM Symp. Parallelism Algorithms Archit.*, 2011, pp. 85–94.
[28] K. Shim, "MapReduce Algorithms for Big Data Analysis," *Proc. VLDB Endowment*, vol. 5, no. 12, pp. 2016–2017, 2012.
[29] N. Sidiropoulos and A. Kyrillidis, "Multi-way compressed sensing for sparse low-rank tenors," *IEEE Signal Process. Lett.*, vol. 19, no. 11, pp. 757–760, Nov. 2012.
[30] C. Yang, X. Zhang, C. Liu, J. Pei, K. Ramamohanarao, and J. Chen, "A spatiotemporal compression based approach for efficient big data processing on cloud," *J. Comput. Syst. Sci.*, vol. 80, pp. 1563–1583, 2014.
[31] L. Ramaswamy, V. Lawson, and S. V. Gogineni, "Towards a quality-centric big data architecture for federated sensor services," *IEEE Int. Congr. Big Data*, 2013, pp. 86–93.
[32] A. Cuzzocrea, G. Fortino, and O. Rana, "Managing data and processes in cloud-enabled large-scale sensor networks: State-of-the-art and future research directions," in *Proc. 13th IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput.*, 2013, pp. 583–588.
[33] Y. Fang, L. Chen, J. Wu, and B. Huang, "GPU implementation of orthogonal matching pursuit for compressive sensing," in *Proc. 17th IEEE Int. Conf. Parallel Distrib. Syst.*, 2011, pp. 1044–1047.
[34] W. Wang, D. Lu, X. Zhou, B. Zhang, and J. Wu, "Statistical wavelet-based anomaly detection in big data with compressive sensing," *EURASIP J. Wireless Commun. Netw.*, 2013, Doi: 10.1186/1687-1499-2013-269.
[35] J. Wang, S. Tang, B. Yin, and X. Li, "Data gathering in wireless sensor networks through intelligent compressive sensing," in *Proc. IEEE INFOCOM*, Mar. 2012, pp. 603–611.
[36] S. H. Yoon and C. Shahabi, "An experimental study of the effectiveness of clustered aggregation (CAG) leveraging spatial and temporal correlations in wireless sensor networks," *ACM Trans. Sens. Netw.*, vol. 3, no. 1, Art. no. 3, 2007.
[37] R. Qiu and M. Wicks, "Cognitive networked sensing and big data," ISBN 978–1–4614–4544—9, DOI 10.1007/978–1–4614–4544–9.
[38] Real Time Big Data Processing with GridGain (2017, Feb. 16). [Online]. Available: http://www.gridgain.com/sitemap/
[39] Managing and Mining Billion-Node Garphs (2017, Feb. 16). [Online]. Available: http://kdd2012.sigkdd.org/sites/images/summerschool/Haixun-Wang.pdf
[40] Hadoop (2017, Feb. 16). [Online]. Available: http://hadoop.apache.org
[41] Sensor Cloud (2017, Feb. 16). [Online]. Available: http://www.sensorcloud.com/
[42] Big Data and Cloud Solutions in Amazon (2017, Feb. 16). [Online]. Available: http://aws.amazon.com/big-data/
[43] Big Data Beyond MapReduce: Google's Big Data Papers (2017, Feb. 16). [Online]. Available: http://architects.dzone.com/articles/big-data-beyond-mapreduce
[44] NASA NEX (2017, Feb. 16). [Online]. Available: http://aws.amazon.com/nasa/nex/
[45] Smart City with Internet of Things (Sensor Networks) and Big Data (2017, Feb. 16). [Online]. Available: http://www.academia.edu/5276488/Smart_City_with_Internet_of_Things_Sensor_networks_and_Big-Data
[46] Balancing Opportunity and Risk in Big Data, A Survey of Enterprise Priorities and Strategies for Harnessing Big Data (2017, Feb. 16). [Online]. Available: http://www.citia.co.uk/content/files/50_135–263.pdf



**Chi Yang** received the BS degree from Shandong University China, and the MS (by research) and PhD degrees in computer science from SUT and UTS, respectively, in Australia. He was a researcher in CSIRO, Australia. Currently, he is a postdoc research fellow for Unitec Stratus Project, New Zealand. His current major research interests include resilient systems, distributed system, large area intelligent sensing system, complex networks, big data processing, and cloud computing.

**Jinjun Chen** received the PhD degree in IT from Swinburne. He is a professor at the Faculty of Science, Engineering and Technology, Swinburne University of Technology, Australia. His research interests include big data, data science, data intensive systems, cloud, cybersecurity, and workflow management. He has published more than 130 papers in high-quality journals and conferences. He received IEEE Computer Society Outstanding Leadership Awards, Vice Chancellor Research Award, and many other awards.