Journal of Computer and System Sciences 80 (2014) 1563–1583

Contents lists available at ScienceDirect

Journal of Computer and System Sciences

www.elsevier.com/locate/jcss

A spatiotemporal compression based approach for efficient big data processing on Cloud

Chi Yang [a,*], Xuyun Zhang [a], Changmin Zhong [b], Chang Liu [a], Jian Pei [c], Kotagiri Ramamohanarao [d], Jinjun Chen [a]

[a] Faculty of Engineering and Information Technology, University of Technology Sydney, PO Box 123, Broadway, NSW 2007, Australia
[b] Joowing Australia Pty Ltd., 26 Entally Drive, Wheelers Hill, VIC 3150, Australia
[c] School of Computing Science, Simon Fraser University, 8888 University Drive, Burnaby, BC V5A 1S6, Canada
[d] Department of Computing and Information Systems, The University of Melbourne, VIC 3110, Australia

ARTICLE INFO

Article history:
Received 16 May 2013
Received in revised form 10 December 2013
Accepted 10 April 2014
Available online 24 April 2014

Keywords:
Big data
Graph data
Spatiotemporal compression
Cloud computing
Scheduling

ABSTRACT

It is well known that processing big graph data can be costly on Cloud. Processing big graph data introduces complex and multiple iterations that raise challenges such as parallel memory bottlenecks, deadlocks, and inefficiency. To tackle the challenges, we propose a novel technique for effectively processing big graph data on Cloud. Specifically, the big data will be compressed with its spatiotemporal features on Cloud. By exploring spatial data correlation, we partition a graph data set into clusters. In a cluster, the workload can be shared by the inference based on time series similarity. By exploiting temporal correlation, in each time series or a single graph edge, temporal data compression is conducted. A novel data driven scheduling is also developed for data processing optimisation. The experiment results demonstrate that the spatiotemporal compression and scheduling achieve significant performance gains in terms of data size and data fidelity loss.

© 2014 Elsevier Inc. All rights reserved.

1. Introduction

With the fast development of modern information technology, we enter a new era of data explosion which introduces the new problems for big data processing. The definition of big data is as follows. Big data [1–4] is a collection of data sets so large and complex that it becomes difficult to process with on-hand database management systems or traditional data processing applications. It represents the progress of the human cognitive processes, usually including 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 [2,3,10,11]. Big data has the typical characteristics of three 'V's, volume, variety and velocity. Big data sets come from many areas, including meteorology, connectomics, complex physics simulations [1,2], genomics [1], biological study, gene analysis and environmental research [2]. Big data sets also come from finance and business informatics in which Internet search is greatly influenced and limited. These data sets grow in high speed and size because they are increasingly being collected or generated by ubiquitous mobile sensing devices, aerial sensory systems, cameras, microphones, software logs, radio-frequency identification readers, business transactions and wireless sensor networks [1,2]. According to literature [3,4], since 1980s, the technological generated information data doubled its size in every 40 months all over the world. In

* Corresponding author.
E-mail addresses: chiyangit@gmail.com (C. Yang), xyzhanggz@gmail.com (X. Zhang), Changmin.zhong@joowing.com (C. Zhong), changliu.it@gmail.com (C. Liu), jpei@cs.sfu.ca (J. Pei), kotagiri@unimelb.edu.au (K. Ramamohanarao), jinjun.chen@gmail.com (J. Chen).

http://dx.doi.org/10.1016/j.jcss.2014.04.022
0022-0000/© 2014 Elsevier Inc. All rights reserved.

the year of 2012, there were 2.5 quintillion (2.5×10^{18}) bytes of data being generated every day. Hence, the technique to process big data has become a fundamental and critical challenge for modern society.

Cloud computing can be regarded as an ingenious combination of a series of developed or developing ideas and technologies, establishing a pay-as-you-go business model by offering IT services using economies of scale [5–10,15–18]. Cloud computing is the use of computing resources (hardware and software) that are delivered as a service over a network (typically the Internet). The name comes from the use of a Cloud-shaped symbol as an abstraction for the complex infrastructure it contains in system diagrams. Cloud computing entrusts remote services with a user's data, software and computation. It allows companies to avoid upfront infrastructure costs, and focus on projects that differentiate their businesses instead of infrastructure. Cloud computing relies on sharing of resources to achieve coherence and economies of scale similar to a utility (like the electricity grid) over a network. At the foundation of Cloud computing is the broader concept of converged infrastructure and shared services.

Cloud computing provides an ideal platform for big data storage, dissemination and interpreting with its massive computation power. In lots of real world applications, such as social networks, complex network monitoring, the scientific analysis of protein interactions and wireless sensor networks self monitoring, it is avoidable to encounter the problem of dealing with big graph data and big graph data streams. To monitor individual nodes, their related behaviours and detecting the abnormal operation, huge amount of streaming graph data must be processed and analysed. It is well known that the processing of big graph data can be costly and inefficient with current typical Cloud computing techniques or tools, such as Map-Reduce [27–30]. Those big graph data processing techniques introduce complex and multiple iterations. Those iterations and recursive algorithms may cause computation problems such as parallel memory bottlenecks, deadlocks on data accessing, algorithm inefficiency [27–30]. 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.

To curb and avoid the formation of the above problems, in this paper, we propose a novel technique for effectively processing big data, especially streaming big graph data on Cloud. Generally speaking, the big data will be compressed firstly with its spatiotemporal features on Cloud. By exploring spatial correlation of big data, we partition a graph data set into clusters so that, in one cluster all edges from the graph have similar time series of data. In each cluster, the workload can be shared by the inference based on time series similarity. By exploiting temporal correlation, in each time series or a single edge in a graph data set, temporal data compression is conducted according to time series trend and data item order. Based on the spatiotemporal compression, a data driven scheduling will be developed to allocate the computation and storage on Cloud for better big data processing services. The evaluation is conducted on three aspects for measuring the performance gains of this novel data processing approach. Firstly, the data size can be significantly reduced by the spatiotemporal compression on Cloud compared to the previous big data processing techniques. Secondly, our data driven scheduling based on the spatiotemporal compression can distribute big data processing workload optimally on Cloud to achieve significant time performance gains. Thirdly, the data processing quality and fidelity loss of our proposed approach comes to most of application requirement.

1.1. Organization of the paper

The research work in this paper is organized as follows. In Section 2, we review the related work and offer the problem analysis. In Section 3, a novel spatiotemporal compression technique on Cloud for big data will be introduced to reduce the size of big data sets. In Section 4, a novel scheduling algorithm is developed to distribute the big data, or graph data on the Cloud platform with better fairness for time performance gains. This scheduling will be based on the processing results from the spatiotemporal compression offered in Section 4. In Section 5, we offer the integrated approach and the related algorithms for the compression and the scheduling proposed in Sections 3 and 4. In Section 6, evaluation is designed and conducted in three aspects to verify the algorithms in terms of data exchange size, the time efficiency and fidelity loss (data processing quality). In Section 7, we conclude the work in this paper with the outlook of future work.

2. Related work & problem analysis

In this section, the current state of art related to this paper are reviewed including big data processing, Cloud computing, spatiotemporal data compression and clustering, and some specific tools for big data processing over Cloud such as MapReduce and Hadoop. With the review of the current typical work, the corresponding problem analysis will be offered.

2.1. Related work

Nowadays, big data is measured with terabyte, petabyte, exabyte or zettabyte which offer a direct feeling of big data size [3]. The huge size is one important feature, but not the only one feature for big data. It can be typically described as the characteristics of three 'V's, volume, variety and velocity. Specifically, big data sets have a large or even endless data volume which makes its computation and management exhaust incredible resources. In addition to the large volume, the sources of big data sets can be very diverse. It means that the format and structure of big data set can change. The data types are far more beyond traditional structured data under the theme of big data. The high velocity of big data makes it

time sensitive and stream-like. The main challenges of big data processing include capture, recover, storage, search, sharing, analysis, interpretation and visualization [1,2].

There are some real world big data examples as follows [1–4,31]. For business, finance institute and scientific research, there is important information buried under the ocean of big data. That information can bring significant financial value. For example, the health care financial cost in US is around $300 billion per year, which is 0.7% annual productivity growth of US. The effective mining and processing big data for US health care means million dollars financial saving. The big data also comes from US retail market. There is 60% retail increase per year, which counts for 0/5%–1% US annual productivity. To analyse and interpret that big data, it can help business to improve its market profits. In the US manufacturing industry, with the adoption of some proper big data processing techniques, the cost for product development decreases up to 50% and the working capital reduces up to 7%. The cost for Europe public sector administration is around €250 billion per year. It occupies 0.5% annual productivity growth of the Europe Union. To understand the big data from climate observation in real time is critical for scientific research, agriculture, public transportation, weather insurance company, aviation and military activities. It can protect profits of everybody from bad weather.

In order to process big data sets, different techniques, approaches and platforms have designed and developed. Whatever the technology is based on, hardware or software, they mainly focus on three topics, increasing data storage, increasing data processing and data availability. Data storage has grown significantly with shifting from analog to digital devices in past 20 years. Similarly to storage devices, the computation capability of modern processing devices can reach more than 6.5×10^{12} million instructions per second. The computation capacity has experienced a sharp increase. Current techniques for big data processing and analysis include classification, parallel computing, distributed computing, cluster analysis, crowd sourcing, regression, spatial analysis, learning, temporal analysis, neural networks, network analysis, optimisation, prediction, pattern recognition, data fusion & integration and visualization, etc.

Those current developments offer a solid support for processing big data in a distributed and paralleled model. Current technologies such as grid and Cloud computing all aim to access huge computing power by aggregating multiple resources and offering an integral system view [8,9]. Among these technologies, Cloud computing proves to be a powerful architecture conduct large-scale, distributed and paralleled processing for complex computing. Cloud computing [5,6] has offered an approach to abstract and use computing infrastructure. An important aim of Cloud computing is to deliver computing as a solution for processing big data, such as large-scale, multi-media and high dimensional data sets. Both big data and Cloud computing are the fastest-moving technologies identified in Gartner Inc.'s 2012 Hype Cycle for Emerging Technologies [5,7]. Therefore, Cloud computing is a fine answer when talking about big data processing method for different resources.

At present, some work has been done for processing big data with Cloud. For example, Amazon EC2 infrastructure as service is a typical Cloud based distributed system for big data processing. Amazon S3 supports distributed storage. MapReduce [12,13,28,29] is adopted as a programming model for big data processing over Cloud computing. Plenty of recent research has investigated the issues of processing incremental data on Cloud. Kienzler et al. [24] designed a "stream-as-you-go" approach to access and process on incremental data for data-intensive Cloud applications via a steam-based data management architecture. Bhatotia et al. [25] extended the traditional Hadoop framework [14] to a novel one named as Incoop by incorporating several techniques like task partition and memorization-aware schedule. Olston et al. [26] presented a continuous workflow system called Nova on top of Pig/Hadoop through stateful incremental data processing.

Recently, MapReduce has been widely revised from a batch processing framework into a more incremental one to analyse huge-volume incremental data on Cloud [13]. It is a framework for processing parallelisable problems across big datasets using a large number of computers (nodes), collectively referred to as a cluster in which all computers (nodes) are on the same local network and use similar hardware; or a grid in which the nodes are shared across geographically and administratively distributed systems. Computational processing can occur on data sets stored either in a file system with unstructured or semi-structured data, or in a database with structured data. MapReduce can take advantage of locality of data, processing data on or near the storage assets to decrease transmission of data. There two main steps to use MapReduce function. At "Map" step, 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. At "Reduce" step, the master node then gathers the answers to all the sub-problems and combines them in some way to generate the final output. MapReduce can be applied to significantly larger datasets than "commodity" servers can handle. It can sort a petabyte of data in only a few hours. The parallelism also offers some possibility of recovering from partial failure of servers or storage during the operation. If one mapper or reducer function fails, the work can be rescheduled if the original input data is still available.

However, MapReduce also has its disadvantage in terms big data processing over Cloud computing. Under MapReduce programming model for Cloud, target problem needs to be parallelisable and split into a set of smaller code. Smaller pieces of code are executed in parallel over Cloud computing platform. This working style of MapReduce introduces lots of iterations when encountering big graph data. The real world big data graph examples include bipartite of appearing phrases in documents, social networks, airline graph, gene expression data, protein interactions and sensor networks monitoring. The modern graph data may take the form of data stream where newly created data is added into big data graph continuously. To process that big data sets or stream in big data graph, MapReduce may not be suitable because many iterations are brought for parallel graph processing [27,30,32]. Firstly, online query support is insufficient. Secondly, graph data cannot be intuitively expressed by MapReduce. Thirdly, each intermediate iteration result of graph data needs to be materialized. The whole graph structure needs to be sent on networks iteration after iteration [32] which incurs huge data movement.

That huge amount of intermediate results and data movement may cause time inefficiency, deadlock, errors and bad performance. So, how to design efficient parallel algorithms for processing big graph data, or big streaming graph data becomes an interesting research topic.

Spatiotemporal data suppression and its related data clustering, compression techniques can be widely found in network data processing area [19–22]. They are useful in terms of data exchange reduction over Cloud platform. Specifically, the work of CAG [20] presented an updated CAG algorithm that forms clusters of nodes with similar values within a given threshold (spatial correlations). The formed clusters remain unchanged as long as the sensor values stay within the given threshold (temporal correlations) over time. With the clustering technique, every time, there is only one sensor data being transmitted; whereas the data aggregation algorithm without clustering requires all nodes to transmit. Low Energy Adaptive Clustering Hierarchy ("LEACH") [22] is a TDMA-based MAC protocol which is integrated with clustering and a simple routing protocol in networks systems such as WSN. The goal of LEACH is to provide data aggregation for sensor networks while providing energy efficient communication that does not predictably deplete some nodes more than others. LEACH is a hierarchical protocol in which most nodes transmit to cluster-heads, and the cluster-heads aggregate and compress the data and forward it to the base station. Each node uses a stochastic algorithm at each round to determine whether it will become a cluster-head in this round. Nodes that have been selected as cluster-heads cannot become cluster-heads again for next k rounds in LEACH, where k is the desired percentage of cluster heads. Hence, each node has a $1/k$ probability of becoming a cluster-head in each round. At the end of each round, each node that is not a cluster head selects the closest cluster-head to join a cluster. Finally, all nodes that are not cluster-heads only communicate with the cluster head according to the schedule created by the cluster-head. LEACH offers an approach to automatically select suitable cluster-heads in terms of energy situation and radio coverage. It solves the technical issues such as topology construction and routing table maintenance. Edara [21] proposed a temporal trend based time series prediction model which can predict the future data with a small cost in real time. With respect to data compression, the temporal and order compression [19] techniques were used to reduce the data exchanges within a network topology. It can be further calibrated to satisfy the big data set reduction over Cloud platform. In work [23] proposed by Ail, the spatiotemporal based clustering and order compression were chosen adaptively for data compression over sensor networks. However, the data exchanging model is quite similar to that over a data exchanging network analysed with Cloud computing. In other words, the compression idea in WSN can be used on Cloud to reduce the size of big data set and graph data.

2.2. Problem analysis: big data processing on Cloud

As introduced in 2.1, nowadays, lots research work has been done towards the big data processing on Cloud. Cloud itself as a powerful computation and storage platform, also proves to be a suitable answer for processing and analysing big data. However, under the theme of big graph data processing, current typical techniques such as MapReduce may introduce high computation cost [27–30,32]. More work is still expected to improve the effectiveness and efficiency in terms of big graph data processing on Cloud. Therefore, compared to the too much iteration introduced by MapReduce, we aim to offer an optimal solution for streaming big graph data processing for applications with high real time requirement, we propose a novel approach to process streaming big data of cluster-head architecture over Cloud platform. This approach consists of two important technique parts. The first part focuses on reducing the data size over Cloud computing platform with spatiotemporal compression. Specifically, a clustering algorithm is developed based on spatial similarity calculation between multiple time series or streams of data. It compares the data streams according to the topology of the streaming data graph topologies from the real world. Generally speaking, under clustered network architecture, there are two classes of data interaction between data nodes from a complex data exchange network. One is the data interaction flows between the sibling nodes. The other is the interaction data flows between the leaf nodes and the cluster-nodes. In terms of the data exchange between leaf nodes and a cluster-head, the similarity model is relatively simple because it only involves the computation of the multiple attributions inside the data. All the similar data streams between a cluster-head and its leaf data nodes can be approximately inferred with a certain mathematic function based on only one or several bilateral data streams. In terms of the data exchange between sibling nodes, a data vector should include different origin and destination. The correspondent computation of similarity should also be modified with the consideration of different data origin and destination. Furthermore, because the data items in streaming big data sets are heterogeneous and carry very rich order information themselves, we develop an order compression algorithm to further reduce the size of big data sets.

The second part of our proposed big data processing approach focuses on the computation resource allocation of the Cloud. For the graph data, especially a big graph data set, the data can be heterogeneous and unfairly distributed over the Cloud platform. However to mapping this data to the Cloud platform for data processing, we need to consider the heterogeneous feature of the data. For example the data size from two nodes may be significantly different. To allocate the computing resource of Cloud according to node is not reasonable. Therefore we aim to offer a scheduling based on the analysis of data size from each node in a data exchange network to achieve shorter big data processing time and higher computing resource utility.

As shown in Fig. 1, the data streams generated from a complex social network pour into Cloud processing platform. To monitor individual nodes, their related behaviours and detecting the abnormal operation, huge amount of streaming graph data must be processed by Cloud. According to different application purposes, the processing over this streaming big graph data can be categorized to two types.

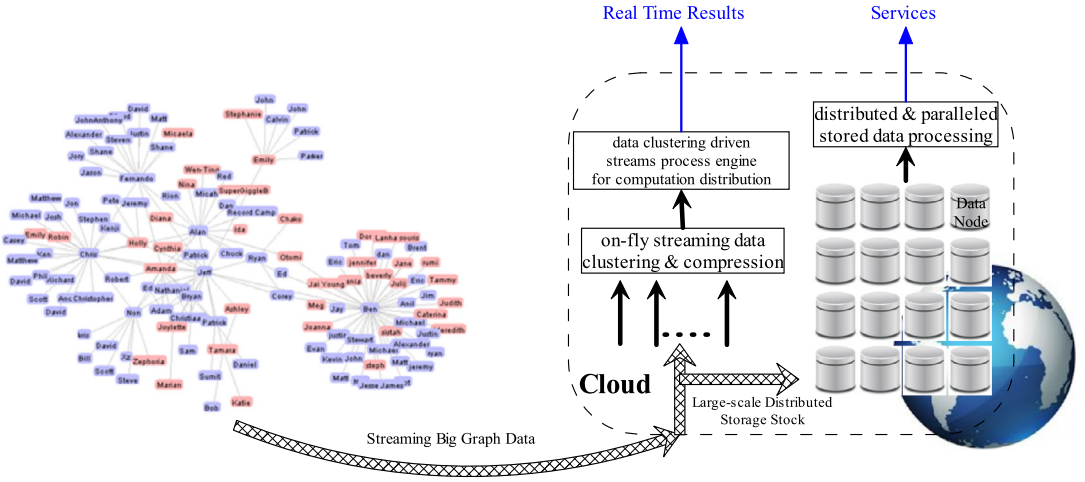

Fig. 1. Requirement for processing on-fly streaming data graph in Cloud computing.

The first one type focuses 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" [14] and its related "Hive", "HBase", "Zookeeper", and so on. The data processing results are offered as services to Cloud end users. The other type of applications cares more about the real time results and instance monitoring feedbacks over on-fly streaming graph data which may be unnecessary to be stored after high velocity data filtering. Under theme of this type of data processing, there is no need for permanent data storage nodes to intake high volume of data. The graph-based streaming data will be directly processed with some efficient filtering and event triggering algorithms. The correspondent approach for allocating the resources of Cloud computing is also needed to be developed to achieve costless computation and quick reaction time for event monitoring. Our approach proposed in this paper will mainly work in this application type to achieve both low computation cost and shorter, quicker processing reaction time.

3. Spatiotemporal compression

In Section 2, some typical big data processing techniques and tools over Cloud has been reviewed and analysed. In this section, we will introduce a novel technique based on spatiotemporal data correlations to compress big data on Cloud.

3.1. Spatiotemporal clustering for time series

In this section, we will introduce a clustering technique based on spatiotemporal data correlations. It computes the similarity of time series with regression. The definition, the similarity computation model and analysis of this similarity computation will be offered. It will be used for clustering the nodes in cluster-head network architecture. It aims to reduce the data exchange within each cluster by data guessing and inference.

3.1.1. Motivation

As shown by the motivating example in Fig. 2, there are several time series collected by the real world noisy sensor nodes deployed in a public transportation area. The sound is measured in standard Decibel (dB).

Specifically, data time series 1 and 2 are quite similar to each other according to the whole sampling period from sample 1 to 90. In other words, time series 1 or 2 can be used to approximately represent each other, which is used by most of current clustering algorithms to reduce the data size. For example, to report the three time series 1, 2 and 3, previous clustering algorithms can divide three time series into 2 time series groups A and B. In group A, there are time series 1 and 2 from sensor 1 and sensor 2. In group B, there is time series of sensor 3. However, there is one problem in the previous clustering algorithms. The timing of re-clustering for time series is not optimal. Because the re-clustering frequency has huge influence for the energy cost, for a given data series set, we want to do the minimum re-clustering to capture the maximum time series difference. In Fig. 2, it can be observed that, from sample 65, the data feature of time series 1, 2 and 3 changes. The time series before sample 65 is very bumpy. It means that, the difference between time series 1, 2 and 3 can change quickly. To capture that changing, more clustering operation is required. But after sample 65, time series 1, 2 and 3 become relatively smooth and static. It means that with less re-clustering operation, the partition of time series groups can be done with the satisfaction to accuracy requirement. In order to capture these attributes of three time series and carry out effective clustering, different method can be adopted. For example, if the data window with of each node can be maintained by a cluster-head, we can use normal regression to predict the data trend of each time series. Hence, that data trend can be used as a criterion for sensor nodes partition in each cluster. Furthermore, how to choose an optimal time frequency

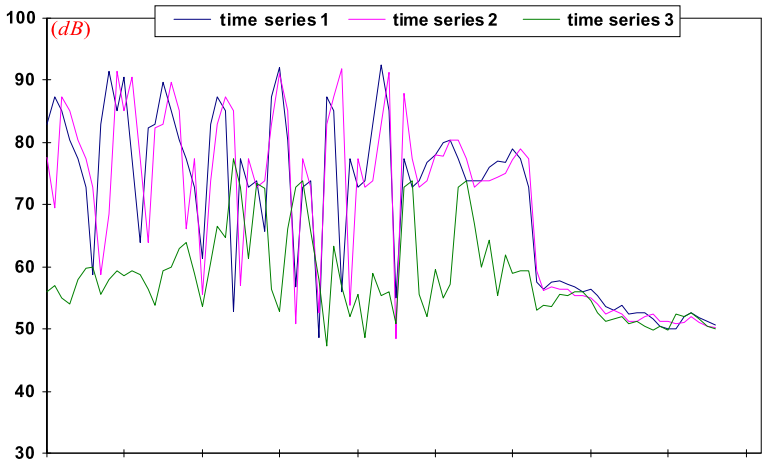

Fig. 2. Motivating example clustering based on time series difference.

of re-clustering according to the feed back of analysis of data history raises an interesting research topic. In this paper, we develop a clustering algorithm with improved data aggregation which in stead of carrying out regression over a set of historical data directly, it carries out data aggregation over a consecutively series of weighted data changes. For example, in Fig. 2, to predict the changing trend after sample 65, the prediction regression model will not directly work on the data samples before time stamp 65. As an alternative, a series of data trends with time stamp from 1 to 64 will be calculated for regression. In addition, with our proposed method, the problem of how to choose an optimal timing for re-clustering will also be discussed. We will give an approach to choose a suitable data distribution to deploy our proposed clustering. Hence, over the unsuitable data distributions, the clustering can be terminated to avoid the cost of frequent clustering operations.

3.1.2. Similarity and error model influence to clustering

Our clustering algorithm based on weighted data changes conducts its decision making for node sets partitioning according to whether two time series being similar enough. So, firstly, it is important to define the similarities of two time series. Popular and direct similarity definition can be based on some average distance of sets of previous data belonging to different time series. To calculate that distance for measuring similarity and to predicate the future similarity of two time series, temporal prediction models should be developed. Current work can be found for temporal prediction based data regression. However, there is a main disadvantage for this prediction model if applying it for clustering. When two time series have shape similar to "cos()" and "sin()" functions, though the regression results of two time series can be of high level similarity, the true situation of two time series can be totally different.

To overcome the above inaccuracy brought by normal regression model based on historical temporal data, we develop a novel regression based data prediction model. Suppose that there are two time series, denoted as $X_1\{x_{11}, x_{12}, \ldots, x_{1m}\}$, and $X_2\{x_{21}, x_{22}, \ldots, x_{2m}\}$. m is the time tamp for each data collection rounds. We aim to predict the average dissimilarity of data trend for X_1 and X_2 in future m rounds. Based on X_1 and X_2, we can calculate a dissimilarity vector $D(d_1, d_2, \ldots, d_m)$, where $d_i = x_i - y_i$. With the data trends vector D, we develop a weighted data regression model to calculate the average dissimilarity of data changes between time series X_1 and time series X_2. The specific regression model is as follows.

X is a data value set with temporal and spatial formed by n sensor vectors. In the matrix (1), the values collected buy sensor nodes 1 to n during m data collection rounds is described. Then we assign a weight vector for the changing slopes of each X_i, denoted as $W\{w_1, w_2, \ldots, w_{m-1}\}$ which means that according to a specific time stamp, the weight for carrying out regression is different. Then the matrix (1) can be changed to a data changing matrix, X' in (2). With $W \times X'$, we can calculate a weighted data changing value matrix $V = W \times X'$ in (3). The weights vector is generated with the assumption that the importance is of variation of Gaussian distribution with the changing of different stamps. In other words, more recent data has more influence for prediction.

$$X = \begin{bmatrix} x_{11} & x_{12} & \ldots & \ldots & x_{1m} \\ x_{21} & x_{22} & \ldots & \ldots & x_{2m} \\ x_{31} & x_{32} & \ldots & \ldots & x_{3m} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{n1} & x_{n2} & \ldots & \ldots & x_{nm} \end{bmatrix} \tag{1}$$

$$X' = \begin{bmatrix} (x_{12}-x_{11})^2 & (x_{13}-x_{12})^2 & \dots & \dots & (x_{1m}-x_{1(m-1)})^2 \\ (x_{22}-x_{21})^2 & (x_{23}-x_{22})^2 & \dots & \dots & (x_{2m}-x_{2(m-1)})^2 \\ (x_{32}-x_{31})^2 & (x_{33}-x_{32})^2 & \dots & \dots & (x_{3m}-x_{3(m-1)})^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ (x_{n2}-x_{n1})^2 & (x_{n3}-x_{n2})^2 & \dots & \dots & (x_{nm}-x_{n(m-1)})^2 \end{bmatrix} \tag{2}$$

$$V = \begin{bmatrix} X'_1 \\ \dots \\ \dots \\ X'_n \end{bmatrix} \times \begin{bmatrix} w_1 \\ \dots \\ \dots \\ w_n \end{bmatrix} \tag{3}$$

In order to calculate the W for a time flow, the probability density function of normal distribution should be configured to describe a time series in which a data with newer time stamp should have higher importance. According to $3\times\sigma$ principle, in Gaussian distribution, we only use the domain between $[0, 3\times\sigma]$ to calculate the final weight vector W, and it is enough to guarantee that $\sum_{k=1}^{m-1} w_k \approx 1$ because more according more than 97% data will be within this $[-(3\times\sigma), 3\times\sigma]$. However, the time stamp for sensing data changes from $[0, +\infty]$ which is not between the domain of $[-(3\times\sigma), 3\times\sigma]$. So in our work, we change the standard normal distribution as follows in (4).

$$\sum_{k=1}^{m-1} w_k = \int_0^{3\sigma} \frac{2}{\sigma\sqrt{2\pi}} e^{(-\frac{(x-\mu)^2}{2\sigma^2})} \tag{4}$$

Because we assume that the starting time for data collection is 0, we can get $\mu = 0$. So the summary value of the whole weights domain is calculated with (5)

$$\sum_{k=1}^{m-1} w_k = \int_0^{3\sigma} \frac{2}{\sigma\sqrt{2\pi}} e^{(-\frac{x^2}{2\sigma^2})} \tag{5}$$

As a result, for a data history window with m items or a vector X_i, we can distribute $m-1$ consecutive to the domain calculated with formula (5). Specifically, the data range between $[0, 3\times\sigma]$ is divided into $m-1$ fragments as 0 to $3\times\sigma/(m-1)$, $3\times\sigma/(m-1)$ to $6\times\sigma/(m-1)$ until to the final fragment, $[(m-2)\times 3\times\sigma/(m-1), 6\times\sigma/(m-1)]$. Then any item w_k in the weight vector W can be calculated following (6). With the above W, the regression matrix V can be evaluated.

$$w_k = \int_{\frac{(k-1)\times 3\sigma}{m-1}}^{\frac{k\times 3\sigma}{m-1}} \frac{2}{\sigma\sqrt{2\pi}} e^{(-\frac{x^2}{2\sigma^2})} \tag{6}$$

To compare the similarity of any two time series X_i and X_j based on our data changing regression model, we only need to compute the summary difference of corresponding V_i and V_j. If $|V_i - V_j| <$ Threshold (threshold is a given error bound from application requirement), time series X_i and X_j will be judged as similar enough and allocated into the same cluster.

3.1.3. Clustering by exploiting data changing

In Section 3.1.2, we introduced our approach for computing the similarity between any two time series X_i an X_j. With that similarity, the clustering process can be carried out. In this section, we will specify what kind of data distribution is suitable our clustering algorithm. As shown in Fig. 3(a), under a certain data distribution, if the similarity of time series is of high level, with the time stamp changing from T_1 to T_2, the clustering algorithm does reduce the number of reporting node. Specifically, in Fig. 3(a), at T_1, only 3 nodes are selected as cluster-heads and at T_2, only 2 nodes are selected as cluster-heads. However, under some high variation data distribution, the effectiveness of clustering algorithm can be not good due to the distribution of big data sets. As shown in Fig. 3(b), at time stamp T_1, the data variation is relative steady. However, when it comes to time stamp T_2, sensing data set is of high volatile and it makes the clustering algorithm divide more cluster-head.

Under some extreme conditions, it can make all the leaf nodes become cluster-heads as shown in Fig. 3(b) when time stamp is equal to T_1. So under this scenario, we do not want to use the clustering algorithm, because the clustering brings no transmission reduction for data exchanging. Furthermore, it introduces further time and resources for the data exchange of computing and clustering from cluster-heads to leaf nodes. Hence, we develop a method which exploits the trade-off between cost and gains brought by clustering algorithm to adaptively deploy our clustering algorithm according to underlining data changes. This method is based on the computation resource consumption model which decomposes and analyses the cost of different activities and operation over Cloud platform for simulating a big graph data set.

- M_s: the size of the new clustering message to be propagated

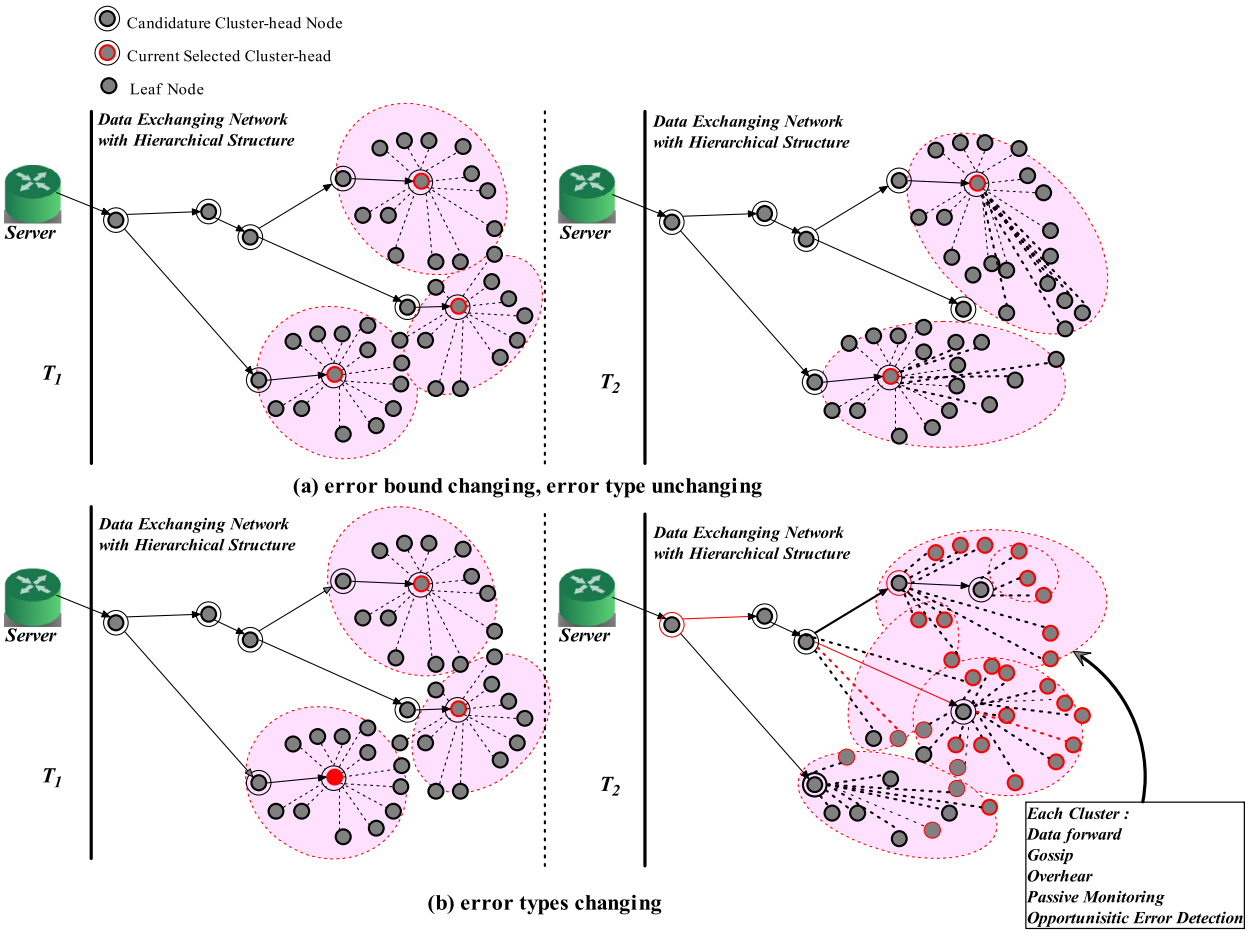

Fig. 3. Influence of data changes to a clustering algorithm with static error model.

- C_b: the time and computation cost to broadcast 1 byte of data
- C_r: the time and computation cost to receive 1 byte of data
- P_s: the size of data block
- $h(G, nodeID)$: the jumps from gateway to leaf nodes
- C_{instr}: computation and time cost for executing instructions for clustering
- D: duration
- t: time stamp

In every data stamp for data exchange within a certain network, suppose that there are n different nodes. Suppose that totally the clustering algorithm selects x cluster-heads and $n - x$ leaf nodes. Then for k rounds of data exchange, the data receiving cost should be $P_s \times k \times t \times (x) \times C_r$. The data sending cost should be $P_s \times k \times t \times (x) \times C_b$. However, the time and computation cost for propagating the new clustering should be counted. The cost of data exchanging is $M_s \times k \times t \times (x) \times C_r$ and $M_s \times k \times t \times (x) \times C_b$ respectively. There also is some cost for clustering algorithm at the final gateway, C_{instr}. If $k \approx n$, it can be observed that no cost will be saved and it also brings new clustering energy cost. When the number of x exceeds a certain level, we will not use the proposed clustering algorithm to realize saving because it has less effects under this scenario. In stead, we use an order compression approach to reduce data size to achieve the size reduction for big data on Cloud.

As described above, under some situation, the proposed clustering algorithm can be ineffective. To offer a compensation method which can still bring data exchange saving when the clustering algorithm cannot achieve performance gains, an order data compression approach is developed. The concept of order compression comes from the traditional data compression area. It means to use the order information carried among a group of data items to compress the data. The order compression can happen in a complex data unit or a group of data units in a data exchanging network. It is a kind of spatiotemporal compression exploiting data correlations in for data compression.

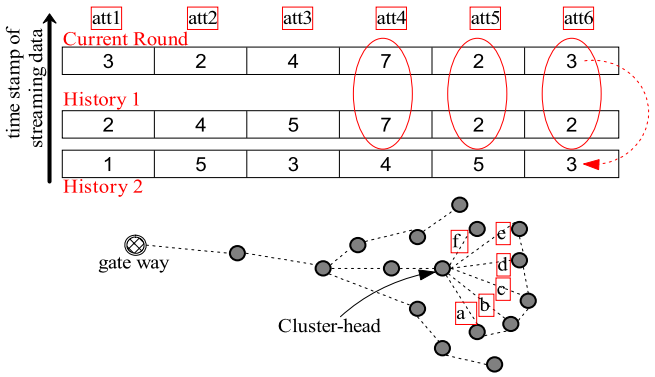

Fig. 4. Order compression for multiple attributes sensing data.

3.2. Order compression with spatiotemporal correlations

3.2.1. Order compression for multi-attributes data

To compress multiple attributes sensing data with order data information, some techniques have been developed in work [19]. As shown in Fig. 4, there is a data sampling and exchanging network. According to the previous techniques [21], when there is a data record collected by a node in this network, to send back this data, data exchange will start. Based on current techniques, there are only two possible choices for the newly collected data. (1) Transmit data to the parent node, (2) suppress the data transmission according to a predefined error range c. The previous technique can easily recognize whether there is a suppression and transmission when there is less data change happening. As shown in Fig. 4, the current collected data record has no change in its 4th and 5th data attributes. So, the transmission of the 4th and 5th attributes will be cancelled according to the in-network data suppression approach [21]. But, if there is also no change in the 6th attribute when comparing the data history table of the 6th attribute in Fig. 4, the transmission of the 6th data cannot be suppressed. In other words, single temporal prediction model loses the opportunity for further data exchanging compression. Both data cluster-head and leaf node in a network maintain the same data history table which can offers the message of previous message for order compression. In the example of Fig. 4, order message of previous attributes can be utilized to represent a location of the repetitive data in the data history table in of the 6th attribute, there exists the opportunity to cancel the transmission of the 6th attribute to reduce the size of the current data record. More specifically, the reporting order of the first attributes can have $3! = 6$ different orders. The unreported attributes 4th and 5th have to obey the original order because the cluster-head needs this order for predicting the unreported attributes. So, totally, 6 types of reported attribute order can represent 6 different locations in the data history table of attribute 6th. The second location of the data history table of the attribute 6th is equal to the current collected data. Then the order is used to transmit the previous three attributes back to the cluster-head. And the cluster-head can infer the value 3 for attribute 6th based on the received attributes order and its maintained data history table of the 6th attribute. By the above method, data exchanging for the 6th attribute is saved compared to the previous single temporal data reduction technique [21].

3.2.2. Order compression based on encoding data changes of nodes

In Section 3.2.1, we mainly discuss how to use order information of multiple attributes to compression the exchanging data size for a big data graph. In this section, the order compression will be used to compression data transmission in tiered tree topology. As shown in Fig. 5, there is a data exchanging network for data sampling and communication. Firstly, the tree topology can be divided into a layer topology. Our research is carried out on a cluster, such as x_0 and all its child nodes, x_1 to x_n. Then our proposed compression can be generalized to the whole data collection network. In the cluster x_0, all the nodes from x_0 to x_n form a one hop network. If x_0 maintains a data buffer for all its child nodes with their length of $L = 2$, at each data gathering round, x_0 can have $l \times n$ previous data for x_1 to x_n.

Because the length of the data buffer is $L = 2$; the buffer for a certain child node can be indexed with two real data reports. For example, the buffer elements for x_5 can be represented with the combination order of x_1 and x_2; and the buffer elements for x_6 can be represented with the combination order of x_3 and x_4. In addition, the x_7 can be represented with the combination order of $(x_1; x_2)$ and $(x_3; x_4)$. With this encoding method, it can be calculated that for a given buffer length L, and n child nodes cluster, the nodes required to represent a buffer with its length L should be $m! \geqslant L$. In other words, we can calculate how many nodes are required, m to represent another node. Then we can determine how many in the node sets with x_1 to x_n should report and how many can be suppressed with the method offered by the in-network prediction model. For the unreported data, it will be approximated with the buffered data and its index is given by the order of the reported data. Theoretically, more nodes are added, more data compression can be achieved with this method without considering the data mapping and index cost. Furthermore, this compression approach can be recursively deployed over the whole data gathering tree topology. For example, in Fig. 5, node h can use the order of cluster x_0, cluster k and

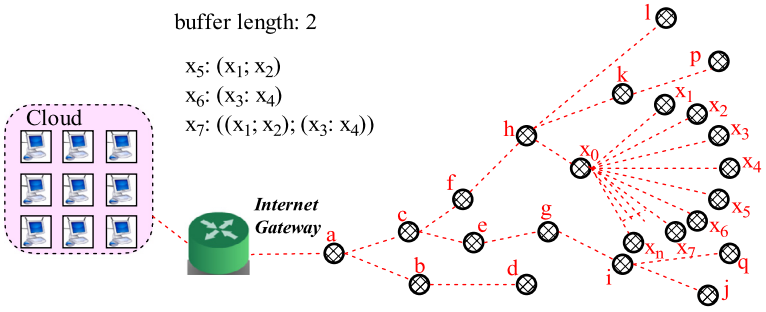

Fig. 5. Order inter-node order compression for data exchange in cluster-head network topology.

node l to carry out the data compression operation over it. The advantage of this order compression based on buffering historical data is that its compression effectiveness is steady under some real data sets.

For example, for independent temporal based data compression and our proposed clustering algorithm, if data sets change dramatically, the cost of re-clustering and independent temporal suppression can be extremely high. However, according to our knowledge, lots of dramatic changes are periodic, which means that most of current data have appeared previously. So if the previous repetitive data can be retrieved and reused, we can greatly decrease the time and resource cost for Cloud computing. In other words, the buffering based order compression can reduce more data exchange under the data set where tradition temporal suppression and the proposed clustering algorithm lose their effects.

4. Data driven scheduling on Cloud

In the previous section, we introduced the approaches for reducing the big data size with spatiotemporal data suppression. However, a smaller size of data does not definitely mean a shorter processing time which is quite related to the task division and workload distribution over Cloud. In order to offer a shorter and quicker processing time, we will introduce a novel scheduling algorithm based on the spatiotemporal compressed data sets.

4.1. Different mapping strategies for scheduling over Cloud

To show the necessity and effectiveness of our data driven scheduling and its mapping, two types of mapping strategies, node based mapping and edge based mapping will be introduced first for comparison.

4.1.1. Mapping for Cloud scheduling with real network nodes

The most direct and easy way to distribute and schedule the big data processing task over Cloud is based on the real work topology of the network itself. Under the theme of this mapping, the mapping algorithm is pretty simple and the computation resources are divided and distributed to each node for simulating and analysing data flows in a real world network. As shown in Fig. 6(a), to filter the graph data stream from a cluster of network nodes, one to one mapping is conducted. For example all the data flowing through node a, and all the operations from node a will be allocated with a computation unit in the Cloud. This mapping and resource allocation is fair according to the real world network topology, but it could be extremely unfair in terms of the distribution of data size under a heterogeneous data exchange network. For example, in Fig. 6(a) compared to node a, node b, c and d may experience huge data flows within certain time duration which will cost much longer time for data analysis. After the computation unit allocated to node a has finalized its processing, it has to wait for the results from node b, c and d to form high level services to end users. In other words, the unfairly distributed workload can significantly delay the reaction and processing time for processing big data sets over Cloud.

4.1.2. Mapping for Could scheduling with data exchange edges

Instead of directly allocating the computation resources over Cloud according to the real world network topology, the mapping can be also carried out based on the data exchanging edge between nodes. As shown in Fig. 6(b), each edge which has data flows over it in a network will be simulated and analysed with a computation unit from Cloud. For example, the edge ab, ac and cd will be allocated with similar computation and storage units respectively. It is clear that the mapping in Fig. 6(b) still cannot solve the problem of unfair distributed data size over each data exchanging edge. Furthermore, the number for the network edges may increase dramatically with the growth of the nodes. Though the cluster-head architecture and 1-hop cluster somehow limit the increasing number for the edges, to process the data graph in a big cluster itself can be a very time consuming job for the Cloud. So, the edge mapping and its scheduling for the Cloud computing power is not suitable for analysing the data exchange cluster in Fig. 6, either.

4.1.3. Mapping for Could scheduling with data exchange quantity

Based on the mapping in Fig. 6(b), we design a novel mapping based on the information carried by the compressed data sets after the processing of our proposed data reduction with spatial clustering, temporal prediction and order compression.

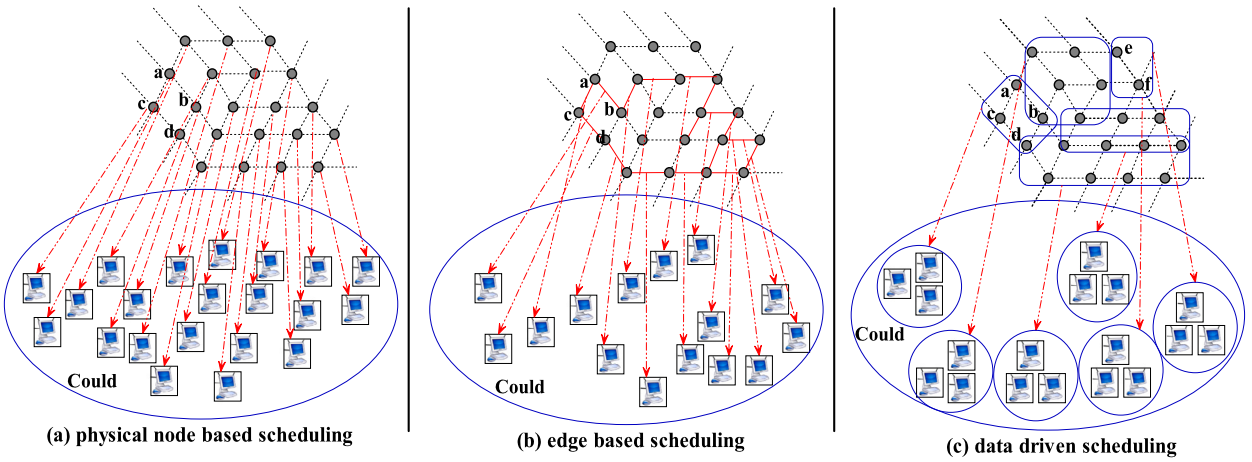

Fig. 6. Comparison between different mappings for computation resource scheduling over Cloud.

As shown in Fig. 6(c), the computation resources are grouped into to some basic units to offer independent processing power capability. Then, the edges in a network will be divided into to several blocks according to their real workload of data flow. For example, according to data exchanging size, clustering situation and compression ratio, the workload for data exchanging and processing over the edges ab, ce, bd, ac is similar to the half workload over the edge ef. It can be concluded that data exchange over the edge ef is of high data exchanging density and difficulty to be compressed and clustered. Hence, the edge ef is grouped into an independent group and allocated with more computation resources from the Cloud platform. With the data driven mapping in Fig. 6(c), it is clear that the mapping algorithm mainly focuses on how to assign the resources according to the real requirement for dealing with data. It manages to distribute the workload for data processing more evenly over the Cloud platform and to make the full use of the computation power of Cloud. Compared to the mapping strategies in Fig. 6(a) and (b), it is more optimal in terms of time saving and resource dispatching.

4.2. Calculation for weighted data exchanging edges

In order to carry out the scheduling strategy introduced in Fig. 6(c), we need to calculate the data exchanging quantity for each edge in a network topology as shown in Fig. 7. According to our work introduced in the previous sections, the data set will be clustered and compressed before being used as a base for offering real time Cloud services. The calculation of the workload for each edge in a network structure is as follows.

It has been introduced that during the filtering process of big data set, clustering and compression have been conducted. With that processing the data suppression ratio based on clustering, denoted as C_s for an edge is calculated. The data suppression ratio based on temporal compression, denoted as C_t is also calculated. For example, the clustering suppression ratio over edge ab is denoted as C_{sab} and the order compression ratio over edge ab is denoted as C_{tab}. Because the clustering and compression have different influence to the data reduction over the edge ab, different weights w_{sab} and w_{tab} are assigned to C_{sab} and C_{tab} respectively. With that we can calculate the real workload of data over edge ab, $D'_{ab} = (w_{sab} \times C_{sab} + w_{tab} \times C_{tab}) \times D_{ab}$ where $w_{sab} + w_{tab} = 1$. The selection of w_{sab} and w_{tab} can come from application requirement or system feedback.

Suppose there are n edges in the network, the total real data exchanging size $D' = E(w_s \times C_s + w_t \times C_t)$, $E \in G(V, E)$. G is the network graph, V is set for nodes and E is edge set. Then, we can get the percentage of data processing workload on each edge. Suppose that the whole computation resource of Cloud is R, the data processing task can be scheduled for over Cloud according to the calculated $D'_{xy} = (w_{sxy} \times C_{sxy} + w_{txy} \times C_{txy}) \times D_{xy}$, where x and y are two nodes in a data exchanging network. As shown in Fig. 7, the exchanging data size over the edge ef is $D'_{ef} = (w_{sef} \times C_{sef} + w_{tef} \times C_{tef}) \times D_{ef}$ and that over the edge ab is $D'_{ab} = (w_{sab} \times C_{sab} + w_{tab} \times C_{tab}) \times D_{ab}$.

5. Spatiotemporal compression based approach

Based on the above compression techniques and scheduling detailed in Sections 3 and 4, the overall approach for efficient big data processing on Cloud is designed in this section. Specifically, we design related separate algorithms and roadmap for the proposed spatiotemporal Cloud data compressions and afterwards scheduling, which constitute the approach.

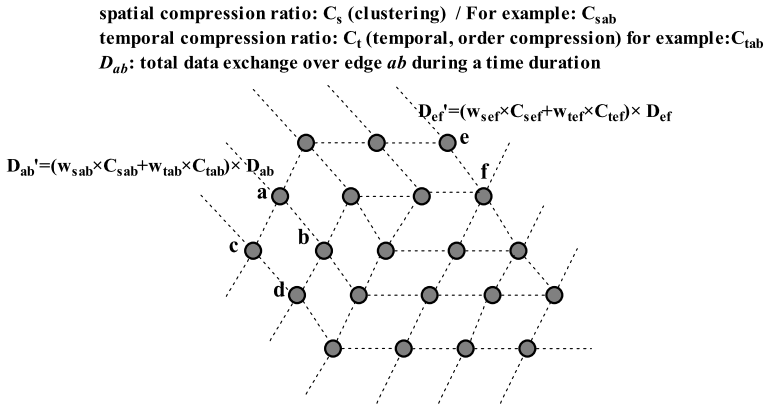

Fig. 7. Computation model for each edge based on temporal and spatial compression ratio.

5.1. Spatiotemporal clustering algorithm

The clustering algorithm is developed on the cluster-head. It takes time series set X and similarity threshold d as inputs. The output is a clustering result which specifying each cluster-head node and its related leaf nodes.

Clustering Algorithm with Data Trend Similarity.

Input: a vector set $X = \{x_1, x_2, ..., x_n\}$, $(v_i = \{v_{i1}, v_{i2}, ..., v_{im}\})$,
x_i is a time series of node i,
disimilarity threshold: d, clustering round: k, $i \in [1, n]$, $j \in [1, m]$;
Output: cluster-head nodes set S *and cluster information*;

(1) While (time stamp $j <= k$)
(2) if ($j\%R == 0$ && $R! = 0$) // R is for reclustering and not the first round
(3) Normalize X to V; $V = X \times W$;
(4) select $\forall$ unselected v_i from set V;
(5) initialize a cluster C_i with v_i; record v_i in S;
(6) While (existing unselected element in S)
(7) select $\forall$ unselected x_i, from set X; transform x_i to v_i;
(8) comparesimilarity(v_j, S);
(9) if (comparesimilarity(v_j, S) $> d$)
(10) initialize C_j with v_j; record v_j in S_j;
(11) else
(12) adding v_i into the C_l with minimum similarity;
(13) if (all nodes have been traversed)
(14) return S;

As shown in the clustering algorithm, in line (1), the data exchange time stamp j is counted from 1 to the data exchange application duration, k. In line (2), if the j is the time stamp for re-clustering, the clustering process will be executed. In line (3), the algorithm changes the computation of Matrix X to data change matrix V with the technique introduced in Section 3.2. From line (4) to (5), the algorithm selects any vector from V to form the first cluster C_i and node x_i will be used as the cluster-head of C_i. From line (6), the algorithm will carry out the selection of a new vector until all the vectors are clustered. Specifically, it selects a vector x_j from X and transforms it into v_j in line (6) to line (7). In line (8), for each newly selected v_j, it will be compare to previous clustered vectors, if the similarity between the is within the given threshold d, v_i and x_i will be added into a previous cluster line (11) to line (12). If the similarity exceeds the given threshold d, a new cluster C_j will be initialized with v_j or x_j as its cluster-head as shown in line (9) to line (10). Finally, it all the nodes have been compared and clustered, the algorithm will return the final clustered nodes set and clustering information S which is a partition plan for X.

5.2. Algorithms for spatiotemporal order compression

5.2.1. Order compression

In order to set up the relationship of the reported data and compressed data, the compression algorithm partitions the n reports from n nodes into two groups X_t and X_s. After this partition, the algorithm aims to construct a mapping between

sets X_t and X_s. Specifically, for a given historical buffer length L, s is calculated for minimum report of order information indexing in line 1 and line 2. In line 3, the initialization is conducted. From line 4 to line 8, the first round partitioning for X and data order mapping is carried out for indexing the elements in X_t with the combination order of the elements in X_s. From line 9 to line 10, the algorithm will recursively use the indexing concept from line 4 to line 8, to mapping the rest of data in X. The compressed elements from X will be also sent to X_s.

Algorithm: Order Compression.
Input: a streaming graph data set $X = \{x_1, x_2, ..., x_n\}$, window size: L;
Output: a partition of data set X, ordered transmission set X_i and suppression set X_s;

1. for an $x_i \subset X$, to index its buffer with L, the required reports size $s! \geq L$;
2. return the minimum s;
3. initialize $X_t = \varnothing$ and $X_s = \varnothing$, $i = 1$; $j = n$;
4. while (sizeof(X) >= $s + 1$)
5. ordering($x_i, x_{(i+1)}, ...x_{(i+s)}$) $\Rightarrow X_t$;
6. $x_{(n-i)} \Rightarrow X_s$;
7. delete $x_i, x_{(i+1)}, ...x_{(i+s-1)}, x_{(j)}$ from X;
8. $i = i + s$; $j = j - 1$;
9. while (sizeof($X_t >= s$ && $X! = \varnothing$))
10. iteration ordering(X_t) for each element from X; $x_k \Rightarrow X_s$;
11. Return final X_t and X_s;

5.2.2. Temporal prediction with order compression

Algorithm: Adding Order Compression to Temporal Prediction.
Input: real-time collected streaming data a_t, $a_t = \{att_1, att_2, ..., att_m\}$
prediction error bound c
Output: real-time collected data a_t, a'_t which has a smaller size than at,
or a suppression of a_t

1. check the temporal inference error bound c;
2. for (int $i = 1$; $i <= m$; i++) {
3. if ($|att_j - att'_j| < c$)
4. carry out normal in-network prediction;
5. else
6. if ($att_j \approx att''_j$) // if att_j can be found close to a value att''_j in history table
7. using the order of previous $j - 1$ attributes to represent the location of "att_j";
8. order compression in node and inter nodes
9. } end for

The order compression algorithm is developed based on the data history buffering strategy. It should be embedded in the original temporal in-network prediction. The input of the algorithm includes the real-time collected data of each node at time interval t. In line 1, the prediction coherency c is checked in line 1 for regular in-network prediction. In line 2, the compressive algorithm checks the reported attributes. If there is one attribute att_j has a similar value to a certain value, att_j stored in its data history table in lines 7 and 8, the order of the previous reported $j - 1$ attributes will be used to report the location of att_j in the data history table. In other words, the order compression will be carried out inter and inner network nodes.

5.3. Algorithm for scheduling with weighted data edges

Based on the above definition and computation, the scheduling algorithm is offered as follows. The algorithm takes the graph data D and its topology G, Could resources as inputs, outputs a scheduling based on partition of R. From line 1 to line 6, the algorithm carries out initialization and creates the first partition for scheduling. From lines 7 to 11, a partition can have more edges if the computation power is not fully used. From lines 12 to 13, if there is an overloaded data exchanging edge, more computational power from Cloud will be allocated. This process will be repeatedly executed over all the edges from the set E.

Algorithm: Data Driven Scheduling.
Input: a big graph $G(V, E)$, a graph data set D, Cloud platform R
Output: a division P of R based on E from G

```
0.  i = 0;
1.  While (E! = ∅) {
2.    get e_xy ∈ E;
3.    D'_xy = (w_sxy × C_sxy + w_txy × C_txy) × D_xy
4.    resource allocation based on k = D'_xy/D' × R;
5.    Create a New Partition P_i;
6.    Add e_xy to P_i;
7.    if (e_xy < k)
8.      get new e_xy ∈ E;
9.      D'_xy = (w_sxy × C_sxy + w_txy × C_txy) × D_xy
10.     Add e_xy to P_i;
11.     i++;
12.   else if (e_xy ≥ 2k)
13      Pi ->2kR;
14      i++;
15. } end While
```

5.4. Overall strategy for Cloud big data processing

With the above offered algorithms including compression and scheduling working at the different stage of our proposed big data Cloud processing stage, the roadmap for using this approach can be described as follows. 1. Input a big data set to Cloud platform. 2. Filter the big data set with the data compression algorithms. During the compression, the spatiotemporal correlations between data are used. The clustering and order compression are combined together for better suppression effect when processing data sets with different distributions. 3. The filtered and compressed data/graph data will be partitioned and distributed on Cloud according to our scheduling algorithm for further data processing as service providing.

6. Evaluation and comparison

To verify the effectiveness of the proposed spatiotemporal compression and it related scheduling strategy for processing big streaming data and graph data over Cloud, experiments are designed based on U-Could (Cloud computing environment at University of Technology Sydney) [15–18]. The streaming data set from a real wireless sensor network system is used for testing the performance gains of the algorithms. The collected data sets will be filtered with our on-fly compression and clustering algorithm over Cloud environment. The time cost and computation resource cost will be recorded to measure the performance gains. Compared to previous big data processing techniques without spatiotemporal compression and its related scheduling, the evaluation is designed to demonstrate the following gains. 1. The new approach significantly outperforms previous one in terms of Cloud resource cost by data compression and avoiding iteration. 2. The new approach can save data processing time significantly by a fair workload distribution strategy. 3. The new approach will not introduce unacceptable data quality loss to most of real world applications.

6.1. Experiment environment and process

In this data exchange sensor network for our testing data, there are 500 nodes deployed in our physical world to conduct tasks such as data gathering, information exchanging and mutual interaction. The nodes are organized as a hierarchical structure with cluster-head and leaf node. Every node collects high frequency data streams such as sounds and vibration. Every node also collects low frequency data streams such as light and temperature. Because sensor networks is based on the wireless communication, there exist huge amount of information errors, loss or redundancy. In each cluster, the data exchange creates a complicated data graph. If we can use Cloud as a powerful tool to simulate and analyse that complicated data flows in that data graph, the better WSN design and more efficient real time query can be offered.

As shown in Fig. 8, even only considering one node, four types of data can be gathered with different frequency. For easily computing the similarity between time series from different nodes, they have been normalized into the same value domain from 1 to 100. Within a 24 hours window, the vibration and sound time series have high sampling frequency and experience dramatic changes. The light and temperature time series have relative smooth data trend and are sampled with a relatively low frequency. The time duration for the testing data set is 24 hours. In each second, the sampled data flow over a node is 0.02 KB on average. In total, there are around 1,000,000 KB data sampled by the whole network. However, under the fine wireless communication channels environment, to monitor and simulate the network behaviour, the overall exchanging data size between nodes exceeds 5,000,000 KB for transmitting the collected “vibration”, “sound”, “light” and “temperature” streaming graph data of 1,000,000 KB. The extra 4,000,000 KB is caused by the communication protocols, gossip between

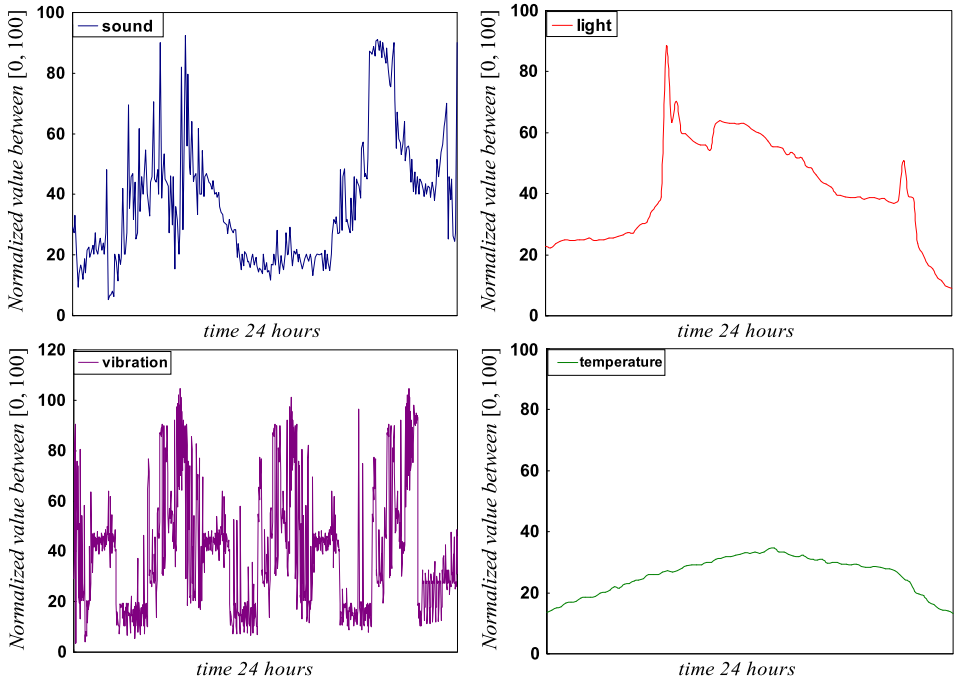

Fig. 8. Heterogeneous environment data sets from 1 physical monitoring node (KB/sec./per node).

nodes, failures, data loss, re-transmission, overhearing and redundancy, etc. However, the light and temperature time series experience relatively smooth changes within time duration of 24 hours. In other words, the data sampling by each real world node is heterogeneous.

According to the curve in Fig. 8, the high frequency sound and vibration time series fluctuate dramatically when considering one node. It indicates that regular temporal prediction may have less effect in compressing the size of the time series. Specifically, the sampling frequency for vibration is 10 bytes per-second and the sampling frequency for sound is 8 bytes per-second. However, the data sampling frequency for light and temperature is relatively low with only 1 byte per-second. In addition to these data, extra communication consumes large amount of computation and storage resource on Cloud. According to our approach, before sending these data streams to Cloud for processing, data sets will be compressed and clustered to reduce the big data size and the task scale for Cloud platform. In total, three groups of experiments are carried out over U-Cloud with the above experiment data sets. In the first group, we test the compression effectiveness. In the second group, we test the effectiveness of the scheduling algorithm. In the third group, we test the accuracy and data loss problems for our approach through the definitions of data quality, such as average accuracy.

6.2. Experiment for the spatiotemporal compression

Firstly, we test the clustering algorithm and compression algorithms with high frequency vibration and sound data sets. As mentioned above, the total real world data exchange within the whole network exceeds 5,000,000 KB. Because the data collection is based on a heterogeneous and asynchronous model, the high frequency data sets of sound and vibration count for 90% data exchange in the real world network. It can be estimated that the real world network data exchange for vibration and sound is around 4,500,000 KB. Both compression and clustering data reduction techniques have a great effect on big streaming data reduction based on the results in Fig. 9 (1), (2), (4) and (5). However, clustering algorithm performs slightly better than order compression for sound data set because for monitoring a vibration system, such as helicopter blades, in a certain time period, they maintain a set of similar sound curves which are very suitable for clustering based data reduction.

In Fig. 9 (3) and (6), the adaptively using clustering and order compression can achieve better performance gains compared to independent clustering and compression based data reduction algorithms over both sound and vibration data sets. In Fig. 9 (3), only 800,000 KB sound data is exchanged, and in Fig. 9 (6), around 1,100,000 KB vibration data is exchanged. Compared to the data size of 4,500,000 KB exchanged in the real world network, significant amount of data exchange is avoided. In other words, with the clustering and order compression, around 60% big graph data from the high frequency vibration and sound data sets can be compressed. This significant reduction will undoubtedly lead to the time cost and computation source saving when analysing the reduced data sets over Cloud platform.

Secondly, we test the clustering algorithm and compression algorithms with low frequency light and temperature data sets. According to above analysis, the high frequency sound and vibration data exchange within the whole network exceeding 4,500,000 KB. It can be easily calculated that the total data exchange for the real world network low frequency light

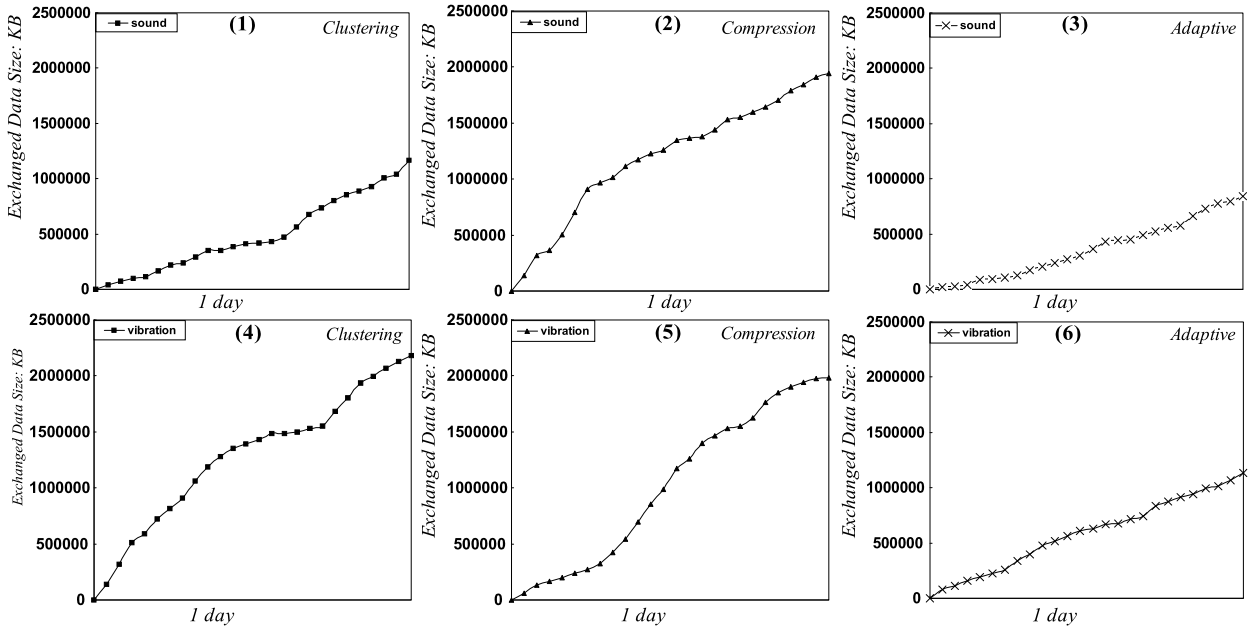

Fig. 9. Exchanging data reduction over high frequency sound and vibration data sets.

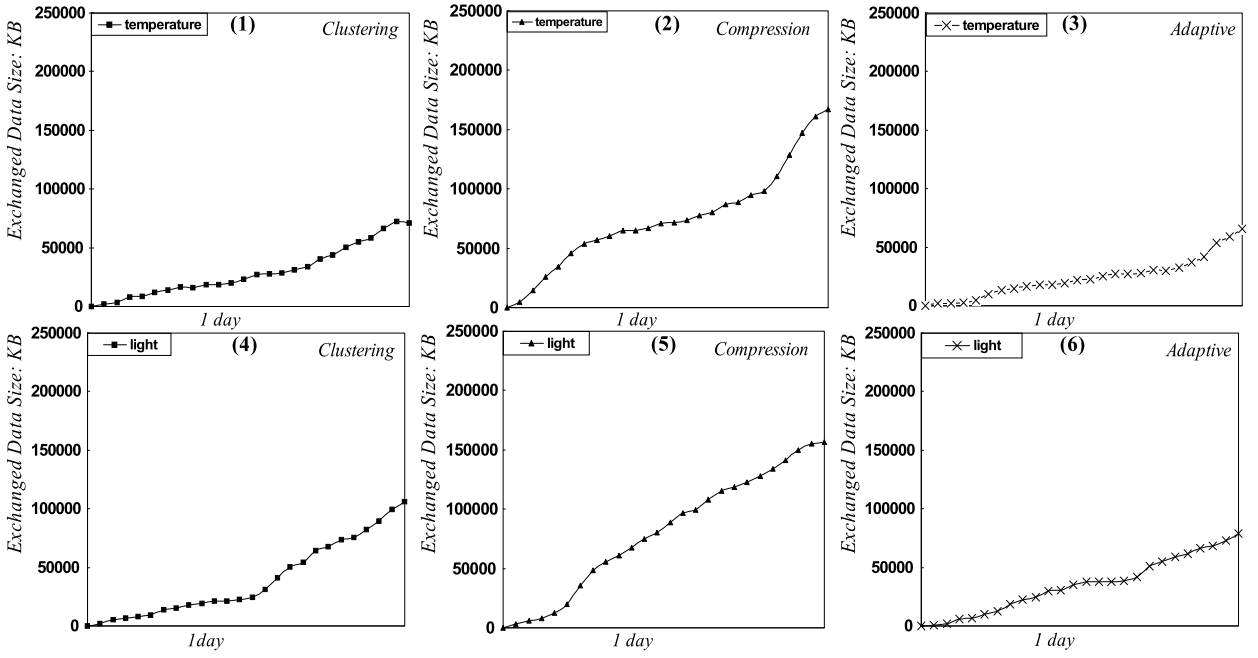

Fig. 10. Exchanging data reduction over light and temperature data sets.

and sound data sets are around 500,000 KB. According to experimental results demonstrated in Fig. 10 (1), (2), (4) and (5), it can be concluded that both clustering and order compression have slightly better reduction effects over the temperature data set than that over the light data set. The reason is that the temperature within one data of a small area keeps quite similar and maintains a constant difference between time series which can be accurately described with our data prediction models for clustering and order compression. Specifically, with the clustering algorithm, around 65,000 KB data is exchanged for the temperature data set; and around 100,000 KB data is exchanged for the light data set. With the order compression algorithm, the exchanged data sizes are 160,000 KB and 150,000 KB for temperature and light data set respectively.

However, in Fig. 10 (3) and (6), the adaptively using clustering and order compression can achieve better performance gains compared to independent clustering and compression based data reduction algorithms over both temperature and light data sets. In Fig. 10 (3), around 65,000 KB sound data is exchanged, and in Fig. 10 (6), around 70,000 KB vibration data

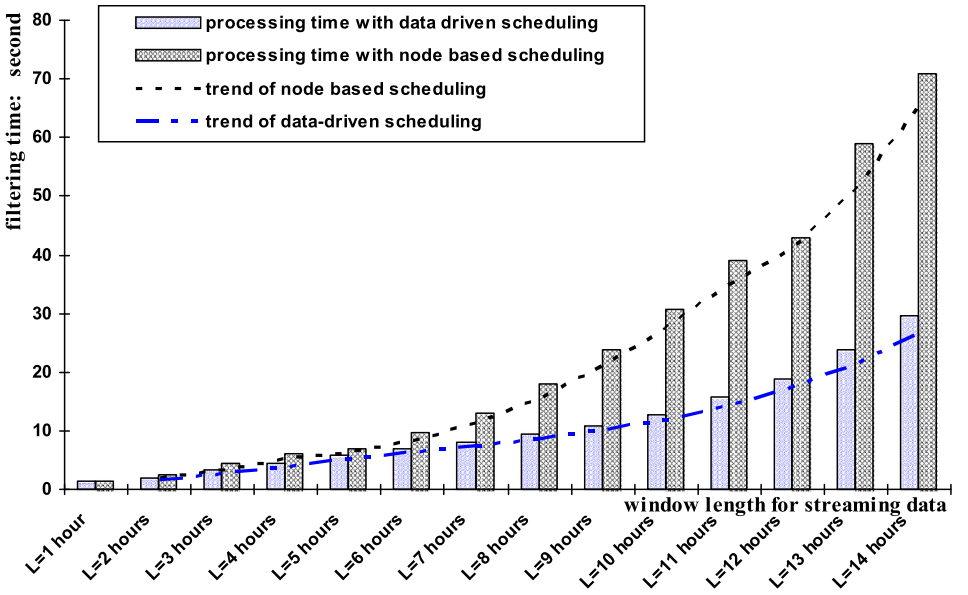

Fig. 11. Processing time for streaming data graph with different window length '*L*'.

is exchanged. Compared to the data size of 500,000 KB exchanged in the real world network, significant amount of data exchange is avoided. In other words, with the clustering and order compression, more than 70% big graph data from the low frequency vibration and sound data sets can be compressed. Compared to the high frequency data sets in the whole testing big graph data set, more data is compressed because the low frequency data sets are much easier to model and predict.

6.3. Experiment for scheduling time performance gains

In the above Section 6.2, we demonstrated that the order compression and clustering algorithm can effectively reduce the data size for future analysis over Cloud platform. A smaller data set means a smaller problem size over a distributed processing platform. But it does not definitely mean a shorter processing and reaction time which are normally determined by the time cost of the final node to finish its task branch in a distributed system such as Cloud. So, scheduling is necessary to dispatch the tasks more evenly and optimal over Cloud platform. In other words, the time consumption situation and real time reaction are good to measure the performance of scheduling algorithm.

As shown in Fig. 11, the horizontal axis stands for the window length which is necessary for most streaming data applications. With the data from a given window length '*L*', the data analysis algorithm will work on the Cloud to digest it and form the final services to the end users. The vertical axis stands for the time cost calculated in second. In this experiment, we implement two scheduling algorithms. They are original scheduling based on the physical topology of the real world network. Because this scheduling ignores the real workload of each node, some nodes over Cloud may be overloaded and others may be idle to wait for those overloaded nodes. Especially after the clustering and order compression, the data stream size flowing through each real world network node can be extremely different. As a result, this physical topology node based scheduling costs more time for filtering the whole streaming data set and simulating the data exchanging compared to our proposed data driven scheduling algorithm in every different window length from 1 hour to 14 hours. The trend curve of our node based scheduling keeps running below the trend curve of the node based scheduling. The larger the window size is, the more filtering time is saved.

However, it is not reasonable to test the window too larger over 14 hours, because it causes a very big data graph which may introduce lots of approximation and failures by magnifying the acceptable errors in our data reduction models. Furthermore, an extra-longer window size makes the filtering time increase exponentially. For example, in Fig. 11, when $L = 14$ hours, the time costs of our data driven scheduling and the real world node based scheduling are around 30 seconds and 70 seconds respectively. Compared to the 7 hours window with the time costs around 8 seconds and 12 seconds, the filtering time and system reaction time is greatly increased. There is an important point to be mentioned here for the experiment results in Fig. 11. With the increase of the stream window length '*L*', the time cost does not increase with a linear relationship; whereas an approximate exponential relationship can be observed. The reason for that exponential increase is that with the increased window length, more nodes from different time stamps join the network graph to change the topology with more complexity.

6.4. Experiment for data quality and fidelity loss

In previous sections, we demonstrated the effectiveness of our clustering and compression techniques for reducing big data size. In addition, we conducted the experiment to test our scheduling algorithm with respect to its time cost. However, because the data reduction process of our techniques is not lossless, the fidelity problem is critical to be discussed to guarantee the service quality offered to end users. In this section the experiment will be conducted to show the fidelity loss and data quality after deploying our data reduction and scheduling over Cloud platform. We aim to prove that under most of applications, our algorithm can achieve efficient big data processing on Cloud without losing acceptable accuracy for most of applications.

6.4.1. Definition for accuracy

Before introducing and analysing the experiment results, we first offer the definition and method for accuracy. The accuracy is based on measuring the similarity between two vectors, one from real big data graph G and the other G' from filtered data as the service provided by Could. Actually, these two vectors are the data items flowing between two nodes within a cluster at a certain time stamp. To describe the similarity between two nodes, we use correlation coefficient model. Suppose that there are two vectors, X and Y. With Correlation Coefficient method, we can calculate the similarity between the two vectors, X from G and Y from G' in formula (7).

$$\mathrm{sim}(X,Y) = r(X,Y) = \frac{\mathrm{cov}(X,Y)}{\sqrt{\mathrm{cov}(X,X)\cdot\mathrm{cov}(y,y)}} \tag{7}$$

From (7) we can find that this similarity resembles to the "cos" similarity computation. $\mathrm{sim}(X,Y)$ has a data range $[-1,1]$. The calculation of "cov(vector1, vector2)" is carried out in formulas (8) to (10).

$$\mathrm{cov}(X,Y) = \frac{1}{n-1}\sum_{i=1}^{n}(X_i-\overline{X})(Y_i-\overline{Y}) \tag{8}$$

$$\mathrm{cov}(X,X) = \frac{1}{n-1}\sum_{i=1}^{n}(X_i-\overline{X})^2 \tag{9}$$

$$\mathrm{cov}(Y,Y) = \frac{1}{n-1}\sum_{i=1}^{n}(X_i-\overline{X})(Y_i-\overline{Y}) \tag{10}$$

So, the similarity between two vectors can be calculated with following formula (11).

$$\mathrm{sim}(X,Y) = \frac{\sum_{i=1}^{n}(X_i-\overline{X})(Y_i-\overline{Y})}{\sqrt{\sum_{i=1}^{n}(X_i-\overline{X})^2(Y_i-\overline{Y})^2}} \tag{11}$$

Because we only want to correlate the accuracy and similarity, only $[0,1]$ data range is selected. The original data range $[-1,1]$ could be normalized into $[0,1]$ for representing the accuracy from 0% to 100%. As shown in formula (12), $\mathrm{sim}(X,Y)'$ is calculated instead of formula (11). $\mathrm{sim}(X,Y)' \in [0,1]$.

$$\mathrm{sim}(X,Y)' = \frac{|\sum_{i=1}^{n}(X_i-\overline{X})(Y_i-\overline{Y})|}{\sqrt{\sum_{i=1}^{n}(X_i-\overline{X})^2(Y_i-\overline{Y})^2}} \tag{12}$$

Hence the accuracy for an edge in G at a time stamp t can be assessed with the formula (13).

$$\mathrm{accuracy} = \mathrm{sim}(X,Y)'\times 100\% = \frac{|\sum_{i=1}^{n}(X_i-\overline{X})(Y_i-\overline{Y})|}{\sqrt{\sum_{i=1}^{n}(X_i-\overline{X})^2(Y_i-\overline{Y})^2}}\times 100\% \tag{13}$$

The final accuracy for "Accuracy" for data service quality between two points within a cluster can be assessed with the Accuracy in formula (14). In our example, $T = 24$ hours is used.

$$\mathrm{Accuracy} = \sum_{t=0}^{T}\left(\frac{|\sum_{i=1}^{n}(X_{it}-\overline{X}_t)(Y_{it}-\overline{Y}_t)|}{\sqrt{\sum_{i=1}^{n}(X_{it}-\overline{X}_t)^2(Y_{it}-\overline{Y}_t)^2}}\right)/T\times 100\% \tag{14}$$

Suppose that in the graph data set $G(V,E)$, there are total S edges (because the cluster-head structure, it avoids the edge explosion) and each edge is index 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 (15) to demonstrate our experiment results in Fig. 12 and Fig. 13.

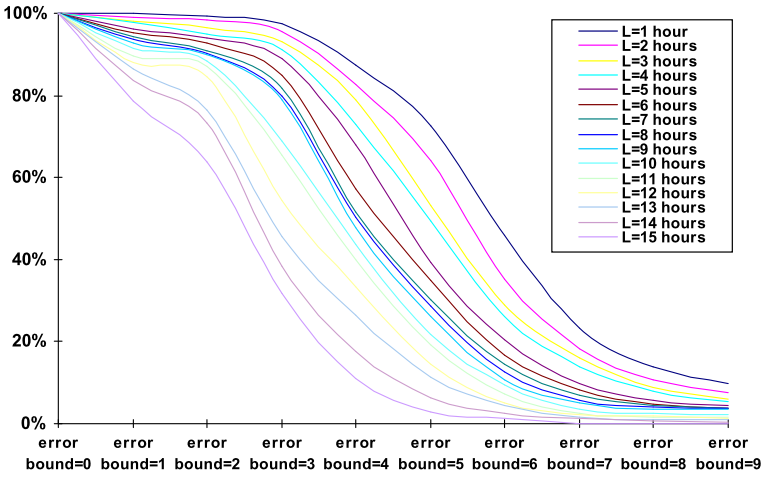

Fig. 12. Fidelity loss with the error bound increase.

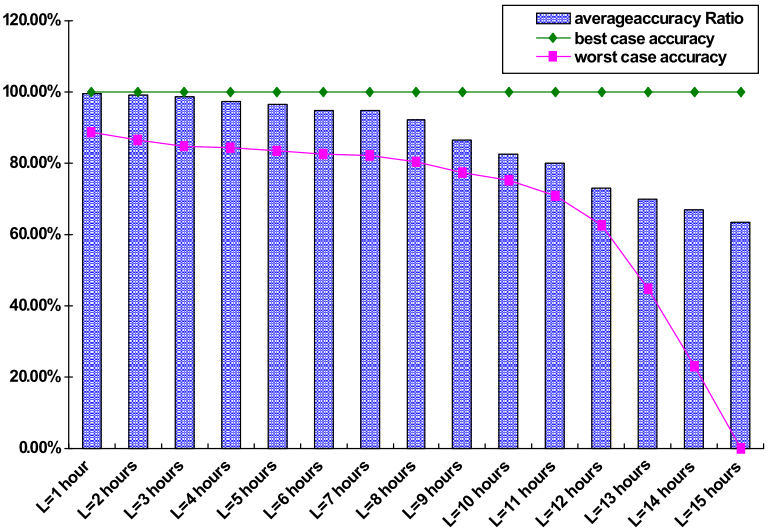

Fig. 13. Comparison of fidelity loss with worst and best accuracy.

$$\text{Average\_Accuracy} = \sum_{s=1}^{S} (\text{Accuracy})_s \tag{15}$$

6.4.2. Data accuracy analysis

The inaccuracy in our technique based on Cloud is mainly caused by the error from clustering process and data compression loss. Both data window length L and prediction error bounds have influence to the accuracy and final service quality. As shown in Fig. 12, the data accuracy decreases with the increase of the given prediction error bound. However, there is another factor, window length 'L' which influences the accuracy. It can be found in Fig. 12 that with the increase of the historical data window length 'L', the accuracy curve runs totally at a low accuracy level. That is caused by the large window in which more nodes and data are involved in simulating the graph data. During that simulating process, the error and failures will have a magnifying effect when repetitively exchanging among more nodes. From the experiment result in Fig. 12, it can be concluded that when the given error bound is less than 2 (after normalization) and the window length is less than 12 hours, the accuracy and data quality of service are within an acceptable level, around 95%. With the error bound larger than 2 and the window length larger than 12 hours, the data quality for service after processing over Cloud platform decreases dramatically. Due to the big errors, there is no need to plot the error bound larger than 9 which is totally useless in real applications. So, we used an empirically study to select 2 as an optimal number for our algorithm. It is also used in the related experiments for data reduction and scheduling.

Specifically, we use as the average value from 0 to 9 for accuracy test. As shown in Fig. 13, with the increase of window length L, the average accuracy decreases dramatically. Whatever the window length is, the best case accuracy can

be achieved as 100%. Best case accuracy is the highest accuracy of a data item offered as the service after the processing over Cloud. However, with the increase of the window length, the worst case accuracy decreases dramatically. Worst case accuracy is the lowest accuracy of a data item offered as the service after the processing over Cloud. It is measured and calculated with the correlation coefficient similarity. Especially, when the window length enlarges to more than 11 hours, the worst cased accuracy drops from 80% to 0 very quickly. In other words, according to the experiment results in Fig. 13, a 2 or 3 hours data window is better for guaranteeing the data accuracy and service quality for deploying our proposed big data technique over Cloud.

7. Conclusions and future work

Cloud promises an ideal platform with massive computation power and storage capacity for processing big data that is of high variety, volume, veracity, and velocity. To reduce the quantity and the processing time of big data sets encountered by the current typical Cloud big data processing techniques, in this paper, we proposed a spatiotemporal compression based approach on Cloud to deal with big data and big graph data from real world applications. In our technique, the big data was compressed firstly according to its spatiotemporal features on Cloud. Based on this spatiotemporal compression, a data driven scheduling was developed to allocate the computation and storage of Cloud for providing better big data processing services. The evaluation was conducted over our U-Cloud platform to demonstrate that the spatiotemporal compression could significantly reduce the data size compared to the previous big data processing techniques on Cloud. Furthermore, our data driven scheduling distributed big data processing workload optimally on Cloud to achieve significant time performance gains. Last but not least, the evaluation results also demonstrated that the data processing quality and fidelity loss of our proposed approach met most of the application requirements.

Data compression can be based on different data correlations or data models. As future work, more spatiotemporal correlations should be exploited and modelled to effectively reduce different big data sets. In accordance with various data correlations and compression models, the scheduling should also be changed dynamically to generate fair workload distribution and to achieve optimal time performance.

References

[1] S. Tsuchiya, Y. Sakamoto, Y. Tsuchimoto, V. Lee, Big data processing in Cloud environments, Fujitsu Sci. Tech. J. 48 (2) (2012) 159–168.
[2] C. Ji, Y. Li, W. Qiu, U. Awada, K. Li, Big data processing in Cloud environments, in: 2012 International Symposium on Pervasive Systems, Algorithms and Networks, 2012, pp. 17–23.
[3] Big data: science in the petabyte era, Nature 455 (7209) (2008) 1.
[4] Balancing opportunity and risk in big data, a survey of enterprise priorities and strategies for harnessing big data, http://www.informatica.com/Images/1943_big-data-survey_wp_en_US.pdf, accessed on March 01, 2013.
[5] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, M. Zaharia, A view of cloud computing, Commun. ACM 53 (4) (2010) 50–58.
[6] R. Buyya, C.S. Yeo, S. Venugopal, J. Broberg, I. Brandic, Cloud computing and emerging it platforms: vision, hype, and reality for delivering computing as the 5th utility, Future Gener. Comput. Syst. 25 (6) (2009) 599–616.
[7] L. Wang, G. Von Laszewski, A. Younge, X. He, M. Kunze, J. Tao, C. Fu, Cloud computing: a perspective study, New Gener. Comput. 28 (2) (2010) 137–146.
[8] L. Wang, J. Zhan, W. Shi, Y. Liang, In cloud, can scientific communities benefit from the economies of scale?, IEEE Trans. Parallel Distrib. Syst. 23 (2) (2012) 296–303.
[9] X. Yang, L. Wang, G. Laszewski, Recent research advances in e-science, Clust. Comput. 12 (4) (2009) 353–356.
[10] S. Sakr, A. Liu, D. Batista, M. Alomari, A survey of large scale data management approaches in cloud environments, IEEE Commun. Surv. Tutor. 13 (3) (2011) 311–336.
[11] B. Li, E. Mazur, Y. Diao, A. McGregor, P. Shenoy, A platform for scalable one-pass analytics using MapReduce, in: Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'11), 2011, pp. 985–996.
[12] J. Dean, S. Ghemawat, MapReduce: simplified data processing on large clusters, Commun. ACM 51 (1) (2008) 107–113.
[13] K.H. Lee, Y.J. Lee, H. Choi, Y.D. Chung, B. Moon, Parallel data processing with MapReduce: a survey, SIGMOD Rec. 40 (4) (2012) 11–20.
[14] Hadoop, http://hadoop.apache.org, accessed on March 01, 2013.
[15] X. Zhang, C. Liu, S. Nepal, J. Chen, An efficient quasi-identifier index based approach for privacy preservation over incremental data sets on Cloud, J. Comput. Syst. Sci. 79 (5) (2013) 542–555.
[16] X. Zhang, C. Liu, S. Nepal, W. Dou, J. Chen, Privacy-preserving layer over MapReduce on Cloud, in: The 2nd International Conference on Cloud and Green Computing (CGC 2012), Xiangtan, China, 2012, pp. 304–310.
[17] X. Zhang, C. Liu, S. Nepal, S. Pandey, 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. 24 (6) (2013) 1192–1202.
[18] X. Zhang, T. Yang, C. Liu, J. Chen, A scalable two-phase top-down specialization approach for data anonymization using MapReduce on Cloud, IEEE Trans. Parallel Distrib. Syst. 25 (2) (2013) 363–373.
[19] C. Yang, Z. Yang, K. Ren, C. Liu, Transmission reduction based on order compression of compound aggregate data over wireless sensor networks, in: Proc. 6th International Conference on Pervasive Computing and Applications (ICPCA'11), Port Elizabeth, South Africa, 2011, pp. 335–342.
[20] S.H. Yoon, 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. (2008) 1–36.
[21] P. Edara, A. Limaye, K. Ramamritham, Asynchronous in-network prediction: efficient aggregation in sensor networks, ACM Trans. Sens. Netw. 4 (4) (2008), article 25.
[22] M.J. Handy, M. Haase, D. Timmermann, An low energy adaptive clustering hierarchy with deterministic cluster-head selection, in: Proc. 4th International Workshop on Mobile and Wireless Communications Network (MWCN), 2002, pp. 368–372.
[23] A. Ail, A. Khelil, P. Szczytowski, N. Suri, An adaptive and composite spatio-temporal data compression approach for wireless sensor networks, in: Proc. of ACM MSWiM'11, 2011, pp. 67–76.
[24] R. Kienzler, R. Bruggmann, A. Ranganathan, N. Tatbul, Stream as you go: the case for incremental data access and processing in the cloud, in: IEEE ICDE International Workshop on Data Management in the Cloud (DMC'12), 2012.

[25] P. Bhatotia, A. Wieder, R. Rodrigues, U.A. Acar, R. Pasquin, Incoop: MapReduce for incremental computations, in: Proceedings of Proceedings of the 2nd ACM Symposium on Cloud Computing (SoCC'11), 2011, pp. 1–14.
[26] 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, X. Wang, Nova: continuous Pig/Hadoop workflows, in: Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'11), 2011, pp. 1081–1090.
[27] S. Lattanzi, B. Moseley, S. Suri, S. Vassilvitskii, Filtering: a method for solving graph problems in MapReduce, in: Proc. 23rd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA'11), San Jose, California, USA, 2011.
[28] J. Conhen, Graph twiddling in a MapReduce world, Comput. Sci. Eng. 11 (4) (2009) 29–41.
[29] K. Shim, MapReduce algorithms for big data analysis, Proc. VLDB Endow. 5 (12) (2012) 2016–2017.
[30] Big data beyond MapReduce: Google's big data papers, http://architects.dzone.com/articles/big-data-beyond-mapreduce, accessed on March 01, 2013.
[31] Real time big data processing with GridGain, http://www.gridgain.com/book/book.html, accessed on March 03, 2013.
[32] Managing and mining billion-node graphs, http://kdd2012.sigkdd.org/sites/images/summerschool/Haixun-Wang.pdf, accessed on March 05, 2013.