An Optimized Iterative Semantic Compression Algorithm And Parallel Processing for Large Scale Data

Ran Jin[1, 2], Gang Chen[2], Anthony K. H. Tung[3], Lidan Shou[2] and Beng Chin Ooi[3]
[1] School of Electronic and Computer, Zhejiang Wanli University
Ningbo, Zhejiang 315100 - China
[e-mail: ran.jin@163.com]
[2] College of Computer Science and Technology, Zhejiang University
Hangzhou, Zhejiang 310027 - China
[e-mail: {cg, should}@zju.edu.cn]
[3] School of Computing, National University of Singapore
Singapore, Singapore 119077 - Singapore
[e-mail: {atung, ooibc}@comp.nus.edu.sg]
*Corresponding author: Ran Jin

Received July 25, 2017; revised November 27, 2017; accepted February 10, 2018; published June 30, 2018

Abstract

With the continuous growth of data size and the use of compression technology, data reduction has great research value and practical significance. Aiming at the shortcomings of the existing semantic compression algorithm, this paper is based on the analysis of ItCompress algorithm, and designs a method of bidirectional order selection based on interval partitioning, which named An Optimized Iterative Semantic Compression Algorithm (Optimized ItCompress Algorithm). In order to further improve the speed of the algorithm, we propose a parallel optimization iterative semantic compression algorithm using GPU (**POICAG**) and an optimized iterative semantic compression algorithm using Spark (**DOICAS**). A lot of valid experiments are carried out on four kinds of datasets, which fully verified the efficiency of the proposed algorithm.

Keywords: Semantic Compression, Spark Computing Framework, GPU, Parallel Processing, Interval Partition

1. Introduction

More and more applications need to effectively explore and analyze mass, high-dimensional data table. The traditional syntax-based compression technique treats the data as a continuous byte, does not use semantic information such as the implicit complex dependencies in the data table, and therefore cannot achieve satisfactory compression effect. For exploratory data analysis, there is no need for an accurate answer. As long as the error range can be guaranteed, fast and approximate answers are often more ideal. So people try to apply semantic compression to large data tables. Semantic compression effectively reduce the content redundancy by mining the association contained in the data, and extracting the semantic model, which is applied to the data compression process. Semantic compression is generally lossy compression, namely, allowing that there is a certain error between the compressed data and the original data. There are some typical methods[1,2] for semantic compression of large data tables: ItCompress[3], Fascicles[4], SPARTAN[5-6] and so on. Among them, ItCompress and Fascicles on the data table to take the line to compress, to eliminate the data redundancy between tuples. SPARTAN combines the Bayesian network and CART, and firstly finds the dependencies between the attributes found in the ranks, and then operates row compression for the results of column compression. But there are many shortcomings existing in this implementation, which are mainly referring to efficiency, complexity and compression performance.

Contributions. In this paper, we start with analysis of ItCompress algorithm, and design a bidirectional ordered selection method based on interval partitioning, meanwhile, we propose an optimization method, which aims to solve the problems such as too many iterations, the random selection of represented rows, not adapt to large-scale datasets and so on, and also propose the iterative semantic compression algorithm (the optimized ItCompress algorithm). In order to further improve the speed of the algorithm, we propose a parallel optimization iterative semantic compression algorithm using GPU environment and an optimized iterative semantic compression algorithm using Spark computing framework. The validity of the proposed algorithm is verified by experiments. The specific contributions of this paper are as follows:

(1) In order to deal with the large-scale dataset, we propose a bidirectional order selection method based on interval partitioning to solve the random problem of representative rows selection in the ItCompress algorithm. The algorithm divides the entire large database table into the same intervals with the number of representative rows to be extracted. Then, the intermediate data row is selected as the candidate representative row for each interval. After that, we compute the rows upward and downward both way simultaneously.

(2) On the basis of the proposed selection method, we propose an optimized iterative semantic compression algorithm.

(3) In order to improve the speed of the semantic compression algorithm, we use the CUDA kernel function to implement the parallel computation of the optimized iterative semantic compression algorithm on the GPU, and propose the GPU parallel algorithm POICAG (Parallel Optimized Iterative Compression Algorithm Using GPU).

(4) In order to improve the speed and quality of the semantic compression algorithm, we propose DOICAS (Distributed Optimization Iterative Compression Algorithm Using Spark).

(5) In order to verify the effectiveness of the proposed algorithm, we conduct many experiments. First, using the four datasets, comparisons of the parallel optimization algorithm,

the stand-alone serial algorithm, the parallel ItCompress and parallel optimization algorithms are operated respectively in the GPU environment, and the influence for selecting the various number of representative rows is tested. Second, the acceleration ratio and scalability using Spark are compared.

The remains of this paper are organized as follows. In Section 2 first review the ItCompress algorithm (Iterative Semantic Compression Algorithm). In Section 3 we provide some related work. Section 4 presents Optimized Selection Method of Representative Rows. Section 5 describes the Spark implementation and GPU implementation. Then experiments are conducted in Section 6. Finally, the paper is concluded in Section 7.

2. Overview

In this section, we first review the ItCompress algorithm (Iterative Semantic Compression Algorithm), which was published in our original paper[Anthony K.H.Tung, 3]. This paper will modify, optimize and extend ItCompress algorithm. Then we give a brief overview of the GPU parallel framework and the Spark computing framework in data processing aspects.

2.1 ItCompress Algorithm

Given a table T, which has m attributes X_1, X_2, X_3, ... X_m, and n rows, we use $R[X_i]$ to represent each attribute value of row R. We denote the domain of attribute X_i as $dom(X_i)$. Our aim is to perform a lossy compression on T such that the values reconstructed from the compressed table satisfy certain error tolerances for each column. We denote these error tolerances as a vector $e=[e_1, e_2, e_3, ..., e_m]$ with e_i being the error tolerance for X_i. The value e_i is interpreted differently depending on the type of X_i. We focus on two of the most popular types: An attribute X_i is said to be numeric if the values in $dom(X_i)$ can be ordered while attributes with unordered, discrete domain values are said to be categorical.

A new semantic compression scheme based on the selection of representative rows is proposed. Each tuple in the Table T is assigned to one of the representative rows and its attribute values are defaulted to be the same with the assigned representative rows unless the actual differs from the default value by more than an preset error threshold. In such case, outlying values are specifically stored for the row. In fact, the scheme is similar to clustering, and each representative row considered like a cluster representative. However, there is a significant difference due to the outlying values that are possible. These attributes could have values in a row wildly different from its assigned representative row. As such, by most standard metrics, the distance between a row and its representative could be very large.

Example Given the table T in **Fig. 1(*a*)** which contains 5 attributes and 8 tuples. Let the preset error threshold for the numeric attributes age, salary and assets be 5, 25000 and 50000 respectively, while no errors are allowed for categorical data. **Fig. 1(*c*)** is shown the selected representative rows and **Fig. 1(*b*)** is shown the compressed table T_c. As can be seen, each row in the compressed table is associated with one of the representative rows using a representative row ID, namely RRid. A bitmap is assigned to each row to provide the position for the outlying values. A "1" at the n^{th} bit indicates that its n^{th} attribute value is within an preset error tolerance threshold of the n^{th} attribute value for the representative row, while a "0" indicates otherwise. In representative rows, each value is selected from responding collection of values for the same attribute in table T, namely $P_i \in (dom(X_1)) \times (dom(X_2)) \times \ldots (dom(X_m))$.

The difficult issue is how to select the representative rows in order to compress the data. For the purpose, the ItCompress algorithm is developed, we describe it in more detail.

age	salary	assets	credit	sex
20	30000	25000	poor	male
25	76000	75000	good	female
30	90000	200000	good	female
40	100000	175000	poor	male
50	110000	250000	good	female
60	50000	150000	good	male
70	35000	125000	poor	female
75	15000	100000	poor	male

(a) Table *T*

RRid	Bitmap	Outlying Values
2	01011	2,025,000
1	11011	75000
1	11111	
1	01100	40,poor,male
1	01111	50
1	01110	60,male
2	11110	female
2	11111	

(b)Table T_c

RRid	age	salary	assets	credit	sex
1	30	90000	200000	good	female
2	70	35000	100000	poor	male

(c)Representative Rows *P*

Fig. 1. Representative Rows and Compressed Table

Algorithm 1 ItCompress

Input：Table *T*, the number of representative rows *k*, error tolerance *e*

Output：Representative rows *P*, a compressed table T_c

1: Select *k* rows randomly from *T* to construct *P*
2: while
3: Initialize array *G*, number *k*, index from 1。
4: for *i*=1 to *T*.size *i*++
5: Compare R_i and *P*, find the biggest coverage value P_j。
6: *G*[*j*]=*i*
7: end for
8: for *i*=1 to *k*
9: Sequential readout responding row in *G*[*i*]
10: Count the number of occurrences of each property value
11: $P_i[X_j]=fv(X_j, G(P_i))$
12: end for
13: Compute *totalcov*, judge whether the loop is over
14: end while

In Algorithm 1, ItCompress algorithm begins by picking a random set of k representative rows from the table T. It then iteratively improves this random selection with the objective of increasing the total coverage over the table to be compressed. There are two phases in each iteration, which are step 4 to 7 of ItCompress and Step 8 to 12 of ItCompress respectively.

Definition 1 Coverage

Let *R* be a row in *T* and let P_i be a representative row in *P*. We say that the coverage of P_i on *R*, $cov(P_i, R)$ is the number of attributes X_i in which $R[X_i]$ is matched by $P[X_i]$.

Definition 2 Total Coverage

For each row R_i in table *T*, let $P_{max}(R_i)$ be the representative row from *P* that gives the maximum coverage among P_i to R_i. So the total coverage of *P* on *T* to be $totalcov(P, T)=\sum_{i=1,2,3,...n} \mathrm{cov}(P_{\max}(R_i), R_i)$.

Definition 3 Maximum Coverage Problem

Find a set of *k* representative rows *P* which maximizes $totalcov(P, T)$.

2.1 Spark Computing Framework

Spark is a distributed cluster computing framework based on memory computing. Compared to MapReduce, it has many advantages: the improved efficiency of lower network transmission and disk I/O. Spark uses memory for data computation for quick processing of queries, real-time return to analysis results. Spark provides higher-level API than Hadoop, and the running speed of same algorithm using Spark is 10-100 times faster than that using Hadoop. Spark is designed for specific types of workloads in cluster computing, that is, workloads of work datasets is reused (such as machine learning tasks) between parallel operations. Spark's computing architecture has three features:

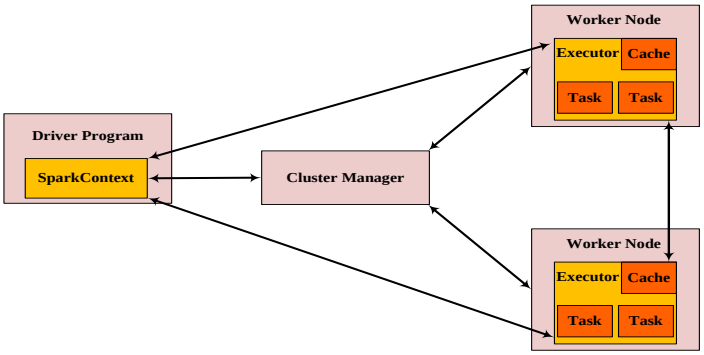

Fig. 2. Spark Operation Mode

(1) Spark has a lightweight cluster computing framework. It applies Scala to its program architecture, and Scala as a multi-paradigm programming language has the features of concurrency, scalability, and support for programming paradigms, which is tightly integrated with Spark to easily manipulate distributed datasets and can easily add a new language structure.

(2) Spark contains data flow computation and interactive computing in large data area. Spark can interact with HDFS to get the inside data files, while Spark can provide a good framework for data mining and machine learning owing to iterations and memory computation, as well as interactive computation.

(3) Spark has a good fault tolerance mechanism. Spark uses RDD, which is represented as a collection of objects that are serialized, read-only, fault-tolerant, and can be executed in parallel by a Scala object in a set of nodes. Spark has made the most active and efficient large data common computing platform because of the characteristic that can efficiently process the distributed datasets, **Fig. 2** describes the Spark's mode of operation.

2.3 GPU Parallel Computing

GPU is designed for graphics processing. Unlike the CPU's serial design pattern, GPU not only has natural parallel characteristic, but also has more obvious advantages on floating-point processing power and memory bandwidth than that of CPU, which contributed majorly to GPU’s strong computing power. Due to the high degree of parallelism in graphics processing, GPU can improve the processing power and memory bandwidth by increasing the parallel processing unit and the memory control unit.

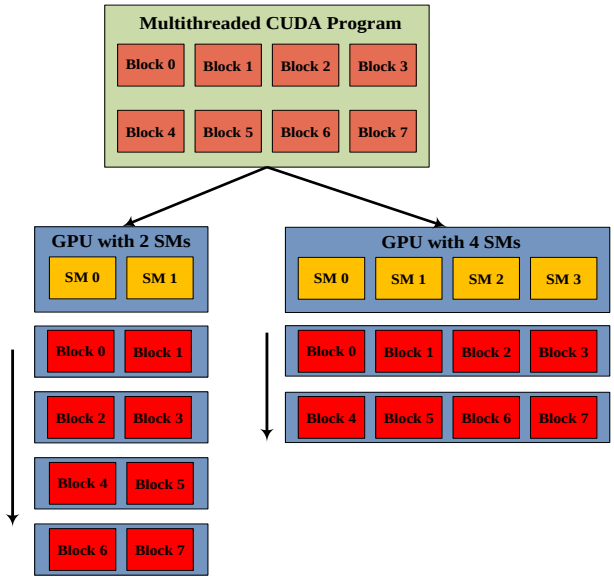

Fig. 3. GPU Parallel Computing Architecture

3. Related Work

Although compression of datasets is a classical research topic in the database research community, the idea of exploiting attribute correlations (a.k.a. semantic compression) is relatively new [3-16]. Many excellent algorithms have been proposed up to now. Jagadish [4] introduce the notion of fascicles, study two problems related to fascicles, and develop algorithms to attack both of the above problems. Moreover, a semantic compression algorithm called ItCompress Iterative Compression is proposed in [3], which achieves good compression while permitting access even at attribute level without requiring the decompression of a larger unit. However, the number of iterative is huge in the case of big data. SPARTAN [5] is a system that takes advantage of attribute semantics and data-mining models to perform lossy compression of massive data tables, which is based on the novel idea of exploiting predictive data correlations and prescribed error tolerances for individual attributes to construct concise and accurate Classification and Regression Tree (CaRT) models for entire columns of a table. Wei [6] propose an effective compression approach based on on-line semantic clustering of GML documents. The approach deals with a GML document under compression on the fly via separating data from structures, clustering data based on the semantic similarities exploited from tags and texts, dictionary-encoding structures and delta-encoding geometric coordinate data before the general text compression on back end. David [7] outlines how Lossy Compression, a branch of Information Theory relating to the compact representation of data while retaining important information, can be applied to the Worst Case Execution Time analysis problem. In [8], a compression algorithm(2P2D), which exploits the obtained group movement patterns to reduce the amount of delivered data. The compression algorithm includes a sequence merge and an entropy reduction phases. However, the method is not really distributed. Wang [9] proposed several search space pruning methods and designed an efficient algorithm called SUMMARY. However, their work only supports categorical attributes. Other effective algorithms are proposed [10-21], but all have the shortcoming that can not handle a large scale data effectively.

Huffman Coding is used to compress attributes [22-24]. The major contribution of Raman's work [23] is that they formalized the old idea of compressing ordered sequences by storing the difference between adjacent elements, which has been used in search engines to compress inverted indexes [22, 28]. Stonebraker [24] presents the design of a read-optimized relational DBMS that contrasts sharply with most current systems, which are write-optimized.

Bayesian networks are well-known general purpose probabilistic models to characterize dependencies between random variables [25-31]. SPARTAN [26] uses a Bayesian network structure to guide the selection of CaRT models that minimize the overall storage requirement, based on the prediction and materialization costs for each attribute. Algorithms for automatically learn Bayesian networks and new structures called "Huffman networks" that model statistical relationships in the datasets are proposed, and algorithms for using these models to then compress the datasets [25]. SQUISH [28] is a system that uses a combination of Bayesian Networks and Arithmetic Coding to capture multiple kinds of dependencies among attributes and achieve near-entropy compression rate.

In recent year, a few scholars have begun to study parallel compression algorithms to speedup the running the operation [33-36]. Cheng [33] proposes an efficient algorithm for fast encoding large Semantic Web data, and present the detailed implementation of proposed approach based on the state-of-art asynchronous partitioned global address space (APGAS) parallel programming model. Urbani [34] researches and proposes a MapReduce algorithm that efficiently compresses and decompresses a large amount of Semantic Web data. Afterwards proposes a set of distributed MapReduce algorithms to efficiently compress and decompress a large amount of RDF data [35]. However, the research in parallel computing is still rare. Moreover, all of research stay still in Hadoop1.0 phase. In view of this, we will propose a parallel computing algorithm based on GPU and a distributed algorithm based on Spark through analysing and optimizing ItCompress algorithm.

4. Selection Method of Representative Rows

The key point of the ItCompress algorithm lies in the selection of the representative rows, and the selection of representative rows is obtained by improving step by step through iterations. The more each attribute value of selected representative row matches the corresponding attribute value in the corresponding table T (i.e., the error threshold is not exceeded), the greater the number of bits "1", the greater the coverage value, and the better the quality of the representative row. A large number of experiments have shown that, this selection method can efficiently extract the best representative rows, whose essence is exhaustive comparison. However, in the case of massive data, the method is greatly challenged. In order to obtain the maximum *coverage* value and the *totalcov* value, it is necessary to compare each row of the entire table T, which will inevitably cause the rapid expansion of the workload. In order to deal with large-scale datasets, we propose a bidirectional ordered selection method based on interval partitioning. The main idea of the method is as follows:

(1) First of all, the value of the numeric attribute of table T is pre-computed and sorted. Take table T in **Fig. 1** as an example to sort table T and get **Fig. 4**.

(2) In order to reduce the uncertainty and efficiency brought by randomly selected attribute values, we divide the entire table T according to the number of selected representative rows. For example, if the number of rows of table T is n, the whole algorithm needs to select d representative rows, then we divided the table T into d intervals, the number of data line in each section is $\lfloor n/d \rfloor$.

age	salary	assets	credit	sex
20	15000	25000	poor	male
25	30000	75000	good	female
30	35000	100000	good	female
40	50000	125000	poor	male
50	76000	150000	good	female
60	90000	175000	good	male
70	100000	200000	poor	female
75	110000	250000	poor	male

Fig. 4. Table T_o After Sorting

(3) The value of the intermediate point element is selected as the initial value of the first iteration from each numeric type attribute of each partition, and the *totalcov* between them and the data row in table T is computed, as shown in **Fig. 5** (25, 30000, 75000, good, female) and (60, 90000, 175000, good, male), the first and last rows of data in each interval are labeled. The second iteration starts from the intermediate data row and then goes upward and downward both ways simultaneously to compare. Once there is a row, whose numeric type data value exceeds the error threshold, the iteration is ended, and the rest of the data lines are all discarded.

(4) Through this optimization iterative way, a large number of data rows that are not satisfied is cleaned up in advance, and efficiency is greatly improved by avoiding unnecessary calculation of consumption. This method is similar to pruning.

age	salary	assets	credit	sex
20	15000	25000	poor	male
25	**30000**	**75000**	**good**	**female**
30	35000	100000	good	female
40	50000	125000	poor	male
50	76000	150000	good	female
60	**90000**	**175000**	**good**	**male**
70	100000	200000	poor	female
75	110000	250000	poor	male

Fig. 5. The Intermediate Data Rows Selected In the First Iteration

5. Parallel Processing of Optimization Iterative Semantic Compression Algorithm

5.1 Distributed Implementation in Spark Environment

Distributed Optimization Iteration Compression Algorithm using Spark (referred to as **DOICAS**) is designed by iterative RDD. DOICAS algorithm in the cluster achieve distributed operation by the Drive/Executor process. The pre-processed dataset is stored in HDFS, then the data is converted into Spark RDD and stored in the memory of the cluster. The data loading process is described in **Fig. 6**. The dataset is stored in the HDFS system in the form of Block, and the SparkContext object converts the data into the RDD stored in the form of Partition through the textFile method and loads it into memory.

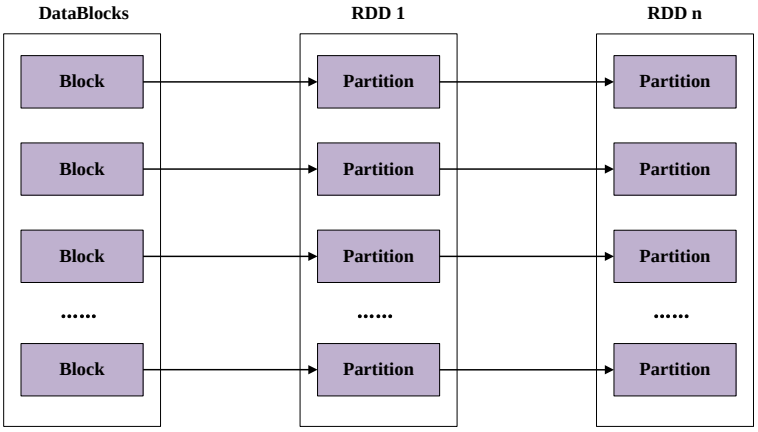

Fig. 6. The Phase of Loading Data

Algorithm 2 DOICAS

Input: A table T, An ordered table T_o, a user specified value k, an error tolerance vector e.
Output: A compressed table T_c, a set of representative rows P
1:**Driver:**
2: Load the dataset T_o from HDFS to convert to RDD.
3:**Executor:**
4: The dataset RDD is divided into k intervals, perform a map operation for each interval.
5:**Loop:**
6: For each section, take the ith row, which $i=\lfloor n/2k \rfloor$
7: Compare the data row to each row in table T, and calculate its coverage value
8: i++ or i--
9: Take the i++ and i-- data lines, compared with the table T of each row, which coverage value calculated
10: If the numeric property value has a value greater than error tolerance, it ends. So each interval acquire a representative row
11:**End Loop**

As shown in Algorithm 2, on each work node, the dataset is extracted as a row unit and converted to a row wrapper class instance so that the dataset RDD is converted into a new RDD stored in memory. The task of the Drive process is to read the dataset and convert it. The Executor process performs a comparison, computes a coverage, and so on to generate the final compressed data and representative rows.

5.2 Parallel Implementation in GPU Environment

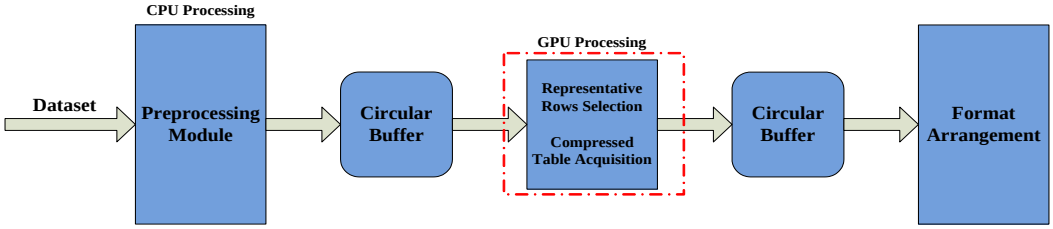

Fig. 7. Optimized Iterative Semantic Compression Algorithm Based on GPU

GPU parallel computing principle is a SIMT (Single Instruction Multiple Thread) mode, that is, a bunch of threads to execute the same instruction, but the processing data can be different.

Therefore, in order to give full play to the GPU parallel computing power, the key is a reasonable allocation of each thread to deal with the data, and strive to deal with the amount of data between the threads more balanced. Because CUDA provides a parallel programming model based on thread grid, CUDA's kernel function is used to realize the iterative semantic compression algorithm using GPU. The important task is to divide the thread grid.

Based on the idea of bidirectional ordered selection based on interval partitioning, we present the Parallel Optimization Iterative Compression Algorithm Using GPU (referred to as **POICAG**), as shown in Algorithm 3. Assuming that the number of stream processors of the GPU is *Pnum*, the number of dataset rows is *n*, grouped according to Equation (1), the number of rows in each group is

$$Row=n/Pnum \tag{1}$$

Algorithm 3 POICAG

Input: *inputPath*[100], *paraPath*[100], *numDim*, *numSample*, *k*, *maxIter*
Output: *k* representative rows
Procedure:

```
1:  Threads=Pnum/Blocks                  // Number of threads per block
2:  Threadi=blockIdx.x*blockDim.x+threadIdx.x  // The global number of the thread
3:  start_row=Threadi*Row                // Label the first line number of each group
4:  end_row=Threadi*Row+Row              // Label the last line number of each group
5:  mid_row=(start_row+end_row)/2        // Obtain the middle line number for each group
6:  for mid_row to start_row             // Compares each upward row in turn
7:  for mid_row to end_row               // Compares each downward row in turn
8: //initialization
9:    srand(100);
10:   cudaSetDevice(0); int i, j, nearest, *member, *tolerance,* aspace, *numValues;
11: //input parameter declaration
12:   int numDim, numSample, k, maxIter; char inputPath[100], paraPath[100];
13: //check and read input parameters
14:   strcpy(inputPath, argv[1]);          // input file path
15:   strcpy(paraPath, argv[2]);           // parameter file path
16:   sscanf(argv[3], "%d", &numDim);      // number of dimension
17:  sscanf(argv[4], "%d", &numSample); // number of samples
18:  sscanf(argv[5], "%d", &k);           // number of centers
19:  sscanf(argv[6], "%d", &maxIter);     // maximum iteration
20://initialize member data
21:  cudaMallocManaged(&member, sizeof(int) * numSample);
22:  cudaMallocManaged(&numMatch, sizeof(int) * numSample);
23:  cudaMemset(member, 1, sizeof(int) * numSample);
24://initialize input data
25:  cudaMallocManaged(&input, sizeof(int) * numDim * numSample);
26:  readRow(inputFile, input, numDim * numSample);
27://initialize parameter data
28://read number of different value in each attribute (1st row)
29:  cudaMallocManaged(&numValues, sizeof(int) * numDim);
30:  readRow(paraFile, numValues, numDim);
31://read range of tolerance for each attribute (2nd row)
32:  cudaMallocManaged(&tolerance, sizeof(int) * numDim);
33:  readRow(paraFile, tolerance, numDim);
34:// read storage requirement of each attribute (3rd row)
35:  cudaMallocManaged(&aspace, sizeof(int) * numDim);
36:  readRow(paraFile, aspace, numDim);
```

```
37:/* initialize center and cluster data */
38:  cudaMallocManaged(&center, sizeof(int) * k * numDim);
39:  cudaMallocManaged(&ccenter, sizeof(int) * k);
40:  for (i = 0; i < k; ++i) do
41:       ccenter[i] = numDim;
42://pick k centers
43:  for (i = 0; i < k; ++i) do
44:              dim3 blockSize(BLOCK_SIZE, BLOCK_SIZE, 1);
45:              dim3 gridSize(ceil(numDim, BLOCK_SIZE), ceil(k, BLOCK_SIZE), 1);
46:// compute initial centers
47: computeCenter<<<gridSize, blockSize>>>(clusterInfoDim,
48:   numValues, tolerance, columnMatch, center, numDim, k);
49://iteration
50:  for (int it = 0; it < maxIter; ++it) do
51:          {//move this loop to a separate function//// GPU call
52:    for (i = 0; i < k; ++i) do
53:            {
54:                    for (j = 0; j < numDim; ++) do
55:                            printf("%d ", center[i * numDim + j]);
56:            }
57:   }
```

POICAG is shown as Algorithm 3, in which, part of Grouping Computing is described from step 1 to 7, and the rest is the main part of POICAG.

6.Performance Evaluation

Aimed to the distributed algorithm using Spark DOICAS, according to the literature [Ran Jin, 2], we build a cloud platform with 8 computers, installed JDK1.8.0 and Spark-1.5.2, the specific configuration information is as follows:

Table 1. Cluster Configuration

Item	Personal Computer	Server
Memory(GB)	8	128
Hard Disk(GB)	512	1000
Processor(GHz)	Intel Core(TM) i7-5500U 2.4GHz	Intel Xeon 2.0
Core Number	4	24
OS	CenOS6.6	CenOS6.6

We implement the GPU-based parallel algorithm POICAG with CUDA C. The client computers are equipped with Intel Core (TM) i7-5500U 2.4GHz processor, and 8GB memory, 64-bit OS. GPU servers use the high performance GPU computing cluster by the school of computing of National University of Singapore(NUS). The Configuration information is listed in **Table 1.**

The experimental datasets are shown as following:

We implement the GPU-based parallel algorithm POICAG with CUDA C. The client computers are equipped with Intel Core (TM) i7-5500U 2.4GHz processor, and 8GB memory, 64-bit OS. GPU servers use the high performance GPU computing cluster by the school of computing of National University of Singapore(NUS).

(1)NHL Dataset

(2)Sponge Dataset [Vibhav Vineet, 3]

(3)RoadSafe Dataset(https://data.gov.uk/data/)

(4)KDD Cup1999 Dataset (http://archive.ics.uci.edu/ml/datasets/)

For the sake of experiment convenience, we clean the datasets and delete the insignificant attributes, the final datasets information is shown in **Table 2**. Among them, the NHL dataset is our artificial simulation dataset, Road Safe Dataset is the safety data about the British road accident and personal casualties. In order to avoid chance, we run ten times for each dataset, and remove the maximum and minimum, and finally take the average. For example, NHL dataset in the CPU stand-alone mode running 10 times the results shown in **Fig. 8**, removing the maximum value 0.051 and the minimum value 0.021.

Table 2. Dataset Information

Dataset	Tuples	Attributes
NHL	856	13
Sponge	2400	10
Road Safe	141663	17
KDD Cup1999	4000000	42

0.049	0.049	0.049	0.046	0.045	**0.021**	0.047	0.051	0.051	**0.051**

Fig. 8. Results of The Ten Runs of The NHL Dataset in The CPU Stand-alone Mode

6.1 Comparison in GPU Mode

(1)Running Time

In order to verify the effectiveness of POICAG (Parallel Optimized Iterative Compression Algorithm Using GPU) proposed in this paper, we compare it with that in CPU stand-alone mode. For different numbers of datasets, we set different *k* values (the number of selected rows).

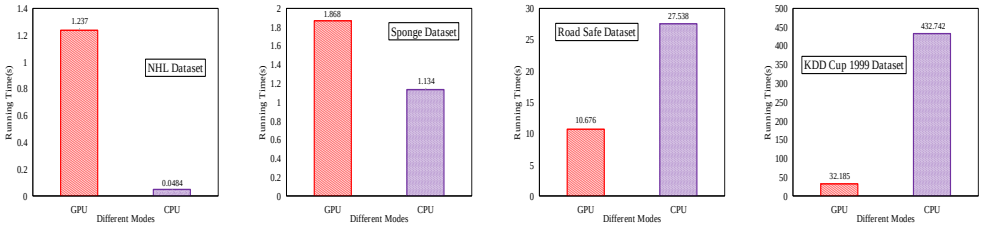

Fig. 9. Four Datasets in the CPU and GPU Mode Running Time

We can be seen from **Fig. 9**, as the amount of data continues to grow, the advantages of GPU parallel computing highlights, and execution time surge in CPU stand-alone mode. It is conceivable that CPU mode running will encounter a huge bottleneck problem when the amount of data increased to T, P level. As the NHL Dataset data volume is small, CPU mode operation has certain advantages. Whereas GPU mode running time is 25.558 times that of the CPU mode, when processing Sponge Dataset, the ratio is reduced to 1.647 times. When dealing with the Road Safe Dataset, the GPU only needs 10.676 seconds and the time is nearly

1/3 of the CPU mode. When dealing with KDD Cup1999 Dataset, GPU advantage is more obvious, running only about 1/13 of the CPU. So GPU is an efficient tool to handle large-scale data.

Table 3. Comparison of run time between ItCompress and Optimized ItCompress

	NHL Dataset	Sponge Dataset	Road Safe Dataset	KDD Cup1999
ItCompress Using GPU	1.398(s)	2.118(s)	13.741(s)	36.813(s)
POICAG	1.237(s)	1.868(s)	10.676(s)	32.185(s)

In order to verify the effectiveness of **POICAG** (Parallel Optimized Iterative Compression Algorithm Using GPU) proposed in this paper, we use the above four datasets to compare the ItCompress algorithm using GPU with **POICAG**. The results are shown in **Table 3**.

(2) Compress Ratio under Different *k*

Table 4. Different *k* Values Influence On the Dataset Compression Ratio (一)

	NHL Dataset			Sponge Dataset		
	***k*=50**	***k*=300**	***k*=500**	***k*=50**	***k*=300**	***k*=500**
ItCompress	0.313	0.202	0.196	0.332	0.231	0.225
	0.293	0.209	0.201	0.312	0.235	0.229
	0.289	0.211	0.205	0.310	0.242	0.231
	0.327	0.199	0.187	0.342	0.219	0.213
	0.318	0.202	0.198	0.332	0.235	0.223
	0.302	0.205	0.201	0.319	0.230	0.230
	0.315	0.210	0.202	0.330	0.239	0.227
	0.299	0.206	0.201	0.311	0.234	0.235
	0.314	0.204	0.199	0.332	0.228	0.227
	0.302	0.205	0.200	0.318	0.232	0.232
Compress Ratio (Optimized ItCompress)	0.334	0.214	0.208	0.354	0.246	0.237
	0.325	0.220	0.213	0.342	0.252	0.252
	0.312	0.217	0.210	0.335	0.239	0.243
	0.330	0.207	0.212	0.349	0.232	0.234
	0.327	0.209	0.199	0.342	0.235	0.237
	0.318	0.212	0.203	0.332	0.239	0.240
	0.329	0.216	0.208	0.344	0.243	0.237
	0.331	0.207	0.200	0.349	0.231	0.229
	0.329	0.213	0.202	0.343	0.242	0.232
	0.322	0.215	0.203	0.341	0.244	0.236
Compress Ratio (POICAG)	0.320	0.210	0.201	0.338	0.235	0.238
	0.319	0.223	0.209	0.335	0.251	0.231
	0.327	0.209	0.199	0.346	0.241	0.230

	0.330	0.211	0.206	0.352	0.243	0.233
	0.323	0.213	0.202	0.339	0.239	0.234
	0.327	0.210	0.200	0.341	0.237	0.228
	0.327	0.213	0.203	0.343	0.241	0.231
	0.329	0.215	0.207	0.344	0.245	0.235
	0.331	0.217	0.205	0.353	0.244	0.233
	0.328	0.212	0.203	0.340	0.240	0.230

Table 5. Different *k* Values Influence On the Dataset Compression Ratio (二)

	Road Safe Dataset			**KDD Cup1999**		
	***k*=50**	***k*=300**	***k*=500**	***k*=50**	***k*=300**	***k*=500**
ItCompress	0.503	0.412	0.401	0.698	0.623	0.612
	0.506	0.405	0.393	0.712	0.628	0.615
	0.511	0.409	0.390	0.703	0.641	0.620
	0.513	0.415	0.402	0.692	0.621	0.616
	0.497	0.401	0.388	0.694	0.619	0.609
	0.502	0.412	0.400	0.696	0.627	0.617
	0.507	0.415	0.403	0.702	0.631	0.621
	0.511	0.406	0.393	0.711	0.634	0.613
	0.494	0.401	0.391	0.708	0.629	0.619
	0.500	0.411	0.392	0.699	0.633	0.623
Compress Ratio (Optimized ItCompress)	0.531	0.423	0.414	0.721	0.629	0.618
	0.527	0.414	0.401	0.730	0.638	0.627
	0.529	0.411	0.400	0.718	0.631	0.630
	0.532	0.420	0.406	0.719	0.627	0.624
	0.535	0.422	0.414	0.723	0.620	0.611
	0.526	0.414	0.403	0.730	0.632	0.618
	0.530	0.421	0.412	0.722	0.628	0.616
	0.528	0.412	0.401	0.710	0.632	0.622
	0.531	0.417	0.403	0.711	0.629	0.614
	0.528	0.414	0.402	0.714	0.630	0.619
Compress Ratio (POICAG)	0.537	0.423	0.411	0.734	0.626	0.621
	0.531	0.421	0.410	0.727	0.641	0.623
	0.533	0.424	0.413	0.733	0.637	0.619
	0.539	0.423	0.412	0.725	0.632	0.621
	0.541	0.430	0.418	0.723	0.630	0.623
	0.543	0.429	0.415	0.719	0.633	0.620
	0.538	0.415	0.402	0.731	0.634	0.614
	0.544	0.433	0.421	0.726	0.641	0.622
	0.540	0.428	0.415	0.728	0.637	0.615
	0.537	0.424	0.410	0.729	0.642	0.621

The *k* representative rows is preselected randomly for ItCompress and POICAG (Parallel Optimized Iterative Compression Algorithm Using GPU). In order to examine whether the difference in *k* value has an effect on compression performance when more or fewer representative rows are selected, we varied *k*, and again repeated ten times with different random initial representative rows for each *k* value. For the same reason, we also repeated the experiment with four datasets. The experimental results are shown in **Table 4** and **Table 5** where each column represents one set of ten repetitions for a selected dataset and specified value of *k*. As can be seen, all values in any column are almost identical, indicating that the variance in compression ration is insignificant for all the four datasets with the value of *k* ranging from 50 to 500 due to different random initialization. Therefor, we believe that both the compression ration of POICAG and Optimized ItCompress are stable although the random initialization. Meanwhile, the two improved algorithms maintain a good compression rate as seen from the experimental results. In contrast, the POICAG algorithm has better stability, the value changes even smaller.

6.2 Comparison in Spark Mode

(1)Test of Speedup Ratio

Speedup ratio is defined by parallel computing to reduce the running time and improve the performance of obtained. It is an important indicator to verify the performance of parallel computing. The greater speedup ratio is, it's indicating that the less time parallel computing consume relatively, and the higher parallel efficiency and performance improve. Under changing the number of Hadoop cluster nodes, respectively use the results of speedup ratio performance tests according to above four datasets. **Table 6** is the running time of datasets under different nodes. **Table 6** and **Fig. 10** show the results.

Table 6. Comparison of running time and speedup

Dataset	Nodes	Total Time(sec)	Speedup
NHL Dataset	1	0.0484	1
	2	1.1293	0.043
	4	0.9319	0.052
	6	0.8726	0.055
	8	0.6942	0.070
Sponge Dataset	1	1.134	1
	2	1.792	0.633
	4	1.325	0.856
	6	1.185	0.957
	8	0.976	1.162
Road Safe Dataset	1	27.538	1
	2	15.772	1.746
	4	9.261	2.974
	6	6.271	4.391
	8	4.809	5.726
KDD Cup 1999	1	432.742	1
	2	38.026	11.380
	4	21.854	19.802
	6	8.629	50.150
	8	6.812	63.526

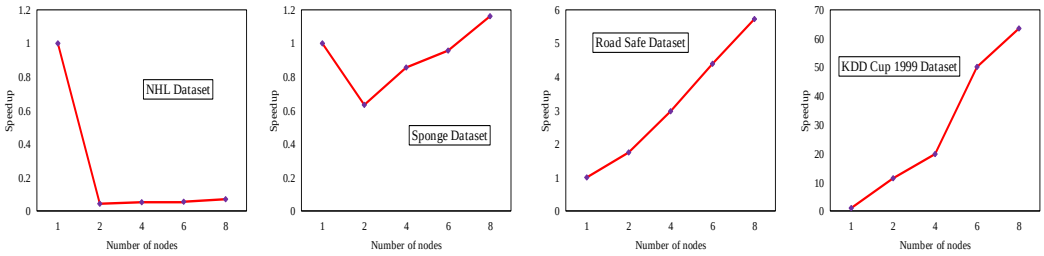

Fig. 10. Speedup Ratio Performance Test

From the experimental results as shown in **Table 6**, we can see that when the Spark distributed platform has only one computer, it is actually degraded into stand-alone CPU mode. With the increase of the number of computers in the platform, the running speed of the dataset is also increase. However, when dealing with the smaller amount of data NHL dataset and Sponge dataset, the advantage of Spark platform is not reflected, basically much slower than the stand-alone mode. For example, in the NHL dataset, the running time of two nodes is 1.1293, which is equivalent to that of GPU parallel mode, yet is nearly 25 times slower than the stand-alone mode. The reason is that Spark distributed platform is a large frame structure that needs to support multiple modules, and there is no advantage in dealing with the small amount of data. But with the increase in the number of nodes, running time gradually reduced, the acceleration ratio gradually increased, but not obvious. In dealing with large-scale dataset KDD Cup1999, CPU stand-alone mode will encounter a small bottleneck, running time consumption is very large, while the Spark cloud platform shows its value, and the running time on which with two nodes is about one eleventh that of stand-alone mode.

(2)Analysis of scalability

According to the paper [Ran Jin, 37], the formula is $\eta=S_p/N$， wherein, S_p represents the speedup ratio, N means the number of cluster nodes. **Fig. 11** shows the efficiency of parallel algorithms proposed in the paper.

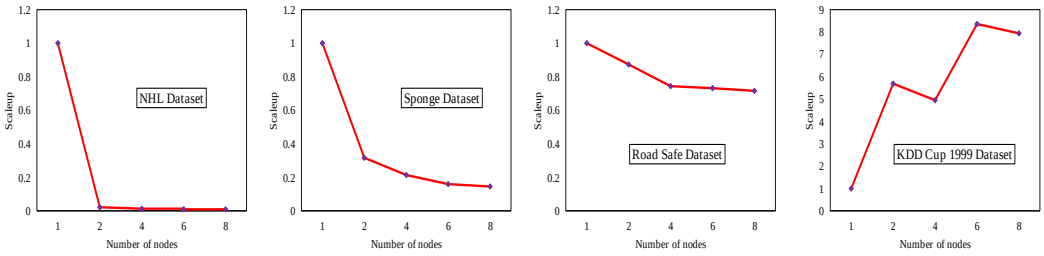

Fig. 11. Expansion Rate Performance Test

Fig. 11 depicts the experimental results on the four datasets. We can find from it: (1) Under the large scale dataset, with the cloud platform work node increases, running time is gradually reduced; (2) In addition to the small dataset NHL, although the performance of different dataset is various, the parallel algorithm using Spark show a basic linear rule between the work nodes and the speedup ratio.

6.3 Effectiveness Comparison of Algorithms

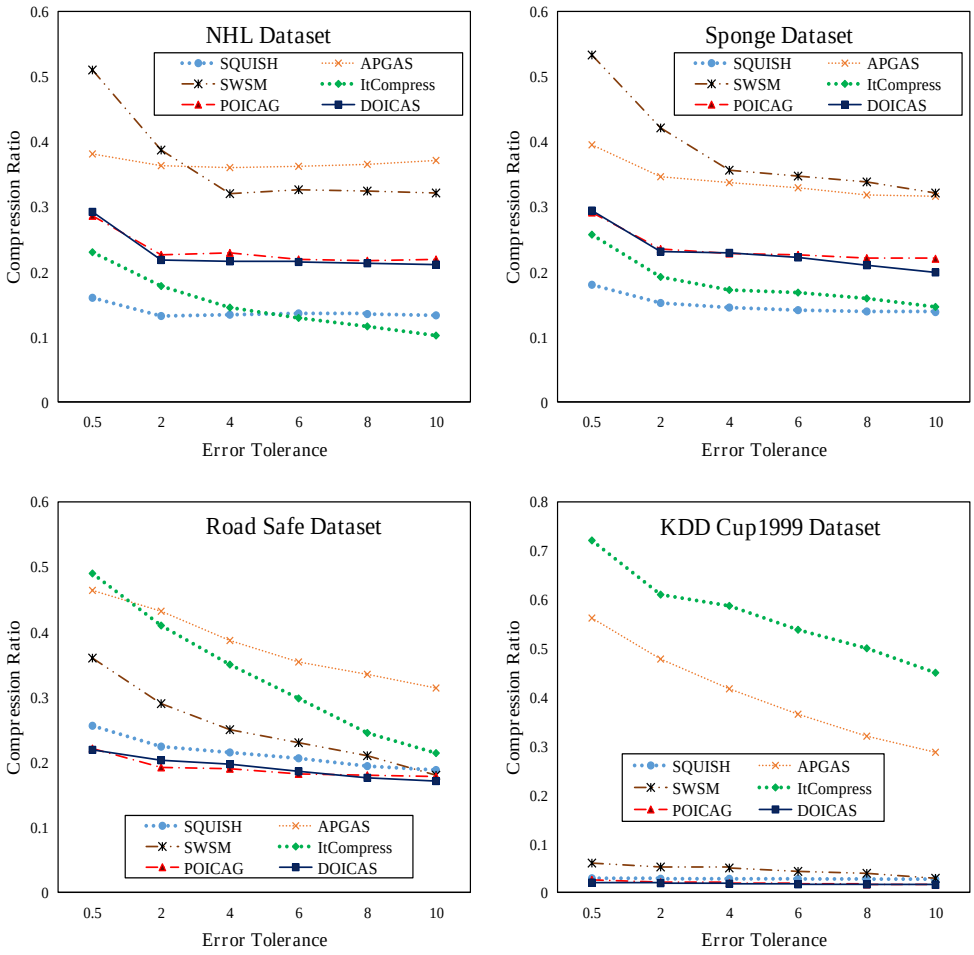

Fig. 12. Error Threshold vs Compression Ratio

Here, We compare POICAG and DOICAS with the other compression algorithms according to effectiveness. SQUISH[28], APGAS[33] and SWSM[34] are proposed in recent years, which have been proved to be effective.

From the graphs in **Fig. 12**, we can make the following observations: (1) SQUISH algorithm acquires better performance than others on the NHL dataset and Sponge dataset, especially when error tolerance threshold is small(0.5%). The main reason is that the encoding scheme adopted by SQUISH can leverage the skewness of the distribution and achieve near-optimal performance. (2) For pure semantic compression, the compressed tables produced by ItCompress achieves about 50% reduction in compression ratio in NHL dataset and Sponge dataset, compared to SWSM. But ItCompress is worse when the number of dataset is large. (3) In Road Safe dataset and KDD Cup1999 dataset, the performance of POICAG and DOICAS is far better than ItCompress, and also than other algorithms. A bidirectional ordered selection method based on interval partitioning is a main reason. In contrast, DOICAS is slightly better than POICAG

7.Conclusion

Relational datasets are being generated at an alarmingly rapid rate across organizations and industries. Compressing these datasets could significantly reduce storage and archival costs. Traditional compression algorithms, e.g., gzip, are suboptimal for compressing relational datasets since they ignore the table structure and relationships between attributes. The existing semantic compression algorithms face many challenges, such as too many iterations, the random selection of representative rows, not adapt to large scale datasets and so on. In this paper, we firstly analyze the ItCompress algorithm, and design a bidirectional ordered selection method based on interval partitioning, then propose an Optimized Iterative Semantic Compression Algorithm. On the basis of this, we further propose a Parallel Optimized Iterative Compression Algorithm Using GPU(POICAG) and Distributed Optimization Iterative Compression Algorithm Using Spark(DOICAS), using GPU environment and Spark computing framework respectively. A lot of valid experiments are carried out on four kinds of datasets. The efficiency of the proposed algorithm is verified by the comparison of speedup, scalability scalability, running time and so on.

ACKNOWLEDGMENT

This work was supported by the Postdoctoral Research Project of Zhejiang Province, and by the National Natural Science Foundation of China under grant no. 61472348 and 61672455, and by the Humanities and Social Science Fund of the Ministry of Education of China under Grant No. 17YJCZH076, and by Zhejiang Science and Technology Project under Grant No. LGF18F020001, and by the Ningbo Natural Science Foundation under Grant No. 2017A610111.

References

[1] Promhouse G and Bennett M., "Semantic Data Compression," in *Proc. of Data Compression Conference*, pp. 323-331, April 8-11, 1991. Article (CrossRef Link).

[2] Schmalz Mark S., "An overview of semantic compression,"in *Proc. of SPIE*, pp. 1493-1495, August 20, 2010. Article (CrossRef Link).

[3] Jagadish H V, Ng R T, Ooi B C and Anthony K H Tung, "ItCompress: An Iterative Semantic Compression Algorithm," in *Proc. of 20th International Conference on Data Engineering(ICDE'04)*, pp. 646-657, March 5, 2004. Article (CrossRef Link).

[4] Jagadish H V, Madar J, Ng R, "Semantic Compression and Pattern Extraction with Fascicles," in *Proc. 1999 International Conference Very Large Data Bases(VLDB'99)*, pp. 186-197, September 7-10, 1999. Article (CrossRef Link).

[5] Babu S, Garofalakis M, Rastogi R, "SPARTAN: A Model-based Semantic Compression System for Massive Data Tables," in *Proc. of ACM SIGMOD'2001 International Conference on Management of Data*, pp. 22-49, May 21-24, 2001. Article (CrossRef Link).

[6] Wei Qingting, Guan Jihong, "A GML Compression Approach Based on On-line Semantic Clustering," in *Proc. of the 18th International Conference on Geoinformatics*, pp. 1-7, June 18-20, 2010. Article (CrossRef Link).

[7] Griffin David, Lesage Benjamin, Burns Alan and RI Davis, "Lossy Compression for Worst-Case Execution Time Analysis of PLRU Caches," in *Proc. of the 22nd International Conference on Real-time Networks and Systems*, pp. 203-212, October 8-10, 2014. Article (CrossRef Link).

[8] Hsiao-Ping Tsai, De-Nian Yang and Ming-Syan Chen, "Exploring Application-Level Semantics for Data Compression," *IEEE Transactions on Knowledge and Data Engineering*, vol. 23, no.1, pp. 95-109, February, 2011. Article (CrossRef Link).
[9] J. Wang and G. Karypis, "On Efficiently Summarizing Categorical Databases," *Knowledge and Information Systems*, vol. 9, no. 1, pp. 19-37, January, 2006. Article (CrossRef Link).
[10] R. Saint-Paul, G. Raschia and N. Mouaddib, "General Purpose Database Summarization," in *Proc. of the 31st International Conference on Very Large Databases (VLDB 2005)*, pp. 733-744, August 30- September 2, 2005. Article (CrossRef Link).
[11] Pham Quang-Khai, Raschia Guillaume and Mouaddib Noureddine, "Time Sequence Summarization to Scale up Chronology-dependent Applications," in *Proc. of the 18th ACM Conference on Information and Knowledge Management*, pp. 1137-1146, November 2-6, 2009. Article (CrossRef Link).
[12] Li Liu, Lifang Wang and Chin-Chen Chang, "A Semantic Compression Scheme for Digital Images Based on Vector Quantization and Data Hiding," *Multimedia Tools and Applications*, pp. 1-14, 2016. Article (CrossRef Link).
[13] Lakshmanan Laks V S, Pei Jian and Zhao Yan, "Efficacious Data Cube Exploration by Semantic Summarization and Compression," in *Proc. of the 29th International Conference on Very Large Data Bases(VLDB'03)*, pp. 1125-1128, September 9-12, 2003. Article (CrossRef Link).
[14] Pham Quang-Khai, Saint-Paul Regis and Benatallah Boualem, "Mine Your Own Business, Mine Others' News!," in *Proc. of the 11th International Conference on Extending Database Technology*, pp. 725-729, March 25-29, 2008. Article (CrossRef Link).
[15] Balaji J, Geetha T.V and Parthasarathi Ranjani, "Abstractive Summarization: A Hybrid Approach for the Compression of Semantic Graphs," *International Journal on Semantic Web and Information Systems (IJSWIS)*, vol. 12, no. 2, pp. 76-99, April, 2016. Article (CrossRef Link).
[16] Zhang Wei, "Graph-based Large Scale RDF Data Compression," in *Proc. of the 37th International ACM SIGIR Conference on Research & Development in Information Retrieva*l, pp. 1276-1276, July 6-11, 2014. Article (CrossRef Link).
[17] Che Wanxiang, Zhao Yanyan and Guo Honglei, "Sentence Compression for Aspect-based Sentiment Analysis," *IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP)*, vol. 23, no. 12, pp. 2111-2124, December, 2015. Article (CrossRef Link).
[18] Feldman Dan, Sung Cynthia and Sugaya Andrew, "iDiary: From GPS Signals to A Text-Searchable Diary," *ACM Transactions on Sensor Networks (TOSN)*, vol. 11, no. 4, pp. 1-41, December, 2015. Article (CrossRef Link).
[19] Cuzzocrea Alfredo and Chakravarthy Sharma, "Event-based Lossy Compression for Effective and Efficient OLAP over Data Streams," *Data & Knowledge Engineering*, vol. 69, no. 7, pp. 678-708, July, 2010. Article (CrossRef Link).
[20] Drinić Milenko, Kirovski Darko and Vo Hoi, "PPMexe: Program Compression," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 29, no. 1, pp. 3-es, January, 2007. Article (CrossRef Link).
[21] P. Yuan, P. Liu, B. Wu, H. Jin, W. Zhang and L. Liu, "TripleBit: A Fast and Compact system for large scale RDF data," in *Proc. of the VLDB Endowment,* vol. 6, nol. 7, pp. 517-528, May, 2013. Article (CrossRef Link).
[22] R. Baeza-Yates and B. Ribeiro-Neto, "Modern Information Retrieval," *ACM press,* pp. 463-466, 1999. Article (CrossRef Link).
[23] V. Raman and G. Swart, "How to wring a table dry: Entropy Compression of Relations and querying of Compressed Relations," in *Proc. of the 32nd International Conference on Very large data bases*, pp. 858-869, September 12-15, 2006. Article (CrossRef Link).
[24] M. Stonebraker, D. J. Abadi, A. Batkin, et al., "C-store: A Column-oriented DBMS," in *Proc. of the 31st International Conference on Very Large Data Bases*, pp. 553-564, August 30-September 2, 2005. Article (CrossRef Link).

[25] S. Davies and A. Moore, "Bayesian Networks for Lossless Dataset Compression," in *Proc. of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 387-391, August 15-18, 1999. Article (CrossRef Link).
[26] Babu S, Garofalakis M and Rastogi R., "SPARTAN: Using Constrained Models for Guaranteed-error Semantic Compression," *SIGKDD Explorations*, vol. 4, no. 2, pp. 11-20, June, 2002. Article (CrossRef Link).
[27] G. Schwarz, "Estimating the Dimension of A Model," *Annals of Statistics*, vol. 6, no. 2, pp. 461-464, March, 1978. Article (CrossRef Link).
[28] Gao Yihan and Parameswaran Aditya, "Squish: Near-Optimal Compression for Archival of Relational Datasets," in *Proc. of the 22nd ACM SIGKDD International Conference on knowledge discovery and data mining*, pp. 1575-1584, August 13-17, 2016. Article (CrossRef Link).
[29] J. Rissanen, "Generalized Kraft Inequality and Arithmetic Coding," *IBM Journal of Research and Development*, vol. 20, no. 3, pp. 198-203, May, 1976. Article (CrossRef Link).
[30] G. G. Langdon Jr, "An Introduction to Arithmetic Coding," *IBM Journal of Research and Development*, vol. 28, no. 2, pp. 135-149, March, 1984. Article (CrossRef Link).
[31] I. H. Witten, R. M. Neal and J. G. Cleary, "Arithmetic Coding for Data compression," *Communications of the ACM*, vol. 30, no. 6, pp. 520-540, June, 1987. Article (CrossRef Link).
[32] M. M. Gaber, A. Zaslavsky and S. Krishnaswamy, "Mining Data Streams:A review," *ACM Sigmod Record*, vol. 34, no. 2, pp. 18-26, June, 2005. Article (CrossRef Link).
[33] Cheng Long, Malik Avinash and Kotoulas Spyros, "Fast Compression of Large Semantic Web Data Using X10," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 9, pp. 2603-2617, September, 2016. Article (CrossRef Link).
[34] Urbani Jacopo, Maassen Jason and Bal Henri, "Massive Semantic Web data compression with MapReduce," in *Proc. of the 19th ACM International Symposium on High Performance Distributed Computing*, pp. 795-802, June 21-25, 2010. Article (CrossRef Link).
[35] Urbani J., Maassen N., Drost F. and Seinstra H. Bal, "Scalable RDF Data Compression with MapReduce," *Concurrency & Computation Practice & Experience*, vol. 25, no. 1, pp. 24-39, January, 2013. Article (CrossRef Link).
[36] Tan Yujuan, Jiang Hong and Feng Dan, "SAM: A Semantic-Aware Multi-tiered Source De-duplication Framework for Cloud Backup," in *Proc. of the 39th International Conference on Parallel Processing*, pp. 614-623, September 13-16, 2010. Article (CrossRef Link).
[37] Ran Jin, Chunhai Kou, Ruijuan Liu and Yefeng Li, "Efficient Parallel Spectral Clustering Algorithm Design for Large Data Sets under Cloud Computing Environment," *Journal of Cloud Computing*, vol. 2, no. 1, December, 2013. Article (CrossRef Link).

Ran Jin received the PhD degree from Donghua University, China, in 2015. He is currently a postdoctoral fellow in the College of Computer Science and Technology, Zhejiang University, China. His research interests include data mining, information retrieval and database.

Gang Chen received the BSc, MSc, and PhD degrees in computer science and engineering from Zhejiang University in 1993, 1995, and 1998, respectively. He is currently a professor at the College of Computer Science, Zhejiang University. His research interests include database, information retrieval, information security, and computer supported cooperative work. He is also the executive director of Zhejiang University-Netease Joint Lab on Internet Technology.

Anthony K.H. Tung received the BSc (second class honor) and MSc degrees in computer science from the National University of Singapore (NUS), in 1997 and 1998, respectively, and the PhD degree in computer science from Simon Fraser University, in 2001. He is currently an associate professor in the Department of Computer Science, NUS. His research interests include various aspects of databases and data mining (KDD) including buffer management, frequent pattern discovery, spatial clustering, outlier detection, and classification analysis.

Lidan Shou received the PhD degree in computer science from the National University of Singapore. He is a professor with the College of Computer Science, Zhejiang University, China. Prior to joining the faculty, he worked in the software industry for more than two years. His research interests include spatial database, data access methods, visual and multimedia databases, and web data mining.

Beng Chin Ooi is currently a distinguished professor of computer science at the National University of Singapore. His research interests include database system architectures, performance issues, indexing techniques and query processing, in the context of multimedia, spatiotemporal, distributed, parallel, peer-to-peer, inmemory, and cloud database systems. He has served as a PC member for a number of international conferences (including SIGMOD, VLDB, ICDE, WWW, EDBT, DASFAA, GIS, KDD, CIKM, and SSD). He was an editor of the VLDB Journal and the IEEE Transactions on Knowledge and Data Engineering, editor-in-chief of the IEEE Transactions on Knowledge and Data Engineering (2009–2012), and a co-chair of the ACM SIGMOD Jim Gray Best Thesis Award committee. He is serving as a trustee board member and the president of the VLDB Endowment. He is a fellow of the IEEE and ACM.