Research on Deep Web Query Interface Clustering Based on Hadoop

Baohua Qiang[1], Rui Zhang[2], Yufeng Wang[3], Qian He[1], Wei Li[1] and Sai Wang[1]
[1]Guangxi Key Lab of Trusted Software, Guilin University of Electronic Technology, Guilin 541000, China
[2]North China University of Water Resources and Electric Power, Zhengzhou 450045, China
[3]The 54th Research Institute of China Electronics Technology Group Corporation, Shijiazhuang 050000, China
Email: qiangbh@guet.edu.cn Email: zhangrui@ncwu.edu.cn

***Abstract*—How to cluster different query interfaces effectively is one of the most core issues when generating integrated query interface on Deep Web integration domain. However, with the rapid development of Internet technology, the number of Deep Web query interface shows an explosive growth trend. For this reason, the traditional stand-alone Deep Web query interface clustering approaches encounter bottlenecks in terms of time complexity and space complexity. After further study of the Hadoop distributed platforms and Map Reduce programming model, a Deep Web query interface clustering algorithm based on Hadoop platform is designed and implemented, in which the Vector Space Model (VSM) and Latent Semantic Analysis (LSA) are employed to represent "Query Interfaces-Attributes" relationships. The experimental results show that the proposed algorithm has better scalability and speedup ratio by using Hadoop architecture.**

***Index Terms*—Hadoop, Map Reduce, Deep Web, LSA, Query Interface Clustering**

I. INTRODUCTION

According to the depth of the information, the Web can be divided into "Surface Web" and "Deep Web". With the rapid development of Internet technology, the information contained on the Web, especially on the Deep Web, is showing an explosive growth trend. As Bright Planet speculated in year 2000 that the entire Internet contains 40 to 90 thousands of Deep Web pages, the information capacity of which is about 7500T [1]. MetaQuery had made more accurate statistics about the whole internet Deep Web pages in the year 2004; the results show that there were some 450 thousands of Deep Web databases [2]. It turns out that the amount of the pages had increased by nearly 9 times after only 4 years. Compared to that contained on the "Surface Web", the information on the "Deep Web" has 5 characteristics below: (1) it could not be obtained by traditional search engines; (2) users acquire the information by filling out a form; (3) the information has a higher quality and a larger quantity; (4) the domain characteristics is more obvious and ; (5) most of the information has a free access. The explosive growth of information contained on Deep Web as well as the great value strongly attracts the attention of academia and business community.

As the Deep Web information portal, how the Deep Web query interface can be clustered effectively is one of the core issues need to be addressed while generating the integrated query interface [3]. At present, researches on Deep Web mainly focus on query interface integration algorithm based on single machine [4-7]. The proposed algorithms can effectively match the related query interfaces among a few Deep Web sites. But facing with the huge amounts of emerging Deep Web databases, the present approaches encounter great challenges in terms of time complexity and space complexity. So it is absolutely necessary and meaningful to study how to use distributed platforms to analyse the massive information on the Deep Web.

In view of the massive characteristic of Deep Web query interfaces, an effective approach is to introduce the parallel processing technology and design a rational and efficient parallel clustering algorithm [8]. Hadoop as a software framework which is able to make a distributed processing on massive data has been widely used. It has high reliability, scalability, efficiency and high fault tolerance [9]. On the basis of further study of the Hadoop platform, we designs and implements Deep Web query interface clustering algorithm on Hadoop platform, and before clustering, we employ constructed domain ontology and latent semantic analysis to make semantic expansions. Thus, the effectiveness of the query interfaces clustering is further improved. Besides, we have verified the correctness and effectiveness of the parallel algorithm design from the recall ratio and precision ratio in contrast to the results on single machine experiment. The results also show that the proposed parallel algorithm has good scalability and speedup ratio.

The main body of this paper is organized as follows. Firstly, we introduce the Hadoop architecture for Deep Web query interface clustering in Section II. And Section III describes the approach of parallelizable clustering algorithm based on VSM and LSA. Then the detailed Deep Web query interface clustering algorithm based on Hadoop is presented in section IV. The experiments and conclusion are given in Section V and Section VI, respectively.

doi:10.4304/jsw.9.12.3057-3062

II. HADOOP PLATFORM ARCHITECTURE FOR DEEP WEB QUERY INTERFACE CLUSTERING

Taking Hadoop Distributed File System (HDFS) and MapReduce as the core, Hadoop provide users with distributed infrastructure which is transparent in system bottom layer [10]. With high fault tolerance, high scalability of HDFS, users can deploy Hadoop on cheap hardware to form a distributed system. MapReduce allows users to develop parallel applications without knowing the details of the distributed system bottom layer. Thus, users can easily build their own distributed platforms and finish the processing of massive data using the computing and storage capability of the cluster.

HDFS uses Master/Slave structured distributed file system and an HDFS cluster consists of a NameNode and several DataNodes. NameNode as the primary server, manages the file system namespace and client access to file operations; DataNode manages the stored data. HDFS allows users to store the data in the form of documents. Internally, the file is divided into several data blocks and they are stored in a group of DataNodes. NameNode performs file system namespace operations, such as open, close, rename a file or a directory, it is also responsible for the mapping from data blocks to specific DataNode. The task of DataNode is processing the read and write requests of the system clients and handling create, delete, and copy to data blocks under the unified coordination of NameNode. Fig. 1 shows the architecture of HDFS.

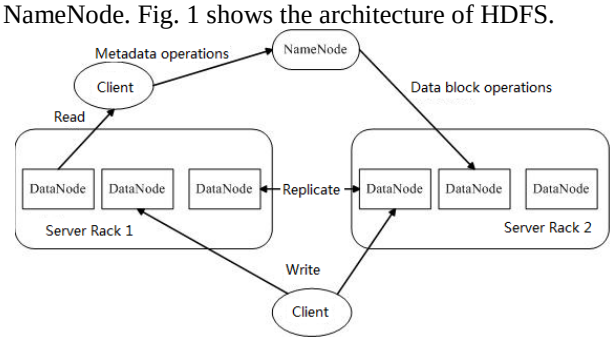

Figure 1. HDFS architecture

MapReduce is a parallel programming model that enables software developers to write distributed parallel programs easily. In the Hadoop architecture, MapReduce is a software framework that is easy to use, the principle of which is: use an input <key, value> collection to produce an output of <key, value> collection; Specifically, MapReduce framework consists of two stages: Map and Reduce. In Map stage, MapReduce divides the input data of the task into fixed-size split, each split is then further broken into a number of key values<key1, value1>. After that, Hadoop creates a Map task for each split to execute user-defined Map functions, takes the corresponding split in <key1, value1> as input, and then calculates and generates an intermediate <key2, value2> collection. MapReduce collects all the value collections that have the same key, forms <key2, list(value2)>, and then divides the meta group into several groups according to the range of the key, corresponding to different Reduce tasks. In the Reduce stage, Reducer integrates the received data from different Mappers together and sorts them according to the key value, then calls the user-defined reduce function and processes the input <key2, list(value2)> to obtain the key value <key3,value3> and then output to HDFS. Fig. 2 shows the process of MapReduce data processing.

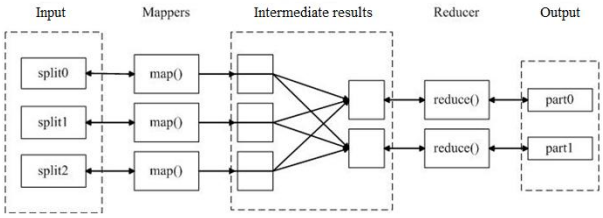

Figure 2. The process of MapReduce data processing

To sum up, the distributed storage used by Hadoop platform can improve the read and write speed and expand the storage capacity; using MapReduce programming model to integrate data on HDFS will ensure the efficiency of data analysis and processing. In view of the rapid growth of the information contained on the Deep Web, if we want to store and manage these useful data and information efficiently and then make further analysis, Hadoop platform is undoubtedly an excellent choice.

III. DETERMINE THE LATENT SEMANTIC RELATIONSHIPS BASED ON VSM AND LSA

A. The Vector Space Model of Deep Web Query Interface

The first step in clustering Deep Web query interface is to convert the Deep Web query interface set to Vector Space Model (VSM) [11]. Assume that we get N Deep Web query interface expressed as $F = \{f_1, f_2 \cdots f_n\}$ and consider it as the column index of VSM model, with $A = \{a_1, a_1 \cdots a_m\}$ representing all attributes obtained from F and consider it as the row index. So that we get a "Query Interfaces-Attributes" matrix C:

$$C = (c_{ij})_{m \times n} \quad (1)$$

Each row of this matrix represents a single attribute and each column stands for a single query interface, the element indicates the number of attributes occurred in query interface. TF-IDF weight will be selected to evaluate the importance of attributes, its basic thought is: if an attribute in a query interface appears a lot, it will also appears much in another similar query interface, and vice versa. Weight is calculated as follows:

$$w_{ij} = TF_{ij} \times log(N/DF_i) \quad (2)$$

Where TF_{ij} represents the numbers of times attribute a_i occurs in query interface f_j; N stands for the total number of query interfaces; DF_i signifies the total number of attribute a_i appears in the N query interfaces. While calculating the distance between cluster objects, the general way is to use Euclidean distance, but considering the existing difference about the number of query interface in different areas, we choose the Cosine Similarity, and it is calculated as follows:

$$sim(i_1, i_2) = \frac{\vec{V}(i_1) \cdot \vec{V}(i_2)}{|\vec{V}(i_1)| \| \vec{V}(i_2)|} \quad (3)$$

At this point, the vector space model of Deep Web query interface is set up.

B. Determine the Latent Semantic Relationships by LSA

VSM can be used to compute the similarities of Deep Web query interfaces by evaluating keywords matching literally. But it is difficult to determine their latent semantic relationships. Latent semantic analysis (LSA) is an effective indexing and searching approach [12], which can be employed to discern the latent semantic similarities among Deep Web query interfaces through constructing latent semantic space model.

The core issue of LSA is how to project high-dimensional "Query Interfaces-Attributes" matrix to lower dimensional latent semantic space by low rank approximation effectively [3]. Singular value decomposition (SVD) is the mathematical basis of LSA. Let C be the $m \times n$ "Query Interfaces-Attributes" matrix, it can be represented as formula 2.1 by SVD:

$$C = U \Sigma V^T \quad (4)$$

U and V stand for orthogonal matrix of $m \times n$, and their columns are orthogonal feature vector of CC^T, respectively. Σ is $m \times n$ matrix, Σ needs to be explained specially:

(1) The eigenvalues of $C_{m \times n}^T C_{m \times n}$ are $\lambda_1, \lambda_2 \cdots \lambda_r$;

(2) $\forall i \in [1, r]$, there exists $\delta_i = \sqrt{\lambda_i}$ and $\lambda_i \geq \lambda_{i+1}$, $\Sigma_{m \times n}$ meets $\Sigma_{ii} = \delta_i$, and other elements of matrix is 0. Σ_{ii} is also called singular value of $C_{m \times n}$.

In LSA, noise data can be removed by low rank approximation [4]. Low rank approximation is defined as follows. Suppose C is a matrix of $m \times n$, its rank is r, and C_k is a matrix of $m \times n$ with rank K and $r \geq k$. Let $X = C - C_K$, if X's norm F as formula 5 is the smallest one, we call C_k is the low rank approximation matrix of C when k is much smaller than r.

$$\| X \|_F = \sqrt{\sum_{i=1}^{M} \sum_{j=1}^{N} X_{ij}^2} \quad (5)$$

SVD is an effective means to solve the problem of low rank approximation. We can firstly obtain Σ_k by reserving the k biggest singular values and setting other $r-k$ singular values as 0 of $\Sigma_{m \times n}$, then calculate $C_k = U \Sigma_k V^T$ according to formula 4, finally we can obtain the approximation C_k of C. Theorem is shown as follow:

$$\min_{Z | rank(Z) = k} || C - Z ||_F = || C - C_k ||_F = \sqrt{\sum_{i=k+1}^{r} \delta_i^2} \quad (6)$$

It can be demonstrated that the above process will produce a matrix C_k with rank k, and its norm has the minimum error. By VSM and LSA, the latent semantic similarities among Deep Web query interfaces can be obtained effectively.

IV. Deep Web Query Interface Clustering Algorithm Based On Hadoop

A. Parallelizable Clustering Algorithms

K-Means and K-medoids clustering are the most commonly used partition-based algorithms in clustering domain [13]. The latter is more "robust" than the former when there were noise and outliers. Unlike mean value, the K-medoids clustering is not sensitive to outliers or other extreme values. However, it is not suitable for distributed scenarios. In contrast, the similarity calculation between each node and the center point in K-Means algorithm is independent, and the center point calculation is done in one cluster. In terms of the code implementation, a new center point calculation could be finished in one reduce function; therefore it is very suitable for parallelization transformation. Sequential execution procedure of the K-Means algorithm is given below firstly, and its parallel design will be given in subsequently.

Input:
k: the number of the Clustering;
S: Deep Web Query Interfaces Set

Output:
K clusters

Steps:
(1) Represent "Query Interfaces-Attributes" matrix with VSM and find the latent semantic relationships by LSA;
(2) Randomly select k query interfaces from S as the initial center point;
(3) Repeat
(4) Classify each query interface into the most similar cluster of center point;
(5) Calculate the mean value of each cluster as the new center point;
(6) Until no change occurs

B. Map Function Design

TextInputFormat is the default input method of Hadoop, each split is separately a map input, each row of data will generate a record and each record is represented as a form of <key, value> which can be accepted by map function, and key represents record byte offset in current split, the type of which is LongWritable, value stands for the content of each row, the type of which is Text. As in this experiment, value is represented by the column string of the vector space model matrix of the query interface. The setup function is executed prior to map in Mapper, and it is executed only once in the Mapper life cycle. Therefore, we can do some initialization operation in the function. The role of setup function in this algorithm is initializing the center point of each cluster, and then storing it into centerList. The pseudo-code of map function is given below:

```
map(LongWritable key, Text value){
/* Parse the value to node object */
    Node node = parse(value);
/* Calculate the center point that each node belongs to */
    node.setCenter(centerList);
/* Take the serial number of cluster that nodes belong to as
key, the string representation of current node as value */
      context.write(new IntWritable(node.center.id),
                    new Text(node.toString()));
    }
```

C. Combine Function Design

Combine process is a part of Mapper and executed after map function. Normally, it can effectively reduce the number of intermediate results; thereby reduce network traffic during data transmission. If designed properly, this process can significantly enhance the execution efficiency of the program. The pseudo-code of the Combine is as follows:

```
combine(IntWritable key, Iterator<Text> values){
  /*Parse "values" to Node, record the number of node in
    values set, then use "count" to save the node number
    belonging to cluster key in current split */
      int count = 0;
      float[] vector;
    while(values.hasNext()){
    Node node = parse(value.next());
  /*Accumulate the component of each node and prepare for
    the new center point that reduce function will calculate */
    vector = plus(vector, node.vector);
      count++;
      }
  /*Splice "vector" and "count" into string value1 */
    Text value1 = toString(count) + "#" +
          toString(vector);
    /* Output "key" and "value" */
    context.write(new IntWritable(key),
          new Text (value1));
      }
```

D. Reduce Function Design

The parameter that reduce function receives is <IntWritable key, Iterator<Text> values >, in which key is the serial cluster number, values are the string representation of all the node component values in cluster. Reduce function is similar to combine function and pseudo-code is as follows:

```
reduce(IntWritable key, Iterator<Text> values){
  /* Parse values to Node and use count to record the
  number of node in values set, define newCount to store the
  node number of each cluster. Use vector to store the
  component of center point */
  int count = 0;
  int newCount = parseInt(value);
  float[] vector;
  while(values.hasNext()){
      Node node = parse(value.next());
      vector = plus(vector, node.vector);
      count++;
  }
 /* Update newCount */
  newCount += count;
 /* Update the component of center point */
  vector /= newCount;
  /* value1 is the string representation of vector */
  Text value1 = toString(count) + "#" +
          toString(vector);
  /* Output key and value1 */
  context.write(new IntWritable(key),
          new Text (value1));
}
```

After a round of , we get newCenters, then compute the next round until convergence.

V. EXPERIMENTS

A. Selection of the Experiment Data

The data in this experiment comes from the Web integration resource library of UIUC which contains the query interface in many fields and store in the form of XML [14]. 221 query interfaces in 4 domains i.e. airfares, automobiles, books and musicRecords are selected in the experiment. In order to simulate the situation of big data, we have made a proportional copy of the original data to expand the size. The experiment data is shown in Table I:

TABLE I
EXPERIMENT DATA

Data Groups	Data Size	Number of Query Interfaces	Copy Multiples
A	1G	210613	953
B	2G	421005	1095
C	3G	631397	2857
D	4G	841789	3809
E	5G	1052181	4761

B. Experiment Data Preprocessing

The irregular definition of query interface attributes has brought too much noise, because of that, some part of query interfaces are lack of enough semantic information. In the light of the characteristics of this experiment, four steps are needed to process the data:

(1) Remove the stop words, but retain the words that have domain meanings, for example in the airfares domain "to", "from" etc.

(2) Stemming reduction and morphology normalization. Revert the different state of attribute words, singular and plural forms to the stemming of the word.

(3) Semantic expansion of query interfaces. Build domain ontology for each of these domains. Take the aviation domain for example, we can build ontology. Fig. 3 shows the hierarchy diagram of aviation domain ontology:

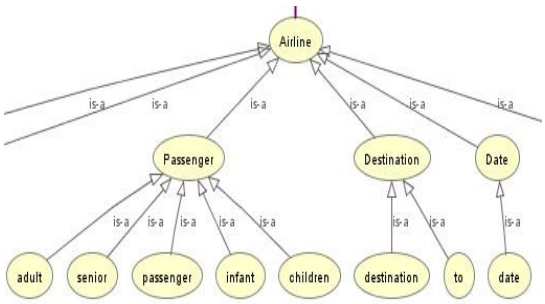

Figure 3. Aviation domain ontology

If an attribute of query interface appears in the ontology, the related attributes of the entire path will be added to the attribute set of the query interface and make semantic expansion.

(4) The latent semantic analysis to query interface is employed to find the latent semantic relations among Deep Web query interfaces and improve the similarity of query interface belonging to the same domain.

C. Experiment Environment

The experiment environment is composed of 4 HP desktops of the same model, configured as follows:

TABLE II
EXPERIMENT ENVIRONMENT CONFIGURATION

CPU	Intel Pentium 2.80HZ dual-core
RAM	4GB
OS	Ubuntu12.4
Hadoop Version	0.20.2

D. The Recall and Precision

The results of Deep Web query interface are unordered collections, so we choose precision ratio P and recall ratio R as criteria to evaluate our proposed algorithm. The formula of R and P is given below:

Where tp (true cases) represents the correct assignment to the corresponding cluster cases; fp (pseudo-positive cases) indicates the wrong assignment cases; fn (pseudo-negative cases) represents the cases that is assigned to the cluster but is not retrieved. The specific values of precision P and recall ratio R is shown in Fig. 4:

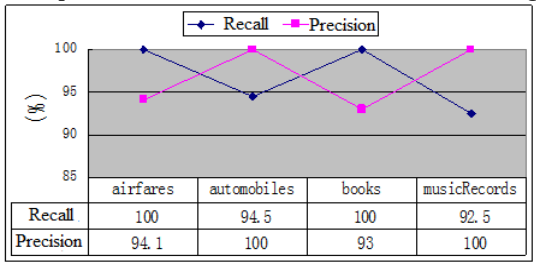

Figure 4. Comparison of Precision and Recall

Since this experiment expands the size of the data by replicating the raw data, 5 groups of Deep Web query interface data: A、B、C、D、E are respectively tested , each group of data has been run respectively on the cluster of 2、3、4 machines. Fixed initial centers, recall ratio and precision ratio remains unchanged. We also get the same P and R value in the case of single machine. Besides, the experiment results that the value of P and R are greater than 90% are also very encouraging. So we can say that the map/reduce distributed algorithm of Deep Web query interface clustering is reliable and significant in the era of big data.

E. Cluster Scalability

Run the data of A, B, C, D, E group on the cluster formed by different number of nodes and compare the run-time. The results are shown in Fig. 5:

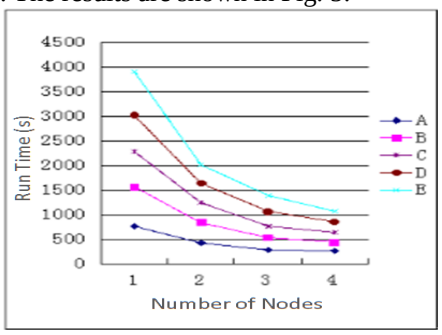

Figure 5. Run-time of different nodes

From the Figure above we can see that when process the same number of query interface, if the number of nodes in the cluster increases, time consuming significantly reduces; the larger the data size, the faster the run-time speed decreases. Therefore, while dealing with the large-scale data, we can improve the process capability of the system by increasing the number of nodes, which reflects the good scalability of the system.

F. Cluster Speedup Ratio

Speedup ratio [15] is the time-consuming ratio of the same task running on a single-processor system and parallel processor system. It is used to measure the performance and effects in parallel system or program parallelization. Speedup ratio is calculated as follows:

Where Sp represents Speedup ratio, denotes the running time in a single processor, denotes the running time in a p-processors parallel system. The experiment results are shown in Fig. 6:

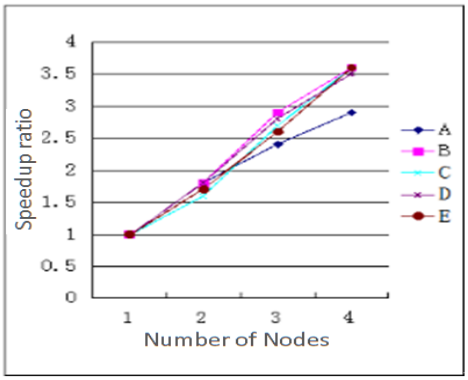

Figure 6. Speedup Ratio

As can be seen from Fig. 6, the speedup ratio of the algorithm is close to linear speedup ratio, with the increment of the data scale, the speedup ratio of the distributed system tend to stabilize. This fully demonstrates the advantages of handling with big data on Hadoop platform. In the view of algorithm design, we introduce the combine function between map and reduce

which can effectively merge locally, reduce the network data transmission between different nodes in cluster, and greatly reduce the unnecessary time consuming. What's more, due to the reasonable design of data structure, the extra system time consuming is also reduced correspondingly.

VI. CONCLUSIONS

In light of the bottleneck that Deep Web query interface clustering meets in handling with massive data on traditional single machine, we designed and implemented the Deep Web query interfaces clustering algorithm based on Hadoop. By experiment on different scale of data sets and different nodes of cluster, the results show that our proposed algorithm has excellent scalability and speedup ratio. However, there is still room for improvement in the implementation details of the algorithm and the platform configurations, for example: how to compress the data to reduce the pressure of network bandwidth; how to set a more reasonable number of reducer, etc. Therefore, the next step we will focus on the Hadoop platform and algorithm design, and further tap the potential of cluster computing.

ACKNOWLEDGEMENT

This work is supported by National Natural Science Foundation of China (grant 61163057, 61201250, 61363029, 61462020), Guangxi Nature Science Foundation (grant 2012jjAAG0063), Open Fund of Guangxi Key Laboratory of Trusted Software (kx201308). The authors would also like to express their gratitude to the anonymous reviewers for providing helpful suggestions.

REFERENCES

[1] Bergman M K. The Deep Web: Surfacing Hidden Value. Bright Planet white paper in Journal of Electronic Publishing. 2001, 7(1):8921-8914
[2] Bin He, Mitesh Patel, Zhen Zhang, Kevin Chen-chuan Chang. Accessing the Deep Web. Communications of the ACM. 2007, 50(5):94-101
[3] Li Yanni, Wang Yuping, Jiang Peng. Multi-objective Optimization Integration of Query Interfaces for the Deep Web Based on Attribute Constraints. Data and Knowledge Engineering, 2013, 86(1):38-60
[4] Furche Tim, Gottlob Georg, Grasso Giovanni. The Ontological Key: Automatically Understanding and Integrating Forms to Access the Deep Web. VLDB Journal, 2013, 22(5):615-640
[5] Wei Liu, Xiaofeng Meng, and Weiyi Meng. ViDE: A Vision-Based Approach for Deep Web Data Extraction. IEEE Transactions on Knowledge and Data Engineering. 2010, 22(3):447-460
[6] Balakrishnan Raju, Kambhampati Subbarao, Jha Manishkumar. Assessing Relevance and Trust of the Deep Web Sources and Results Based on Inter-Source Agreement. ACM Transaction on the Web, 2013, 7(2):1-32
[7] Furche Tim, Gottlob Georg, Grasso Giovanni. OXPATH: A Language for Scalable Data Extraction, Automation, and Crawling on the Deep Web. VLDB Journal, 2013, 22(1):47-72
[8] Tien James M. Big Data: Unleashing Information. Journal of Systems Science and Systems Engineering, 2013, 22(2): 127-151
[9] Gattiker A, Gebara F. H., Hofstee H. P. Big Data Text-oriented Benchmark Creation for Hadoop. IBM Journal of Research and Development, 2013, 57(3), 3-4
[10] Tom White. Hadoop: The Definitive Guide. O'Reilly Media, ISBN: 978-1-4493-1152-0, 2012
[11] G. Salton, A. Wong, C. S. Yang. A Vector Space Model for Automatic Indexing, Communications of the ACM, 1975, 18(11), 613–620.
[12] Olney, Andrew.M. Generalizing Latent Semantic Analysis. IEEE International Conference on Semantic Computing. 2009, p40-46
[13] Velmurugan T., Santhanam, T. Computational Complexity between K-Means and K-Medoids Clustering Algorithms for Normal and Uniform Distributions of Data Points. Journal of Computer Sciences, 2010, 6(3):363-368.
[14] http://metaquerier.cs.uiuc.edu/repository/datasets/tel-8/.
[15] Mostovyi Oleksii, Prokopyev Oleg A., Shylo Oleg V. On Maximum Speedup Ratio of Restart Algorithm Portfolios. Informs Journal on Computing, 2013, 25(2), 222-229.

Baohua Qiang was born in 1972. He received the B.E. degree and M.A. degree from Southwest University in 1996 and 2002 respectively and the Ph.D. degree from ChongQing University in 2005. He went to University of Illinois as a visiting scholar from May to August in 2007. He had finished his postdoctoral research at South China University of Technology from 2007 to 2009. Now he works at Guilin University of Electronic Technology. His major research interests are Web information processing, intelligent search, massive data processing and network information integration.

Rui Zhang was born in 1980. She received her Master degree at information institute in Southwest University at 2006 and now is studying for her doctorate. Her research interests are Software Engineering, Data Mining and Big Data.