TELKOMNIKA Indonesian Journal of Electrical Engineering
Vol. 12, No. 11, November 2014, pp. 7927 ~ 7934
DOI: 10.11591/telkomnika.v12i11.6357

Design and Analysis of Parallel MapReduce based KNN-join Algorithm for Big Data Classification

Xuesong Yan*, Zhe Wang, Dezhe Zeng, Chengyu Hu, Hao Yao
School of Computer Science, China University of Geosciences, Wuhan, China
*Corresponding author, e-mail: yanxs1999@126.com

Abstract

In data mining applications, multi-label classification is highly required in many modern applications. Meanwhile, a useful data mining approach is the k-nearest neighbour join, which has high accuracy but time-consuming process. With recent explosion of big data, conventional serial KNN join based multi-label classification algorithm needs to spend a lot of time to handle high volumn of data. To address this problem, we first design a parallel MapReduce based KNN join algorithm for big data classification. We further implement the algorithm using Hadoop in a cluster with 9 vitual machines. Experiment results show that our MapReduce based KNN join exhibits much higher performance than the serial one. Several interesting phenomenon are observed from the experiment results.

Keywords*: multi-label classification, KNN join, MapReduce*

1. Introduction

Multi-label classification, the exception of single-label classification, is highly required in many modern applications, such as protein function classification, music categorization and semantic scene classification. In past decades, multi-label classification have been made a significant contribution to bioinformatics, especially to protein subcellular localization **Error! Reference source not found.**]. To put it simply, multi-label classification **Error! Reference source not found.**] is a mining method that assigns a set of labels to an unseen instance. Each instance in multi-label classification can be identified by a set of labels $Y \subseteq L$, $|L| > 2$. For example, the famous song named Scorpions can be classified into both 'rock' and 'ballad'. For semantic scene classification, a photograph can be labeled with more than one genre such as mountains, lakes and forests in a similar way.

In essence, there are two methods for multi-label classification: (1) Problem transformation and (2) algorithm adaptation **Error! Reference source not found.**]. And one frequently-used method of algorithm adaptation methods is k nearest neighours **Error! Reference source not found.-Error! Reference source not found.**] which depends on similarity searches. The similiarity join has become an important database primitive for supporting similarity searches and data mining **Error! Reference source not found.**]. As one operation of three well-known similiary join, the k-nearest neighbour join (KNN join) retrieves k most similar pairs and is frequently applied to mumerous applications including knowledge discovery, data mining, and spatial databases **Error! Reference source not found.**, **Error! Reference source not found.**]. Since both the join and KNN search are expensive, especially on large data sets and/or in multi-dimensions, KNN join is a costly operation. Lots of research, in the literature **Error! Reference source not found.**],[**Error! Reference source not found.-Error! Reference source not found.**] bave been devoted to impove the performce of KNN join by proposing efficient algorithms, many of which have been focused on improving algorithm and the centralized, single-thread setting that is impossible used in a distributed system. With the rapid explosion in the volume of data in big data era, the multi-label classification using serial KNN join cannot satisfy our needs already.

P.Malarvizhi et al. **Error! Reference source not found.**] propose an algorithm of classifying labels to the documents of the web. In their approach, they use binary classification of binary classifier based on MapRedcue framework to assign the set of positive label to the

Received July 7, 2014; Revised September 19, 2014; Accepted October 6, 2014

documents of the web. Their method needs numerous reduce functions when each instance in datasets have masses of labels.

In recent years, MapReduce**Error! Reference source not found.**] has been widely used in industry and academia. It is regarded as a simple yet powerful parallel and distributed computing paradigm to explore the cloud computing resources. Meanwhile, MapReduce architecture has good scalability and fault tolerance mechanisms so that it already becomes the one of the mostly used parallel and distributed systems. Therefore, we are motivated to incorporate MapReduce architecture into the design of distributed and parallel KNN algorithm for big data multi-label classification.

The main contributions of this paper are as follows:

a) We novelly apply the MapReduce concept in the design of distributed and parallel KDD algorithm for big data classification.
b) We actually implement our algorithm in a cluster with X servers. Extensive performance evaluations are conducted. Specially, we also analyze the influence of cluster size of MapReduce on the categorization (performance/accuracy)?Performance evaluation results also validate the high performance of our algorithm over conventional serial one.

2. KNN join

KNN join, proposed by [**Error! Reference source not found.**] is an important similary join operation and it combines each point of one point set with its k nearest neighbours in the other set. For example, it is the join of the k nearest neighbors(NN) of every point in a dataset R from a dataset S **Error! Reference source not found.**]. Each record in R(or S) is represented as a d-dimension point. For one point in dataset R such as r point, we get knn(r, S) by calculating the similarity distances which is Euclidean distances d(r, s), between r in R and every record in S in this paper.

KNN join algorithm is as follows:

$$knnJ(R,S)=\left\{\left(r,knn\left(r,S\right)\right)\middle|\ for\quad all\quad r\in R\right\} \tag{1}$$

From the above, we can depicte the KNN similarity join in figure 1. When 3 points in dataset R want to find two neighbours in dataset S, KNN join algorithm returns 6 results.

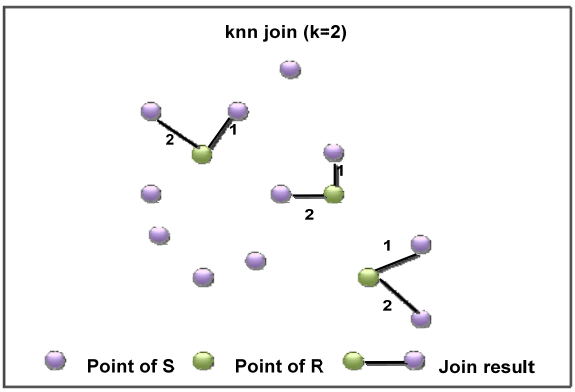

Figure 1. KNN Join Operations

3. MapReduce Cluster

Implemented by Hadoop, Google's MapReduce **Error! Reference source not found.Error! Reference source not found.**], is a programming model that can be created on a humble hardware condition and serves for processing large data sets in a massively parallel

manner. It can divides a task into some jobs and automatically parallelize and schedule jobs in a distributed system.

Figure 2 shows the data flow diagram of MapReduce. Firstly, MapReduce splits datasets into hundreds of thousands of small datasets. Secondly, one node which is a common computer generally processes one small dataset and products intermediate data. Finally, a large number of nodes merges the intermediate data and then products the final output data.

In the process of computing, the computation inputs a set of input key/value pairs and produces a set of output key/value pairs. Then the map function processes the input data and generates a series of intermediate data. The reduce function regards the above intermediate data as input data and produces the final output data.

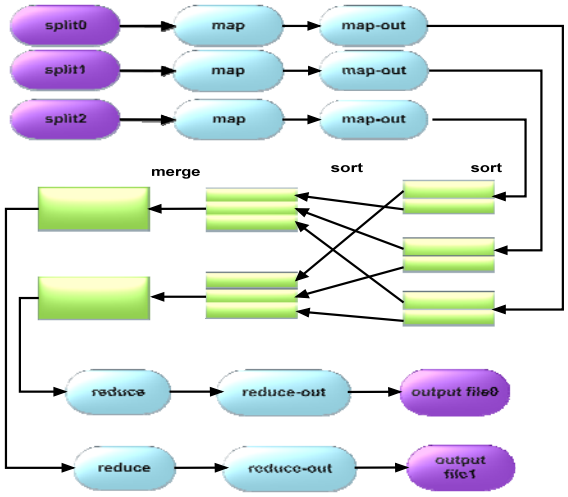

Figure 2. Data Flow Diagram of MapReduce

During the above process, it can be seen that two essential functions, i.e. map function and reduce function, are involved. Under such framework, developers shall design their own map function and reduce function based on according to the task requirement.

A MapReduce cluster consists of a master machine called master node and several slave machines called data nodes. The master node allocates map tasks and reduce tasks to data nodes and monitors their operations. A file in a MapReduce cluster is usually stored in a distri-buted file system(DFS) which splits a file into equal sized chunks. The splits are then distributed and replicated to all machines in the cluster. To execute a MapReduce job, users can decide the number of map tasks *m* and reduce tasks *r.* In most instances, *m* is the same as the nu-mber of splits for the given input file(s). After master node assigned map and reduce tasks to data nodes, the input and output of map and reduce functions are as follows:

map $<k1,value1>$ -> $<k2,value2>$
reduce $<k2,list(v2)>$ -> $list(v2)$

A combine function can be invoked between map function and reduce function to ease network congestion caused by the repetitions of the intermediate keys *k2* produced by map functions **Error! Reference source not found.**]. The combine function plays a similar but optional role with reduce function. It likes:

combine $<k2, list(v2)>$ ->$list<k2, v2>$

3. KNN Join Based on MapReduce

3.1. Data Normalization

We change the format of input datasets in order to get desired training files and testing files. In each dataset, we alter each old record expressed as <att1,att2,att3…,label1,label2…> to new record described as <id, att1, att2, att3,… label1, label2,…>.

3.2. Design of Parallel KNN Join

In KNN join, the most time-consuming step is the calculation of distance between instances in dataset R and instances in dataset S. Each instance in two files includes one id. Files exist in HDFS and are processed as <*key*, *value*> pairs which represent each record in the files **Error! Reference source not found.**]. Our parallel algorithm are divided into two phases.

Phase- I: Phase- I stores R*S instances and completes total distance calculations. Figure 3 shows flow diagram of Phase- I. First of all, we split all files and spread them across all mappers. Each split is sent to a Mapper in the form of <*key, value*> where *key* is the offset in bytes of this record to the start point of the data file and *value* is the content of this record. We mark the testing file(the training file) in file id =0 (=1) and partition the input files into multiple groups. An input record randomly generates a partition id named group id as the output key of map function and the output value is a string with the content of each record and relevant file id.

A list of intermediate <*key1*, *value1*> with the same key are sent to the same Reducer. *Key1* is unique group Id and *value1* is value lists obtained from map function. For value list with same *Key1*, values with file id = 0 (id = 1) are put into bucket R (bucket S), according to file id. The distance between R and S is then calculated as**Error! Reference source not found.**] . So for the output <*key2*, *value2*> pairs, *key2* is null and *value2* is a text containing a record's id from the testing file, i.e. id1 and id2, and the distance between these two records.

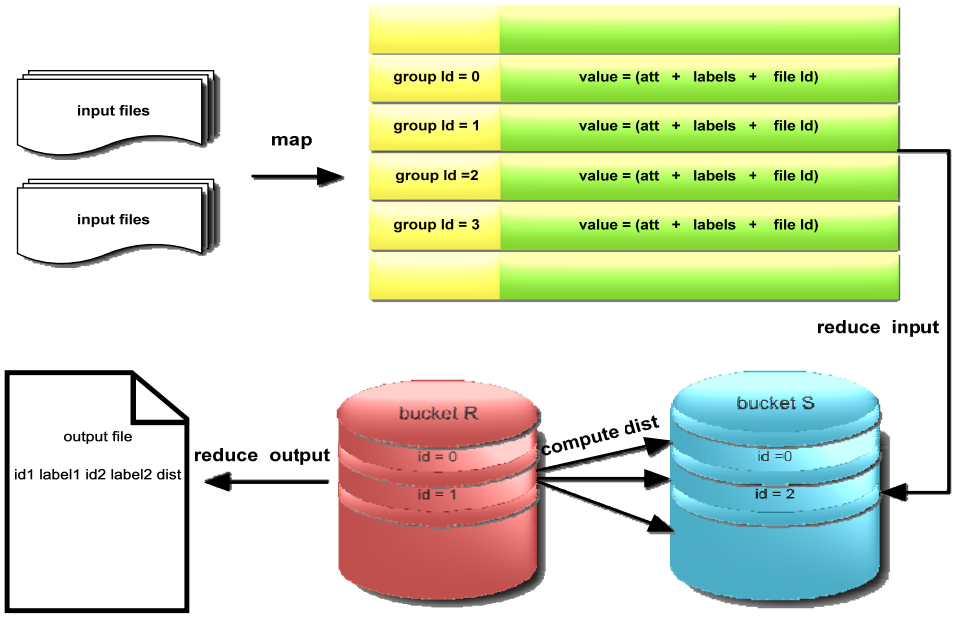

Figure 3. Flow Diagram of Phase- I

The pseudocode of Phase- I is summarized in Algorithm1.

Algorithm 1.

Map input:<*offset, original record*>
Map output:<*key1, value1*>, where *key'* is group Id and *value'* is a combination with origial record and file Id

1. Assgin each input record a group Id and a file Id
2. Take group Id as *key1*
3. Take the combination of original record and fileId as *value1*

4. End
Reduce input:< *key1, value1*>
Reduce output:<*key2, value2*>, where *key2* is null and *value2* is a combination with id1, id2 and distance.
1. Dis = ComputDist(training record, testing record)
2. Take null as *key'*
3. Take the combination of id1,id2 and distance as *value'*
4. End

Phase-II: After Phase- I, we get an intermediate file. For the input <*key*, *value*> pairs of map function, *key* is the offset and *value* is the content of the record in the intermediate file. Then we set id1 as the output's *key*, and *value* is the combination of id2 and distance.

In reduce function, we get k nearest neighbors of each testing record, the file id of which is 0, based on distances.

The pseudocode of Phase-II is summarized in Algorithm2.

Algorithm 2.
Map input:<*offset, record*>
Map output:<*key1, value1*>pairs, where *key1* is id1 and *value1* is a combination with id2 and distance
1. Take id1 as *key1*
2. Take the combination with id2 and distance as *value1*
3. End
Reduce input:< *key1, value1*>
Reduce output:<*key2, value2*>pairs, where *key2* is id1 and *value2* is the label set
1. For each *key1*, find its k nearest neighours
2. Determine the label set according to voting mechanism
3. Take id1 as *key2*
4. Take the label set as *value2*
5. End

4. Results and Analysis

4.1. Cloud Environment and Datasets

We actually implement our algorithm in Clustertech Cloud Business Platform (CCBP) with 9 vitual machines(VM), each of which contains 4*2.00GHz Dual-Core and 4GB RAM. Each VM runs Ubuntu 12.10 (64-bit) with hadoop-0.21.0. One VM serves as the master node and the other VMs act as slave nodes. We provide each slave with 100GB hard drive space and 20GB root space and allocates 5.3GB to DFS.

We utilize 4 common multi-label datasest in our experiments, including CAL500, emotions, yeast, scene, as shown in Table 1.

Table 1. 4 Multi-label Datasets

dataset	att_number	label_number
CAL500	68	174
emotions	72	6
yeast	103	14
scene	294	6

Table 2. Average Accuracy Rate of Parallel KNN Join and Serial KNN Separately for Four Datasets

Average accuracy rate	Parallel KNN join	Serial KNN join
emotions	72.66%	72.68%
CAL500	80.33%	80. 28%
scene	88.23%	87.46%
yeast	75.54%	75.53%

4.2. Performance Evaluation

We conduct experiments to evaluate parallel KNN in cluster of different size. For each dataset, 10 experiments are performanced for each cluster size and the average accuracy and time spent are calculated, as shown in Table 2 and Tabel 3 respectively. From Table 2, we can clearly see that the average accuracy rate of our parallel algorithm is almost as high as serial KNN join, which means that our algorithm is feasible and efficient.

From Table 3, we can see that the average time spent shows as a decreasing function of the cluter size. For example, for emotions, it takes 58.891s, 55.021s and 48.135s for 2 slaves, 4 slaves and 6 slaves, respectively. In order to facilitate analysis, we represented results in Figure 4.

Figure 4 plots average running times of 4 multi-label datasets versus different clusters of 2 slaves, 4 slaves, 6 slaves, 8 slaves respectively. Unit of measure is seconds. With the growth of slaves, the average running times of yeast and emotions linearly decrease. However, for CAL-500 and scene, Curve has the rising trend slightly sometimes, the reason of which is that the time saved by computing is less than the time increased by data communication of cluster. So the cluster of appropriate size makes scene.

Table 3. Average Running Times that Four Datasets Spent in 4 Clusters of 2 Slaves, 4 Slaves, 6 Slaves, 8 Slaves Respectively

	2 slaves	4 slaves	6 slaves	8 slaves
emotions	58.891s	55.021s	48.135s	46.346s
CAL500	56.318s	48.809s	49.616s	52.242s
scene	133.977s	122.001s	183.749s	123.17s
yeast	123.687s	118.746s	114.17s	108.115s

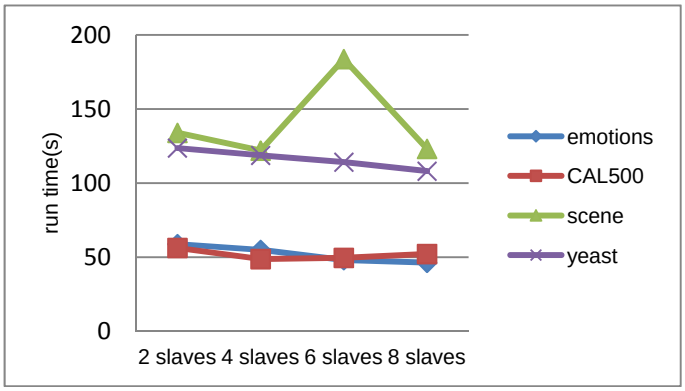

Figure 4. Average Running Times of 4 Multi-label Datasets versus Different Clusters of 2 slaves, 4 slaves, 6 slaves, 8 slaves respectively.

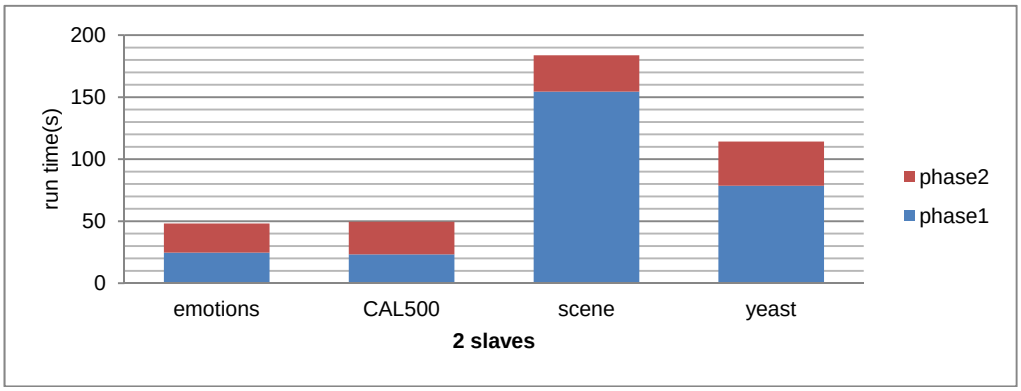

Figure 5. Running Time of phase1 and phase2 versus 4 datasets in cluster of 2 slaves

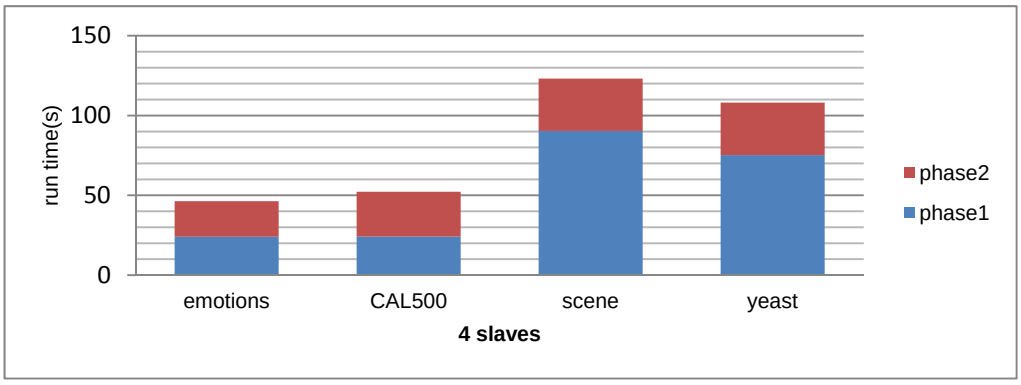

Figure 6. Running Time of phase1 and phase2 versus 4datasets in cluster of 4 slaves

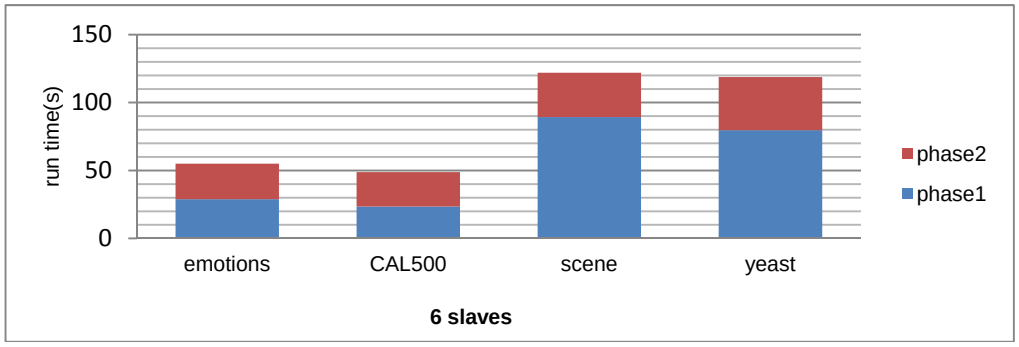

Figure 7. Running Time of phase1 and phase2 versus 4datasets in cluster of 2 slaves

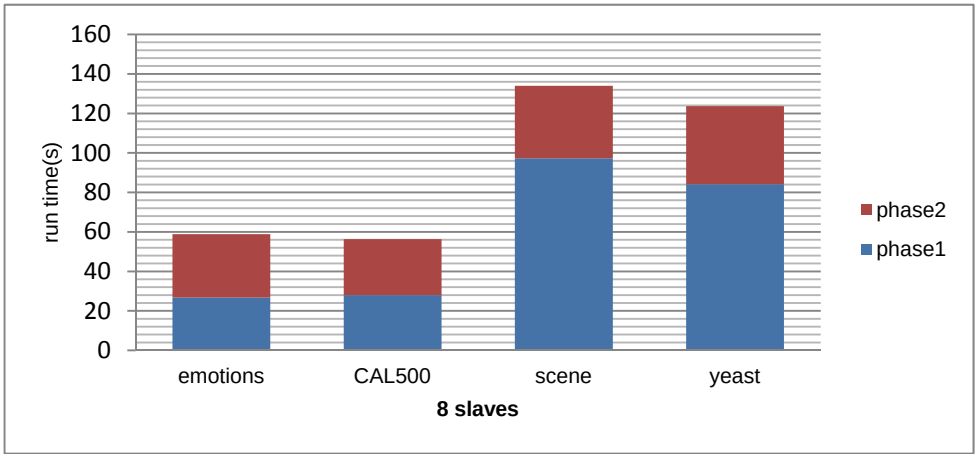

Figure 8. Running Time of phase1 and phase2 versus 4datasets in cluster of 2 slaves

To clearly show the time spent in the two essential phases in our MapReduce based KNN algorithm, we further investigate the time spent in Phase-I and Phase-II, respectively. Figure 5,6,7,8 show running time of Phase-I and Phase-II versus 4 datasets in cluster of 2 slaves, 4 slaves, 6 slaves and 8 slaves, respectively. We can see that show that Phase-I is the most time consuming phase and it decreases with the increase of cluster size. For example, for

dataset scene, when there are 2 slaves, 174.045s (78.4% of the total time) is spent on Phase-I and it decreases to 96.484s when the cluster size becomes 4.

4. Conclusion

In KNN join, the most time-consuming step is the calculation of distance between instances. With the increase of data, serial program cannot meet our requirements on the timeliness. In this paper, we design and implement a parallel KNN join using MapReduce for big data multi-label classification. From results above, we can see that the average accuracy of our parallel algorithm is almost as high as conventional serial KNN join and the average time spent shows as a decre-asing function of the cluter size. Therefore, our method is proved to be an efficient and feasible solution to multi-label classification dealing with big data.

Acknowledgements

This paper is supported by Natural Science Foundation of China. (No.61272470 and No.61203307), the Provincial Natural Science Foundation of Hubei (No. 2012FFB04101) and the Fundamental Research Founds for National University, China University of Geosciences (Wuhan).

References

[1] Wan S, Mak MW, Kung SY. *Adaptive thresholding for multi-label SVM classification with application to protein subcellular localization prediction.* Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on. IEEE. 2013: 3547-3551.
[2] Tsoumakas G, Katakis I. Multi-label classification: An overview. *International Journal of Data Warehousing and Mining (IJDWM)*. 2007; 3(3): 1-13.
[3] Zhang ML, Zhou ZH. ML-KNN: A lazy learning approach to multi-label learning. *Pattern recognition*. 2007; 40(7): 2038-2048.
[4] Zhang ML, Zhou ZH. *A k-nearest neighbor based algorithm for multi-label classification [C] //Granular Computing.* IEEE International Conference on. IEEE, 2005; 2: 718-721.
[5] Böhm C, Krebs F. The k-nearest neighbour join: Turbo charging the KDD process. *Knowledge and Information Systems*. 2004; 6(6): 728-749.
[6] Kavraki LE, Plaku E. Distributed Computation of the knn Graph for Large High-Dimensio- nal Point Sets.
[7] Xia C, Lu H, Ooi B C, et al. *Gorder: an efficient method for KNN join processing.* Proceedings of the Thirtieth international conference on Very large data bases-Volume 30. VLDB Endowment, 2004: 756-767.
[8] Yao B, Li F, Kumar P. *K nearest neighbor queries and knn-joins in large relational databases (almost) for free.* Data Engineering (ICDE), 2010 IEEE 26th International Conference on. IEEE. 2010: 4-15.
[9] Malarvizhi P, Pujeri RV. *Multilabel Classification of Documents with Mapreduce*. International Journal of Engineering & Technology (0975-4024). 2013; 5(2).
[10] Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters. *Communications of the ACM*. 2008; 51(1): 107-113.
[11] Zhang C, Li F, Jestes J. *Efficient parallel kNN joins for large data in MapReduce.* Proceedings of the 15th International Conference on Extending Database Technology. ACM. 2012: 38-49.
[12] Ranger C, Raghuraman R, Penmetsa A, Bradski G, Kozyrakis C. *Evaluating MapReduce for Multi-core and Multiprocessor Systems*. Proc. of 13th Int.Sympo- sium on High-Performance Computer Architecture (HPCA). Phoenix, AZ. 2007.
[13] Lämmel R. Google's MapReduce programming model Revisited. Science of computer programming, 2008; 70(1): 1-30.
[14] Zhao W, Ma H, He Q. Parallel k-means clustering based on mapreduce. Cloud Computing. Springer Berlin Heidelberg. 2009: 674-679.
[15] Liang Q, Wang Z, Fan Y, et al. *Multi-label Classification based on Particle Swarm Algorithm*. Mobile Ad-hoc and Sensor Networks (MSN). IEEE Ninth International Conference on. IEEE, 2013: 421-424.
[16] Fayed HA, Atiya AF. *A Novel Template Reduction Approach for the-Nearest Neighbor Method*. Neural Networks, IEEE Transactions on. 2009; 20(5): 890-896.