Acta Technica 62 (2017), No. 6B, 559–570 © 2017 Institute of Thermomechanics CAS, v.v.i.

Mass data fast retrieval in distributed cluster based on svm optimized neural network

Weibo Zhou[1], Yong Zhong[1]

Abstract. As for low retrieval efficiency of Lucene retrieval algorithm for big data, based on incremental retrieval of MapReduce, full-text Lucene engine and parallel retrieval method is proposed in the Thesis. By improving incremental retrieval, increasing capacity of index buffer zone and reducing indexing and writing frequency of disk files, we have achieved the goal of increasing efficiency in creating index, created parallel Lucene full-text retrieval engine based on MapReduce parallel computing module in Hadoop framework and thus realized parallelized execution of big data retrieval. In the end, establishing simulation environment for full-text retrieval of big data and displaying single-process retrieval mode of proposed parallel retrieval mode can substantially promote retrieval accuracy and execution efficiency. The simulation results show that proposed MapReduce-Lucene retrieval process has better retrieval accuracy and operating efficiency.

Key words. Incremental index, Big data, Intelligent algorithm, Distributed computation, Mass data, Data retrieval.

1. Introduction

The research field of search engine involves information inquiry, database technology, machine-aided training and multiple research disciplines. At present, companies with mature search engine voluntarily reserve it as private technology and therefore researches on application and extension of search engine technology are restricted to a certain degree [1, 2]. In recent years, Lucene technology attracts too much attention and its application field is increasingly wider. Therefore, researches on efficiency promotion of retrieval and index have great success, which lays a foundation for application of Lucene technology [3]. For this purpose, based on previous research achievements, MapReduce algorithm framework is used in design in the Thesis to realize parallelized improvement of Lucene retrieval process. At the same time, full-text retrieval experiment is designed and 102 to 106documents are selected and used

[1]Chengdu Institute of Computer Applications, Chinese Academy of Science , Chengdu, China

to establish test platform. Compared with previous achievements, the performance of parallel retrieval in different document scales can be used for experiment and test so as to verity effectiveness of parallel experiment.

2. Parallel framework of incremental Lucene system

2.1. System framework

In algorithm design process, based on users' application, we have carried out top-down analysis on system requirements. Later on, we have carried out effective division of system based on system requirements: 4 groups of modules such as system feature extraction, system storage, system inquiry and system retrieval. And then, derive the relation for system dependent components based on supporting technology, as shown in Fig. 1.

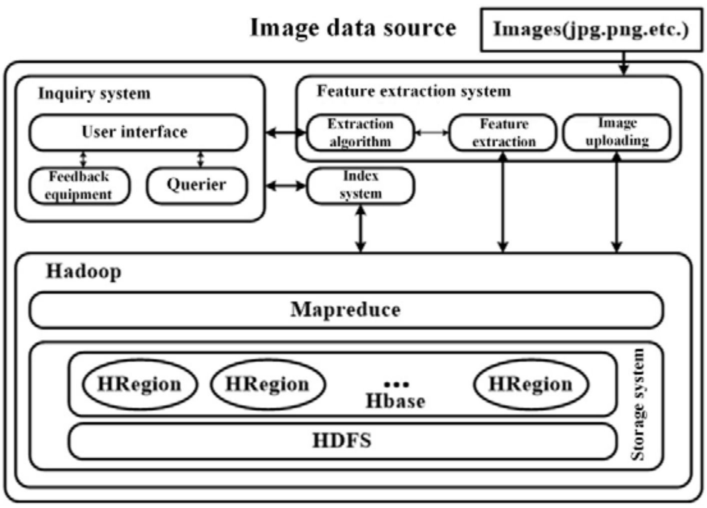


Fig. 1. System parallel framework

As shown in parallel framework in Fig. 1, above-mentioned four groups of modules are mutually dependent and are indispensable. Except the system inquiry module in parallel framework, the rest modules are established based on Hadoop framework and design modules mainly include: (1) system storage module. Functions of the module in system are mainly to realize storage of document and data, feature data extraction and retrieval; (2) system feature extraction module. Its function is mainly to execute and upload feature extraction data of the document in HDFS system structure. (3) System retrieval module. Establish corresponding index for extracted feature library. (4) System inquiry module. Design system inquiry module based on framework of B/S algorithm and it can be divided into feedback equipment, querier and 3 components at operating interface.

2.2. Incremental Lucene index

In system indexing process, the index creation process will be executed in MapReduce process. For this purpose, records in Hbase Table can be regarded as input in MapReduce computing framework. The key/value of this input value is recorded in Hbase Table as Row key and the value shall be columa family of this record. The function of IdentityTableMap in Hbase class library is to turn records into key/value of row key and Columa Family and then output it to Reduce operation process. For this purpose, we can directly use map() mapping mode in IdentityTableMap and then acquire a group of IndexTableReduce by customized method so as to accomplish the process of reduce() created by index. At the same time, to make it convenient for reasonable algorithm design, we need to accomplish the following operations in reduce() process: mapping row data in Hbase to Lucene document and then adopt byte serial for storage of row key and Columa Family. And then encapsulate this Lucene document and then carry out customized format output after encapsulation in LuceneDocumentWrapper. The format is based on IndexOutputFormat and then accomplish creating process of corresponding retrieval. Specific process are shown in Fig. 2.

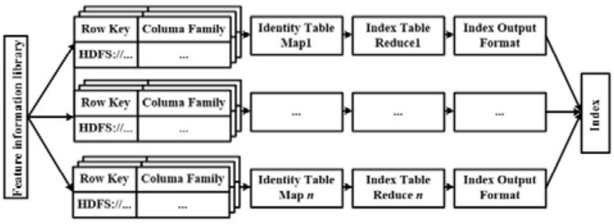


Fig. 2. Index process

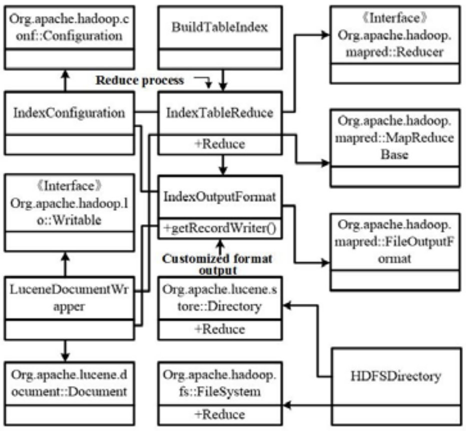

Org.apache.hadoop.conf::Configuration
BuildTableIndex
《Interface》 Org.apache.hadoop.mapred::Reducer
Reduce process
IndexConfiguration
IndexTableReduce
+Reduce
Org.apache.hadoop.mapred::MapReduceBase
《Interface》 Org.apache.hadoop.io::Writable
IndexOutputFormat
+getRecordWriter()
Org.apache.hadoop.mapred::FileOutputFormat
Customized format output
LuceneDocumentWrapper
Org.apache.lucene.store::Directory
+Reduce
HDFSDirectory
Org.apache.lucene.document::Document
Org.apache.hadoop.fs::FileSystem
+Reduce

Fig. 3. Incremental index system

Parallelized operation can be realized in index creation process and therefore we need to design based on incremental index and this design is based on HDFSDirectory and can inherit Lucene Directory. And then, we can encapsulate it into HDFS system structure to make it convenient for reading and storage of Lucene index on HDFS. After incremental Lucene parallel index is created, we need to execute feature parallel extraction of information retrieval to realize parallelized retrieval of information.

3. Storage of parallel extracted feature

3.1. Parallelized Mapreduce framework

Mapreduce parallel framework is the parallel execution framework of distributed big data which is relatively mature at present. At the same time, it has relatively mature application in cluster execution process of parallel big data. The name of Mapreduce derives from two process operations involved in this framework: map and reduce processes. The essence of map process is similar to that of mapping process and it is the mapping process of dataset members. After map process, feedback will be provided in list of results. As for results acquired in map process, parallelized structure will be used in reduce process to realize them. Specific conditions are shown in Fig. 4[14].

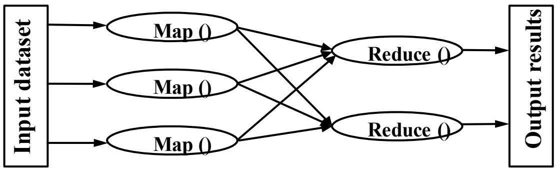


Fig. 4. Mapreduce model

In parallel framework execution, the problem we need to solve is the decomposition of mutual independent operation in data execution sub-process to make it convenient for acquiring full play of computing resource performance. In map and reduce processes, the algorithm framework is established mainly based on key-value involved in processing data. The specific form is:

$$Map : (k_1, v_1) \to [(k_2, v_2)] \; . \tag{1}$$

$$Reduce : (k_2, [v_2]) \to [(k_3, v_3)] \; . \tag{2}$$

In system execution process, the function of management host is to carry out dynamic and real-time adjustment on the number of host for parallel computation according to input setting of data breakpoint. At the same time, the management host is used to carry out real-time treatment of fault in computational process. At the same time, the management host can give command to programmers without hardware and operation experience to make it convenient for data resource distributed operation of large-scale structure.

3.2. Texture extraction process

Tamura texture extraction process is based on feature vector and it is the feature combination of 3 to 6 kinds of texture documents. Commonly used features at present include contrast, roughness and directionality [13].

Roughness indicator: measurement required for the indicator is the proportion between space size of texture feature and granularity document. In case the window dimension is: $2^{2k} \times 2^{2k}$, then the average value of moving times of gray value near the pixel point (i, j) is:

$$a_k(i,j) = \sum_{i=i-2^{k-1}}^{i+2^{k-1}-1} \sum_{j=j-2^{k-1}}^{j+2^{k-1}-1} \frac{p(i',j')}{2^{2k}} . \tag{3}$$

In Equation (1), $p(i', j')$ is the gray value near the pixel point (i', j') and its deviation value in vertical and horizontal directions are:

$$c_k(i,j) = \max \left(\begin{vmatrix} a_k\left(i-2^{k-1},j\right) - a_k\left(i+2^{k-1},j\right) \\ a_k\left(i,j-2^{k-1}\right) - a_k\left(i,j+2^{k-1}\right) \end{vmatrix} \right) , \tag{4}$$

Confirm window dimension and acquire the maximum value k of $c_k(i, j)$. In general, k is distributed within the range from 0 to 2:

$$\hat{k}(i,j) = \arg\max_k c_k(i,j) \ , \tag{5}$$

Solve the average value of window dimension of all pixel points and then solve the roughness of document feature:

$$l_{coarse} = \frac{1}{wh} \sum_{i,j} 2^{\hat{k}(i,j)} . \tag{6}$$

In Equation (4), w is document width and h is document height. The roughness extraction process of Tamura texture is shown in Fig. 5 and the result value returned in that process is the calculated value of roughness indicator in document. In above-mentioned flow computing process, the window dimension of pixel point (i, j) is omitted to calculate this process.

Contrast indicator: the indicator is mainly used to measure light and shade degree shown in document and its computational process is:

$$Contrast = \frac{\sigma}{\sqrt[4]{\alpha_4}} . \tag{7}$$

$$\alpha_4 = \frac{\mu_4}{\sigma_4} = \frac{\frac{1}{wh}\sum_{i,j}\left(\mu(i,j) - \mu\right)^4}{\sigma_4} . \tag{8}$$

In above-mentioned equation, σ is the deviation in gray value of document, μ is

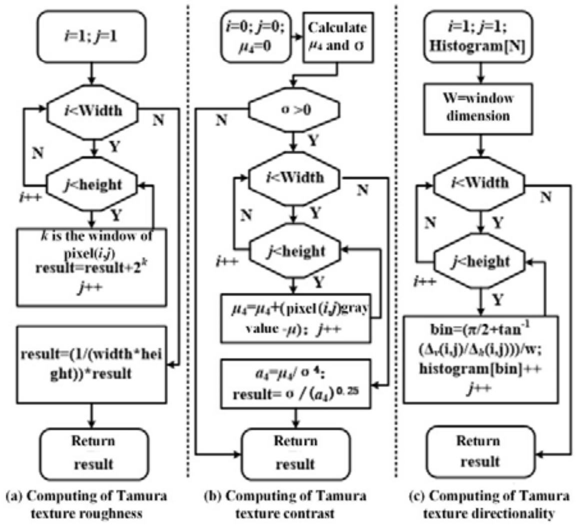

i=1; j=1
i<Width
N
Y
N
j<height
i++
Y
k is the window of pixel(i,j) result=result+2^k j++
result=(1/(width*height))*result
Return result
(a) Computing of Tamura texture roughness
i=0; j=0; μ4=0
Calculate μ4 and σ
σ>0
N
Y
i<Width
N
N
Y
j<height
i++
Y
μ4=μ4+(pixel(i,j)gray value -μ); j++
a4=μ4/σ^4; result=σ/(a4)^0.25
Return result
(b) Computing of Tamura texture contrast
i=1; j=1; Histogram[N]
W=window dimension
i<Width
N
N
Y
j<height
i++
Y
bin=(π/2+tan^-1(Δv(i,j)/Δh(i,j)))/w; histogram[bin]++ j++
Return result
(c) Computing of Tamura texture directionality

Fig. 5. Feature extraction process

the average gray value of document, $\mu(i,j)$ is the gray value of pixel point (i,j), μ_4 is the central moment of order-4 gray value, w is document width and h is document height. The contrast indicator acquisition process of Tamura texture is shown in Fig. 5 and feedback result in this process is the contrast indicator of document.

Directionality indicator: distribution and intensive feedback along texture in a certain direction and the computing form of direction gradient of pixel point (i,j) is:

$$g_{i,j} = \begin{pmatrix} \Delta_h(i,j) \\ \Delta_v(i,j) \end{pmatrix}. \tag{9}$$

Computing process of gradient $\Delta_h(i,j)$ in horizontal direction is: solve the deviation between gray value of 3 groups of pixel at the left pixel point (i,j) and gray value of 3 groups of pixel at the right pixel point (i,j), and then calculate gradient $\Delta_v(i,j)$ in vertical direction: solve the deviation between gray value of 3 groups of pixel at the upper pixel point (i,j) and gray value of 3 groups of pixel at the lower pixel point (i,j).

$$\begin{cases} \Delta_h(i,j) = \sum\limits_{k\in\{-1,0,1\}} p(i+1,j+k) - p(i-1,j+k), \\ \Delta_v(i,j) = \sum\limits_{k\in\{-1,0,1\}} p(i+k,j+1) - p(i+k,j-1). \end{cases} \tag{10}$$

The vector $g_{i,j}$ can be converted into polar form:

$$(|g_{i,j}|, \varphi_{i,j}) = \begin{pmatrix} \frac{|\Delta_h(i,j)|+|\Delta_v(i,j)|}{2}, \\ \tan^{-1}\left(\frac{\Delta_v(i,j)}{\Delta_h(i,j)}\right) + \frac{\pi}{2} \end{pmatrix}, \tag{11}$$

Mark a direct square strip $\varphi(k)$ in direction histogram, which meets:

$$\frac{2k-1}{2n} < \frac{\varphi_{i,j}}{n} < \frac{2k+1}{2n} \left(\text{mod}1\right) , \tag{12}$$

In case there are pixel points with the number conforming to $|g_{i,j}| > t$, then the histogram in this direction is equivalent to gradient of this document in horizontal direction. The process to acquire directionality for Tamura texture feature extraction is shown in Fig. 5.

3.3. Feature parallel storage process

Feature parallel storage process is designed in a group of MapReduce framework. This work is mainly to read document data stored in HDFS structure, extract feature texture of this document data and then store acquired feature data in HBase class library. For this purpose, ExtractMap is created and then realize it through map() mapping process. Above-mentioned computing flows of processing process are shown in Fig. 6. At the same time, no customization is carried out on Reduce and directly output map() mapping output through reduce() process based on WentityReducer but reduce() operation does not work.

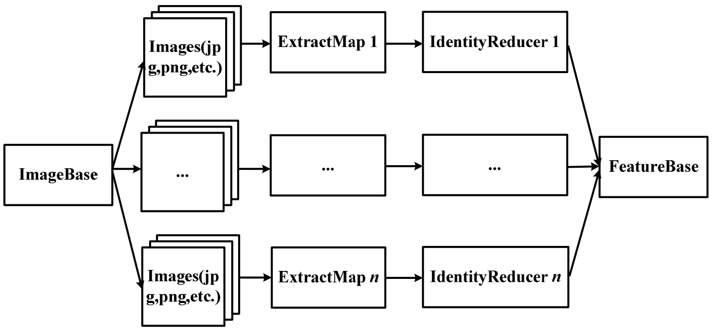


Fig. 6. Parallel mapreduce feature storage

After feature texture extraction of documents to be processed, execute the following process: in process control at client-side, Uploader is used for request of HDFS and then upload batches of local documents. After storage of texture feature, RunExtract object will request to create HTable object and then start ExtractMap operation process. And then, control and operate extraction and storage process of feature texture in above-mentioned processes.

Class relationship between feature texture extraction algorithms in designed documents is shown in Fig. 7.

In the figure, Client is mainly responsible for resources scheduling for above-mentioned system. But ExtractMap object is the core of algorithm computing and the process needs to be implemented according to feature texture extraction process. ImageFilelnputFormat is the input format of map() mapping process customized for design value to make it convenient for reading digital documents.

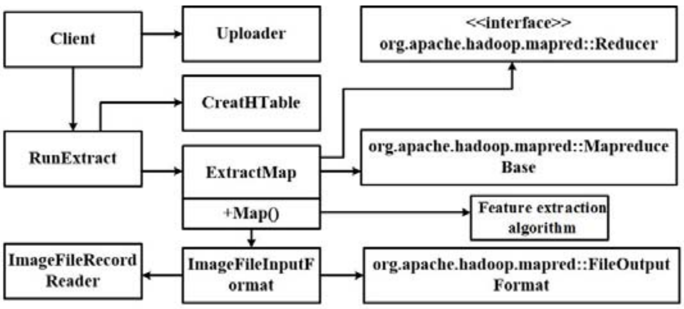

Client
Uploader
<<interface>> org.apache.hadoop.mapred::Reducer
CreatHTable
RunExtract
ExtractMap
+Map()
org.apache.hadoop.mapred::Mapreduce Base
Feature extraction algorithm
ImageFileRecord Reader
ImageFileInputFormat
org.apache.hadoop.mapred::FileOutput Format

Fig. 7. Feature texture extraction class relationship

4. Experimental analysis

Hardware system is: CPU is i3-4200U, RAM is 4G ddr3-1333 and hard disk is 500G Seagate. To verify effectiveness of index optimization process, carry out simulation in two parts corresponding to the design. Experiment in the first part reproduces the method of single-process Lucene index and the dataset in the Thesis is used to acquire a new experiment result. In specified node (such as 8 nodes), computing results of parallel MapReduce-Lucene index are used for comparison. At the same time, the algorithm in Literature [15] is used as the algorithm for lateral comparison; in the second part is aimed at computing results of above-mentioned parallel MapReduce-Lucene index so as to calculate change conditions of algorithm performance indicators in different nodes and different document libraries.

4.1. Fixed experimental subjects

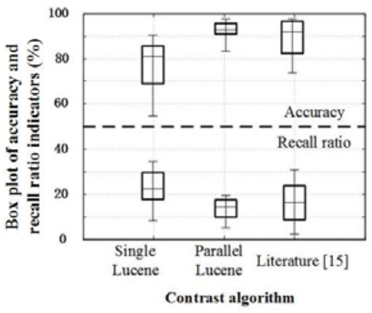
To make it convenient for verifying influences of above-mentioned index optimization process on computing efficiency of index, 1000 documents (82.4MB data) selected from Literature [14] are used for simulation test and the size of each document varies from 0.02M to 2M. Add index document and Chinese word segmentation module for module preprocessing and then create document index according to parallel framework of incremental Lucene system. 50 to 100 groups of features words will be used to substitute documents. Chinese word segmentation module is created by word segmentation API developed by Chinese Academy of Sciences. 10 groups of subject keywords are used to search the index. Computing nodes of algorithm operation are 8 and they are used for contrast experiment. First of all, carry out internal storage and read-in on documents by Lucene, create single index segment, combine index segments and then carry out write-in on disk (Map process). However, due to upper limit for the number of documents in index segment, the combination operation shall be carried out without restrictions. Contrast accuracy of contrast indicator selection index, recall ratio and operating time of algorithm. Experimental results are shown in Table 1.

Table 1 illustrates index of single-process Lucene, parallel MapReduce-Lucene index, computing accuracy of index process in Literature [15], recall ratio and running time index of algorithm. It is known from Table 1 that parallel MapReduce-Lucene index method has relatively higher inquiry accuracy, lower recall ratio and less computing time and above-mentioned experiment data reflects improvement of computing performance of designed algorithm.

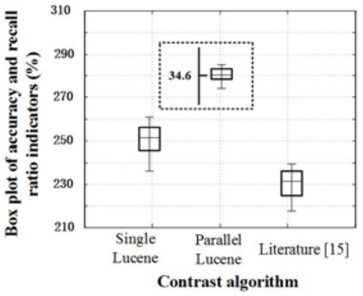
Table 1. Optimization of the index results (run 30 times)

Algorithm	Accuracy/%	Recall ratio/%	Time/S
Single Lucene	82.3%	23.4%	251.6
Parallel Lucene	96.2%	15.3%	34.6
Literature [15]	91.4%	16.8%	231.4

Table 1 shows average value indicator of three contrast algorithms after 30 times of operation and it is the consideration of algorithm stability. Fig. 1 has illustrated indication distribution figure of accuracy, recall ratio and running time index of algorithm in above-mentioned 30 times of operation.



(a) Accuracy and recall ratios



(b) Running time index contrast

Fig. 8. Index distribution contrast

Fig. 8a is the distribution figure of accuracy and recall ratio indexes and Fig. 8b is the distribution figure of running time. It is known from Fig. 8a that MapReduce-Lucene index method in the Thesis has relatively lower index distribution figure and it is known from this that the stability of MapReduce-Lucene index method is higher. In running time shown in Fig. 8b, index distribution figure of MapReduce-Lucene index method is relatively lower, which reflects stability of the algorithm in the same way.

4.2. Algorithm adaptability

According to index computing results of proposed parallel MapReduce-Lucene, calculate change conditions of algorithm performance indicators in different nodes and different setting conditions of document library. First of all, set variation range of document library scale: 102 to 106documents. computing nodes are set to be 8. Algorithm contrast effects are shown in Fig. 9.

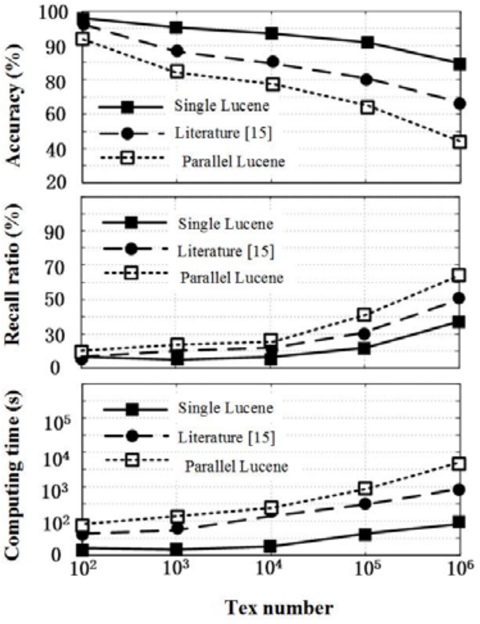


Fig. 9. Performance metrics for different document number algorithms

According to experimental results shown in Fig. 9, in accuracy and recall ratio indexes and in different document numbers, the algorithm in the Thesis is always superior to the algorithm of single-process Lucene index and the index algorithm in Literature [15]. In calculating time index and in 8 given computing nodes, when the document number is 10^4, node computing resources are saturated.

Given computing nodes are respectively 4, 8 and 16 and the text number is 10^4 and the variation conditions of the algorithm in accuracy, recall ratio and algorithm running time indexes are shown in Fig. 10.

Fig. 10 illustrates the algorithm performance indicators in different numbers of computing nodes. It is known from simulation data in the figure that in 4 to 8 variation processes in computing stage, promotion in performance indicators of the algorithm is relatively obvious; in variation process of 8 to 16 computing nodes, variation in algorithm performance is not too obvious, which shows the number of nodes at this time is saturated for 10^4 of text number.

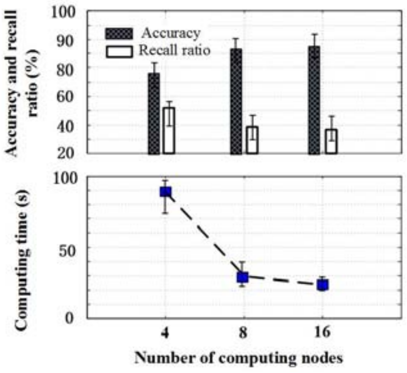


Fig. 10. The number of different computing nodes

5. Conclusion

To improve retrieval efficiency of Lucene in big data, incremental full-text Lucene engine parallel retrieval method based on MapReduce is proposed in the Thesis. First of all, research is carried out on Lucene index principle structure, improve and increase index capacity in buffer zone and reduce index writing frequency of disk files based on incremental index and realize improvement in index creation efficiency; secondly, since index efficiency promotion of single-process Lucene is limited and based on parallel MapReduce computing module in Hadoop framework, parallel Lucene full-text retrieval engine is created to realize parallelized execution of big data retrieval.

Acknowledgement

The Sci.&Tec. Support Plan of Sichuan Province "Research on Key Technology of Big Data" under the grant No. 2014GZ0013.

References

[1] Dong J, Krzyzak A, Suen C Y: (2005) *Fast SVM Training Algorithm with Decomposition on Very Large Data Sets*[J]. IEEE Transactions on Pattern Analysis & Machine Intelligence, 27(4):603-618.

[2] Liu R, Wang Y, Baba T, et al.: (2007) *SVM-based active feedback in image retrieval using clustering and unlabeled data*[J]. Pattern Recognition, 41(8):2645-2655.

[3] Li X, Wang N, Li S Y: (2007) *A Fast Training Algorithm for SVM Via Clustering Technique and Gabriel Graph*[C]// Advanced Intelligent Computing Theories and Applications. With Aspects of Contemporary Intelligent Computing Techniques, Third

International Conference on Intelligent Computing, ICIC 2007, Qingdao, China, August 21-24, 2007. Proceedings. DBLP, 2007:403-412.

[4] Harvey R W: (2011) *Theoretical and experimental comparison of different approaches for color texture classification*[J]. Journal of Electronic Imaging, 20(4):49-59.

[5] Patra S, Bruzzone L: (2014) *A Novel SOM-SVM-Based Active Learning Technique for Remote Sensing Image Classification*[J]. IEEE Transactions on Geoscience & Remote Sensing, 52(11):6899-6910.

[6] Malvadkar S D, Waskar S R, Patil N K, et al.: (2015) *A Fast Clustering Based FSS Algorithm for High Dimentional Data using SVM*[C]// International Journal of Engineering Research and Technology. ESRSA Publications.

[7] Qiao P L, Chen S F: (2009) *Intrusion Detection System Technique Based on BP-SVM*[C]// International Conference on Management and Service Science. IEEE, 2009:1-3.

[8] Qi Y L, He W, Shu H: (2008) *An Optimized Approach on Reduced Kernel Matrix to ClusterSVM*[C]// International Conference on Intelligent Information Hiding and Multimedia Signal Processing. DBLP, 2008:1446-1449.

[9] Anyanwu L O, Keengwe J, Arome G A: (2010) *Scalable Intrusion Detection with Recurrent Neural Networks*[C]// Seventh International Conference on Information Technology: New Generations. IEEE, 2010:919-923.

[10] Huang T S, Dagli C K, Rajaram S, et al.: (2008) *Active Learning for Interactive Multimedia Retrieval*[J]. Proceedings of the IEEE, 96(4):648-667.

[11] Khan M M, Ahmad A M, Khan G M, et al.: (2013) *Fast learning neural networks using Cartesian genetic programming*[J]. Neurocomputing, 121(18):274-289.

[12] Shi W, Zhu Y, Huang T, et al.: (2016) *An Integrated Data Preprocessing Framework Based on Apache Spark for Fault Diagnosis of Power Grid Equipment*[J]. Journal of Signal Processing Systems, 2016:1-16.

[13] Wang X: (2014) *Aero-engine Gas Path Fault Diagnosis Based on Improved Support Vector Machine and Synergetic Neural Network*[J]. Journal of Information & Computational Science, 11(10):3533-3542.

Received May 7, 2017