Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2013, Article ID 684615, 7 pages
http://dx.doi.org/10.1155/2013/684615

Research Article

Content-Based Image Retrial Based on Hadoop

DongSheng Yin and DeBo Liu

School of Information Science and Technology, Sun Yat-sen University, Guangzhou 510000, China

Correspondence should be addressed to DongSheng Yin; yindsh@mail.sysu.edu.cn

Received 9 July 2013; Accepted 22 August 2013

Academic Editor: Ming Li

Copyright © 2013 D. Yin and D. Liu. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Generally, time complexity of algorithms for content-based image retrial is extremely high. In order to retrieve images on large-scale databases efficiently, a new way for retrieving based on Hadoop distributed framework is proposed. Firstly, a database of images features is built by using Speeded Up Robust Features algorithm and Locality-Sensitive Hashing and then perform the search on Hadoop platform in a parallel way specially designed. Considerable experimental results show that it is able to retrieve images based on content on large-scale cluster and image sets effectively.

1. Introduction

Content-based image retrieval (CBIR) is a long-term hotspot in computer vision and information retrieval and there are many mature theories on this topic. For example, an algorithm proposed in early time by using multiresolution wavelet decompositions [1] had achieved favorable results in searching images that are similar in content or structure. And scale invariant feature transform (SIFT) [2], published in 1999, was able to be stable with light, noise, and small perspective change in a sense. But SIFT was so complex that it cost too much time and this led to several other improvements, such as principle component analysis SIFT (PAC-SIFT) [3] which speed up feature matching by reducing the dimension of image features and fast approximated SIFT [4, 5] which speed up by using an integral image and an integral orientation histogram. Another considerable algorithm on CBIR is speededup robust features (SURF) [6] and it is stable and fast enough to gain excellent results on region of computer vision such as object recognition and 3D reconstruction. SURF is also involved from SIFT but faster and announced to be more robust than SIFT when coming to image transformation.

Normally, CBIR has two steps, the feature extraction which mainly affects the quality of searching, and the feature matching, which mainly affects the efficiency. Usually, features are in high dimension, so matching features means searching in high-dimension. There are many ways to search high dimension space such as linear scanning, tree searching, vector quantization, and hashing. Among these methods, hashing is the easiest way to keep time complexity O(1) as well as designing as a fuzzy search method. Details about hashing will be discussed later.

Even features matching is optimized via hashing; due to the huge amount of information of CBIR, time complexity is still too high, preventing it from being widely used. Particularly in the age of explosive expansion in information, the stand-alone method for CBIR is becoming harder and harder to fulfill the load of storage and computing brought by data explosion. Hadoop [7] is an open-source software framework for reliable, scalable, distributed computing. It enables large datasets processing distributedly across clusters using simple programming models. It is widely used by IT companies like Yahoo!.

In this paper, image features extraction and matching are combined with three techniques, SURF, LSH, and Hadoop distributed platform, intending to migrate computing to cluster with multiple nodes and improve the efficiency of CBIR significantly. The rest of the paper is organized as follows. Section 2 discusses related algorithms and techniques. Architecture of implementation of such method is discussed in Section 3. Section 4 presents our experimental results and analysis. Finally, the conclusions are drawn in Section 5, with a brief description of future work.

2. CBIR Algorithms and Hadoop Introduction

2.1. Speeded-Up Robust Features. The SURF algorithm is divided into two stages: the interest points' detection and feature description. In the first stage, integral images and fast Hessian matrix is used for detection of image features. In the second stage, a reproducible orientation of each interest point is fixed first, then constructing a square region aligned with the selected orientation and extracting the SURF descriptor with 64 dimensions from this region. Steps are as follow [6].

(1) *Scale Spaces Analysis*. Image pyramids are built by repeatedly Gaussian blur and subsampling.

(2) *Interest Points Locating*. The maximum of the determinant of the Hessian matrix is calculated first. Then, nonmaximum suppression in a $3 \times 3 \times 3$ neighborhood to the image is applied, scale and image space interpolation are taken.

(3) *Orientation Assignment*. Haar-wavelet responses in x and y direction in a circular neighborhood around the interest point is calculated. A dominant orientation is estimated and it is now invariant to rotation.

(4) *Descriptor Extraction*. A square region centered around the interest point is constructed and oriented along the orientation selected. Then, the region is split into 4×4 square subregions and the sum of wavelet response in horizontal and vertical direction of each subregion is calculated. After normalization, a descriptor in 64 dimensions is obtained.

SURF is improved from SIFT both are based on robust points (or interest points) which are not sensitive to transformation, brightness, and noise but is less complex and more efficient than SIFT due to the smaller number and lower dimension of descriptors. An open-source library OpenSURF (http://www.mathworks.com/matlabcentral/fileexchange/28300) written by Dirk-Jan Kroom for SURF descriptor extraction is used in this paper.

2.2. Locality Sensitive Hashing. It is mentioned previously that SURF descriptors are in 64 dimensions and matching features means searching in high dimension. Among the four regularly searching methods, locality sensitive hashing (LSH) [8] is the fastest way for indexing and could be faster than other three methods for several orders of magnitude in huge-scale searching. In addition, LSH is based on probability and is more suitable for nonprecise searching. LSH was first introduced by Indyk and Motwani for nearest neighbor search [9]. The main idea is to hash vectors using several hash functions and make sure that for each hashing, the vectors with smaller distances between each other are more likely to collide in probability than that with longer distances. Different hash functions could be designed for different metrics such as Euler distance.

A family of functions can be defined as follows [10].

A family $\mathcal{H} = \{h : s \to U\}$ is said to be locality sensitive if, for any q, function $p(t) = Pr_{\mathcal{H}}[h(q) = h(v) : q - v = t]$ decreases strictly as t increases. That is, the probability of the collision of q and v decreases as the distance between them increases.

Stable distribution is one of the most important methods for LSH function implementation. And Gaussian distribution, one kind of famous stable distribution, is used for LSH function design frequently.

Given k, L, w, suppose that A is an $k \times d$ Gaussian matrix, A_i represents the i row of A, $b \in \mathfrak{R}^k$ is a random vector, and $b_i \in [w]$, $x \in \mathfrak{R}^d$; then the hash code of x can be represented as

$$g(x) = (h_1(x), \dots, h_k(x)), \tag{1}$$

$$h_i(x) = \frac{A_i x + b_i}{w}, \quad i \in [k], \tag{2}$$

$g(x)$ is the concatenation of k hash codes, and it is regularly designed as normal hash function to obtain the final scalar index, such as the following one recommended by Andoni and Indyk in one of his LSH library [10]:

$$\begin{aligned} g(x) &= f(a_1, \dots, a_k) \\ &= \left(\left(\sum_{i=1}^{k} r_i' a_i \right) \bmod \; prime \right) \bmod \; tableSize, \end{aligned} \tag{3}$$

in which r_i' is a random integer, *prime* equals $(2^{32} - 5)$, *tableSize* represents the size of hash table, usually equals to $|P|$, the size of searching space.

b_i in (2) is a random factor, and because it can be noticed that A itself is random already, so just set $b_i = 0$. Denominator w in (2) represents a segment mapping such that the similar values in numerator can be hashed into the same code for the purpose of neighbor searching and its value represents the segment size. In this paper, we chose $w = 0.125$.

There are two more parameters that should be determined; k for the number of hash codes should be calculated by each hash function and L for the number of hash functions. From the fact that two similar vectors will collide with the probability greater than or equal to $(1 - \delta)$ when applying LSH, we get some conditions that k and L should satisfy.

Suppose that the distance of a query q and its neighbor v is less than a constant R, and let $p_R = p(R)$; then

$$Pr_{g \in \mathcal{G}}[g(q) = g(v)] \geq p_R^k. \tag{4}$$

And for all L hash tables, the probability that q and v does not collide is no more than $(1 - p_R^k)^L$; that is

$$1 - (1 - p_R^k)^L \geq 1 - \delta. \tag{5}$$

We get better performance if there are less hash tables. So let L be the minimum possible integer and there is

$$L = \text{floor} \frac{\log \delta}{\log(1 - p_R^k)}. \tag{6}$$

Now L is a function of k.

According to Andoni and Indyk [10], the best value of k or L should be tested by sampling. Experimental results of Corel1K image set testing show that k prefer 5, and let the value of L be 7 from (6).

2.3. Hadoop. Hadoop is mainly compose of Hadoop distributed file system (HDFS), MapReduce, and HBase. HDFS is a distributed file system using by Hadoop while HBase is a distributed NoSQL database. And MapReduce is a kind of simple but powerful programming model for parallelly processing large dataset. The operation of MapReduce contains two steps: the map step that outputting ⟨key, value⟩ pairs after processing the input data and the reduce step that collecting and processing the ⟨key, value⟩ pairs coming from the map step with the same key. Figure 1 shows how MapReduce works.

3. System Design

3.1. Overall Design. The overall design of this system is as in Figure 2. Among all the modules, Feature Extraction and Feature Matching are the most time consuming. And Matlab is used as auxiliary because there are too many image processing and matrix operations. The workflows are as follow.

(1) *Image preprocessing*, including image scaling and graying (notice that SURF is based on gray images and is nonsensitive with image scaling).

(2) *SURF extraction*: multiple vectors in 64 dimensions are obtained.

(3) *Hashing*: hashing each feature from last step using (2) and (3), and 7 hash codes are obtained.

(4) *Feature matching*: for each hash codes from last step, search corresponding hash table for match features using MapReduce then results are collected and sorted.

(5) Output results.

3.2. Parallelization Design for Feature Matching. In this module, candidate matching features for each feature of input image are searched and candidate similar images are selected according to matching counts. For simplicity, two features are considered to be similar if their hash codes collide. In this step, the input is the feature set and several hash codes for each feature, and the output is the list of candidate similar images. This is the most time-consuming part of the whole system and is implemented by MapReduce.

Suppose $\langle K, V\rangle$ represents Key-Value pair in MapReduce; then workflow of query in parallel is shown as Figure 3.

The feasibility of this parallelization is based on two facts.

(1) All splits are pairwise independent. That is, there are no relationships between any two splits. The format of features description is the same as the value of mapper input. Each line contains information about a feature's hash codes and image id to which it belongs and features are independent. So splits based on line break can be processed independently and concurrently.

(2) Results from all mappers will be collected by reducer. In addition, the number of reducers is set to one; thus, all parallel processing output will be counted and sorted in single reducer. Eventually, retrial results are unrelated to the way the descriptions split.

Suppose that N represents set of n job servers, N_i represents the ith job server, and t_i equals to the time N_i finishes its task; then the total time for the whole cluster to finish its searching assignment is

$$T = \max\{t_1, \ldots, t_n\}. \tag{7}$$

As the cluster is designed to be isomorphic, so in the case of overall task remains constant, the minimum of total time should be

$$T_{\min} = t_1 = \cdots = t_n; \tag{8}$$

that is, it will take the cluster the least time as long as the sizes of all splits are the same. What should be pointed out here is that, due to the independence of each split, actually t_i is related to N_i and its real task. So, only the elements that could be controlled are discussed here.

If the size of description file before splitting is Size (MB), the split size is seg (MB), then

$$\text{seg} = \frac{\text{Size}}{n} > 64?64 : \frac{\text{Size}}{n}. \tag{9}$$

Meaning that the task is averagely split first, and every job server gets a split with the size of seg $=$ Size$/n$. At this time, the task is balanced in all nodes, and T will get its minimum value. If seg is larger than 64 (MB), which is HDFS default block size, it may cost extra effort due to cross block accessing. At this time

$$\text{Size} = k * 64 + \text{tail}, \tag{10}$$

where k is integer and tail is less than 64 MB. That is, a description file with size Size is split into $k+1$ parts, with k parts size 64 MB, one part size tail. These $k+1$ parts then will be assigned to job servers by Hadoop job tracker. In later section, it can be seen that such segmentation strategy which combines all compute resource and features of Hadoop framework into thinking is simple and highly efficient.

3.3. Data Structure. There are three kinds of data in the system: the image set, the features, and the hash tables. The image set is stored in the OS file system and the other two are stored in HBase built on HDFS, the Hadoop file system. HBase does not support SQL query, so primary keys or range of primary keys are needed. Details are as follow.

(1) *Image Set*. Images are stored as normal image files such bmp files in the local file system, named start from 1 incrementally.

(2) *SURF Features Table*. Normally, there are hundreds of features for each image, so invert index is better for features storage. The pattern is in this form:

$$\left(\mathit{ImageId\_i}_{\text{key}}, \mathit{Float}, \mathit{Float}, \ldots\right) \tag{11}$$

ImageId represents the identifier of an image in the local file system while i represents the ith features of image *ImageId*. Followings are 64 floats, representing a vector in 64 dimensions.

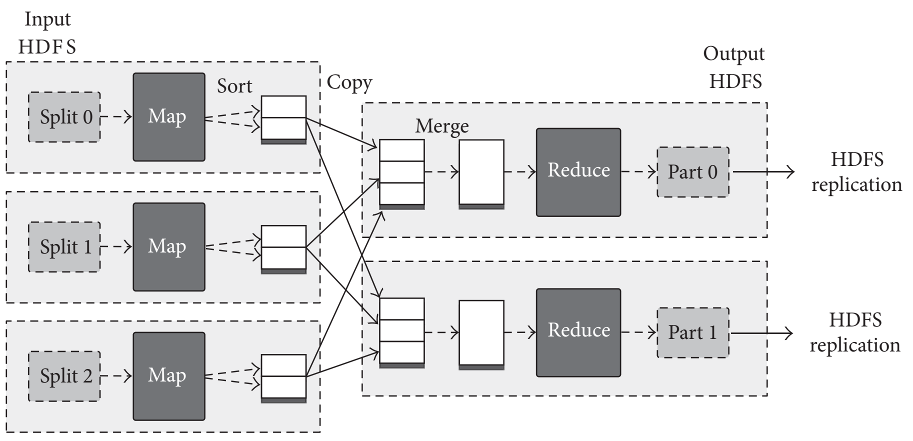


Figure 1: The MapReduce programming model.

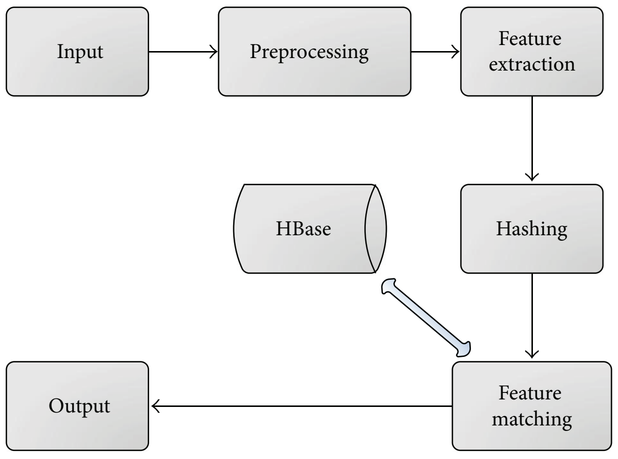


Figure 2: Overall design. Preprocessing, Feature Extraction and Hashing are mainly matrix operations and are implemented as Matlab scripts for the purpose of speeding up and convenience.

(3) *Hash Table*. The hash table is used for neighbor searching. That is, select a hash code of a specific feature as key and query the corresponding value. There are 7 hash tables according to LSH in this paper, so the storage form is similar to SURF features table for the purpose of reducing database connections. That is,

$$\left(\text{Hashcode\_}i_{\text{key}}, \textit{ImageId\_}j_1, \ldots, \textit{ImageId\_}j_k\right) \quad (12)$$

Hashcode represents a hash value while i represents the ith hash function and $i = 1, \ldots, 7$; for example, 55555_3 represents that the hash value of the third hash function is 55555. And $\textit{ImageId\_}j_k$ is a key in SURF features table.

4. Experiment and Analysis

4.1. Accuracy. Precision and recall are the main criteria for evaluating content-based image retrieval algorithms. Because the primary focus of this system is the design and implementation of distributed computing and for fuzzy searching in large-scale image database, recall is less meaningful, so only the precision is selected and tested.

As mentioned previously, SURF is robust to revolution and small change of perspective, so Corel image database is used for fully testing the precision of the system. Corel image database contains 10 kinds of images, each of 100 images, a total of 1000 images, including human, landscape, and architecture.

Results show that the percentage of the 6 returning results that are similar to the input image is 60.4%, as shown in Figure 4.

Here, two images are considered similar to each other if at least 30 pairs of features are matched, and the more they match, the more similar those two images are. As in Table 1, there are 5 outputs for image with ID "118." Among them, the most similar one is "159" except "118" itself, with 86 matches in total 267 features.

In addition, CBIR algorithms are related to specific image database in a considerable sense, so the result in the paper is just for reference.

Table 1: Example for feature matching.

Input	Outputs	Matches	Feature number	Percentage
118	118	256	267	99.25
	159	86	267	32.21
	472	79	267	29.59
	127	70	267	26.22
	199	70	267	26.22

4.2. Experimental Environment. The topology of the system deploying environment is as in Figure 5. The physical machines are listed in Table 2. The configurations of virtual machines are listed in Table 3 and the software environments are listed in Table 4. As shown, there are 11 physical machines, containing 1 master and 30 slaves and HBase Region Servers which are all virtual machines.

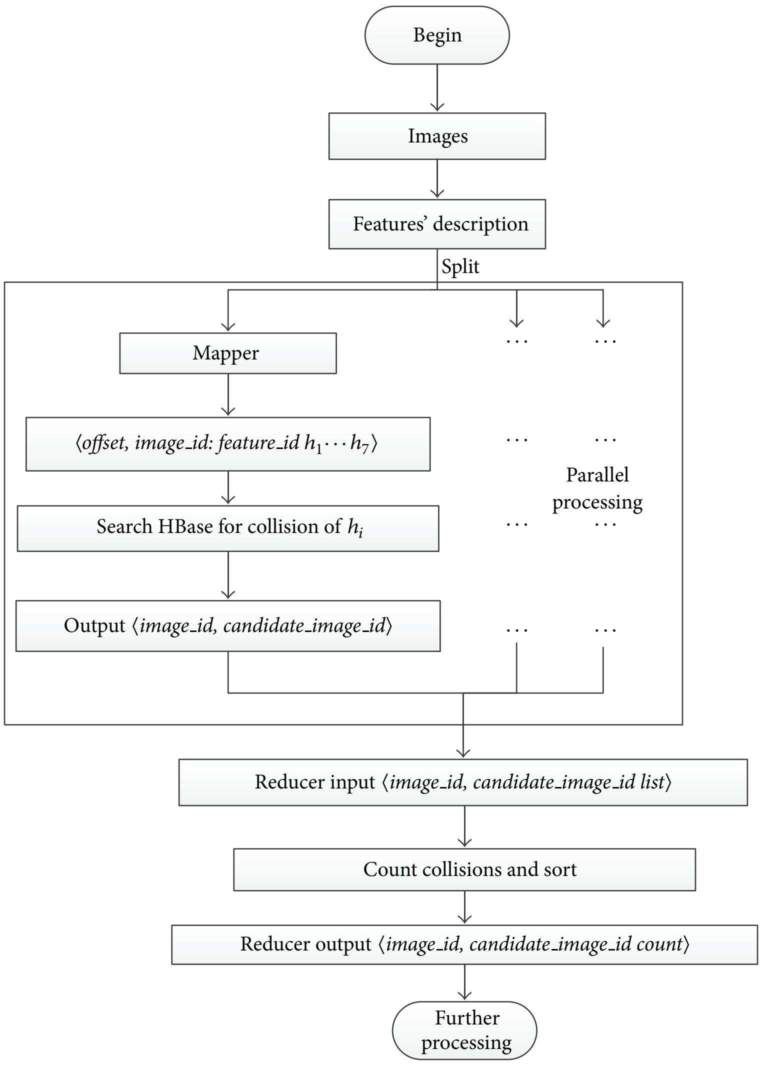

Begin
Images
Features' description
Split
Mapper
⟨offset, image_id: feature_id $h_1 \cdots h_7$⟩
Search HBase for collision of h_i
Output ⟨image_id, candidate_image_id⟩
Parallel processing
Reducer input ⟨image_id, candidate_image_id list⟩
Count collisions and sort
Reducer output ⟨image_id, candidate_image_id count⟩
Further processing

Figure 3: Parallel processing model.

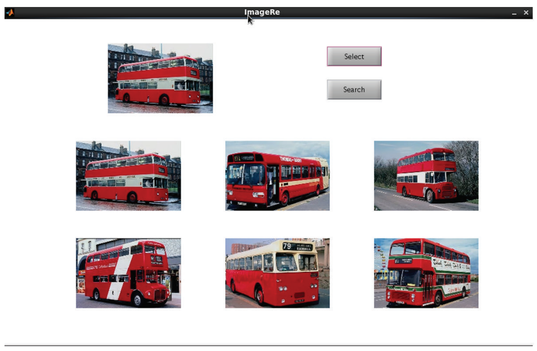


Figure 4: Typically, 6 images that are most similar to the input one will be shown.

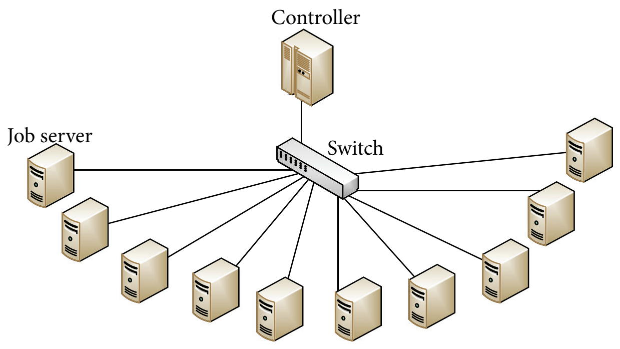


Figure 5: Topology of system deploying environment. 11 servers are connect with a switch.

4.3. Results and Analysis. In the system, 30 million features of 159955 images are recorded. The size of data, including scaled images, features, and hash codes, is up to 9.93 GB.

Efficiency testing is divided into two parts: the feature extraction and feature matching. Feature extraction is tested in single node as its algorithm is not distributed and feature

Table 2: Physical machines configurations.

Type	CPU	Memory	Number
Controller	Intel Core i7 2600 4 × 3.4 GHZ	16 GB	1
Job Server	Intel Xeon E3-1235 4 × 3.2 GHZ	32 GB	10

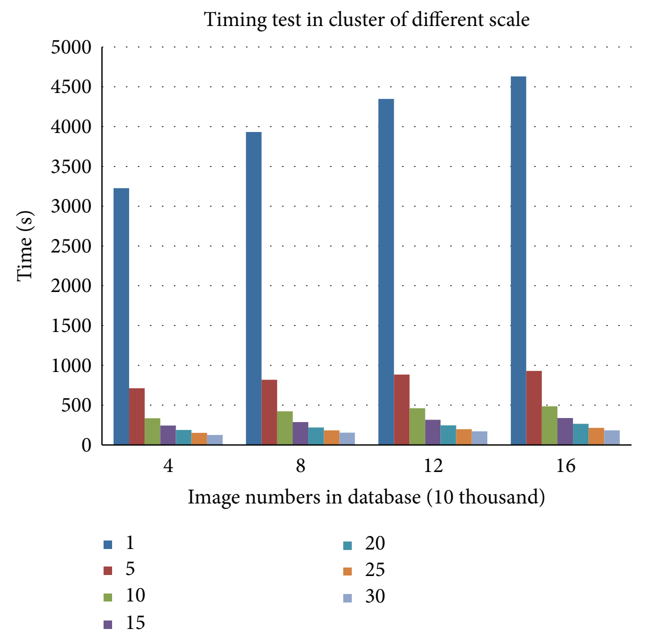


Figure 6: Time spent for feature matching. Legends from 1 to 30 indicate the number of nodes in a cluster.

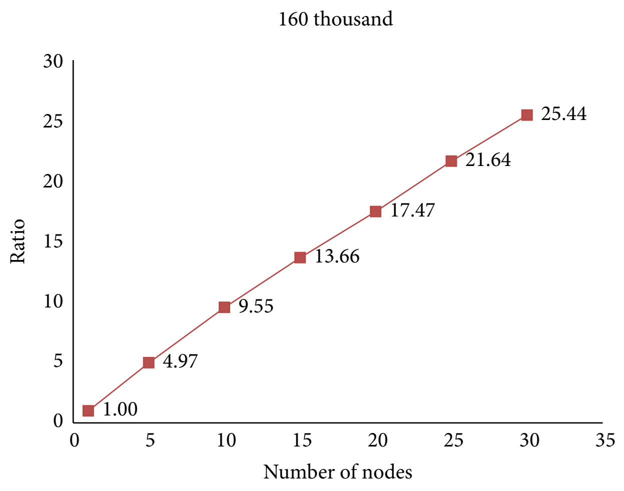


Figure 7: Ratio of performance improvement with 160 thousand images in database.

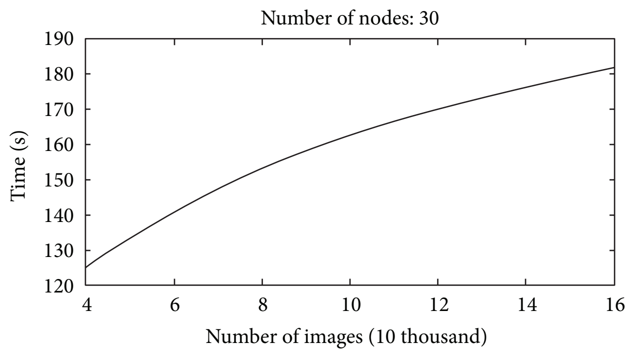


Figure 8: Time spent increases logarithmically as data in database growths when there are 30 nodes running in a cluster.

matching is tested in cluster of different scale (single node, five nodes to thirty nodes increasing by 5 nodes a time), respectively.

Extracting the features of all the 159955 JPEG images with the resolution less than or equal to 256∗256 in the Controller takes about 180 minutes, in speed of about 14.8 images per second. Extracted features will be written into database, so this extraction operation only has to be done for one time.

Results of feature matching of 31993 input images whose features have been extracted are shown in Table 5 and Figure 6.

Table 5 shows that, when there are 40 thousand images in the database, it costs 3227 seconds for single node to accomplish the job while only 125 seconds for thirty nodes and the ratio is almost 25.8. Other scales of database are similar. In addition, with 160 thousand images in database, the system can match features as fast as 0.006 second per image.

Figure 6 shows that, in a specific database, consuming time is reducing linearly as the number of nodes increases. That is, the performance of the system is increasing linearly as the number of nodes increases.

Another view of the results shown in Table 5 is in Figure 7. The abscissa represents the nodes of a cluster, and the ordinate is the ratio of cost time in single node and in corresponding scale of cluster. For example, 9.55 means that the time cost in single node is 9.55 times of that in cluster of 10 nodes. From Figure 7, it can be confirmed that, in a specific database, the performance of the system increases almost linearly as the number of nodes increases which indicates that the system is able to accomplish the jobs distributedly by calling all the nodes in the cluster efficiently and fully prove the system's availability and excellent expansibility in distributed environment. Other cases are similar.

Time consumed for different databases in cluster of 30 nodes is shown in Figure 8. It can be seen that the time cost is increased logarithmically as database growths, indicating that the system has strong suitability and advantage in performance for large-scale database. Other cases are similar.

To conclude, especially from the analysis of Figures 6–8, it can be seen that the system has two major advantages.

(1) It can be expanded quite easily as the performance of the system is increasing linearly with the increasing in the number of nodes. Adding nodes means speeding up.

Table 3: Virtual machines configurations.

Role	Belongs to	Frequency	Memory	Number	Network
Master	Controller	5.0 GHz	8 GB	1	1 Gbps Switch
Slave	Job Server	3.0 GHz	4 GB	3 × 10	

Table 4: Virtual machines software environment.

OS	Hadoop	HBase	JDK
Centos 6.3	Hadoop-1.0.1	HBase-0.92.1	1.6.0_22

Table 5: Time (second) spent for feature matching.

Nodes	10 thousand				
	4	8	12	16	Average
1	3227	3933	4347	4630	4034.25
5	713	819	883	931	836.5
10	335	423	461	485	426
15	243	287	317	339	296.5
20	188	220	247	265	230
25	152	183	198	214	186.75
30	125	153	171	182	157.75
1 : 30	25.82	25.71	25.42	25.44	25.60

(2) Time cost is increased logarithmically as database grows. So, it will perform better against larger database.

The system designed is able to take full advantage of distributed architecture, making the searching rate increased almost linearly as the number of nodes grows to achieve the goal of fast content-based image retrial.

5. Conclusion

In this paper, a method for content-based image retrial under distributed framework is proposed and discussed from theory and implementation. Experimental results show that it is able to retrieve images based on content on large-scale image sets effectively. Distributed framework and large scale of data are the main focus of the system. It is rather significant to solve the severe problems of huge amount of computing and storage by combining the traditional CBIR with distributed computing to gain higher efficiency. In addition, the system is remaining to be improved such as the following:

(1) *Make Improvement on CBIR Algorithms*. CBIR is a complexity technique. So, in the context of getting an acceptable result, only one algorithm is used to simplify. Other image processing techniques could be applied to improve both the precision and performance. Techniques such as the affine invariant features extraction method in [11, 12] which can be used for object classification both in database building and parallel searching stages may speed up the whole process by indexing images to different category.

(2) *System Optimization and Real-Time Enhancement*. Optimization can be done by optimizing Hadoop and HBase. But as it takes a long time for Hadoop jobs to start up and be scheduled, it is not suitable for real-time processing. Other distributed computing model such as Twitter Storm (http://storm-project.net/) stream computing can be considered.

References

[1] W. Niblack, R. Barber, W. Equitz et al., "QBIC project: querying images by content, using color, texture, and shape," in *Storage and Retrieval for Image and Video Databases*, pp. 173–187, February 1993.

[2] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004.

[3] Y. Ke and R. Sukthankar, "PCA-SIFT: a more distinctive representation for local image descriptors," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR '04)*, pp. II506–II513, July 2004.

[4] M. Grabner, H. Grabner, and H. Bischof, "Fast Approximated SIFT," in *Proceedings of the Asian Conference on Computer Vision*, pp. 918–927, Hyderabad, India, 2006.

[5] B. Smith, "An approach to graphs of linear forms," unpublished.

[6] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-Up Robust Features (SURF)," *Computer Vision and Image Understanding*, vol. 110, no. 3, pp. 346–359, 2008.

[7] T. White, *Hadoop: The Definitive Guide*, O'Reilly Media, 2009.

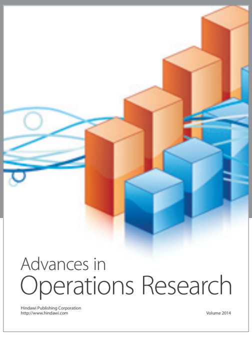
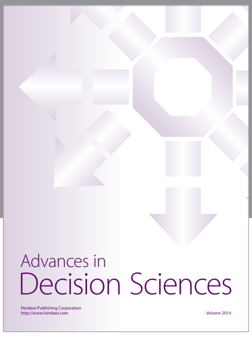
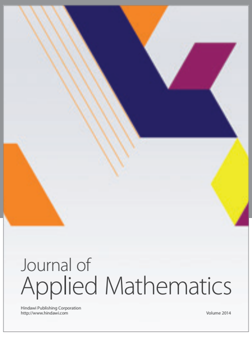
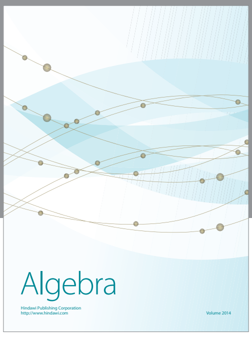
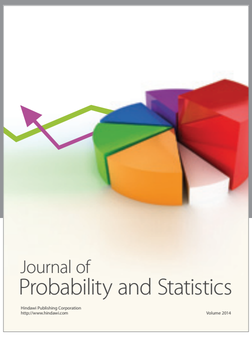
[8] A. Gionis, P. Indyk, and R. Motwani, "Similarity search in high dimension via hashing," in *Proceedings of the International Conference on Very Large Databases*, 1999.

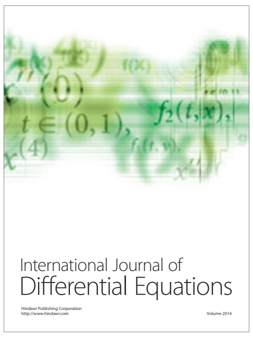
[9] P. Indyk and R. Motwani, "Approximate nearest neighbors: towards removing the curse of dimensionality," in *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, pp. 604–613, Dallas, Tex, USA, May 1998.

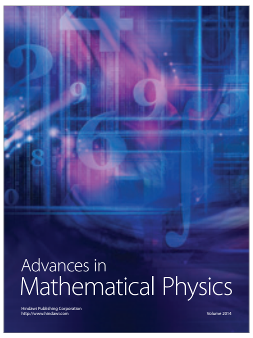
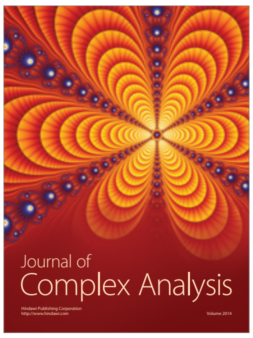
[10] A. Andoni and P. Indyk, "E2LSH_0. 1 User Manual," June 2005.

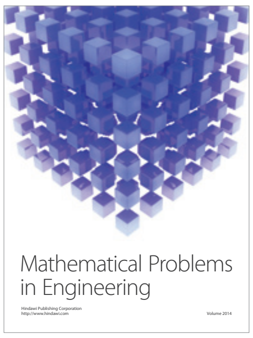
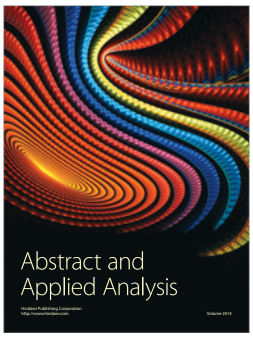
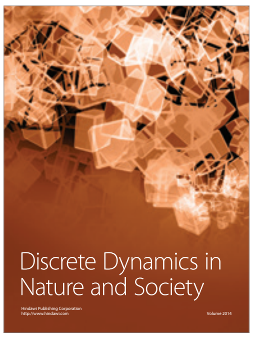
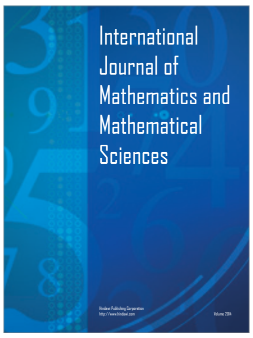
[11] J. Yang, M. Li, Z. Chen, and Y. Chen, "Cutting affine moment invariants," *Mathematical Problems in Engineering*, vol. 2012, Article ID 928161, 12 pages, 2012.

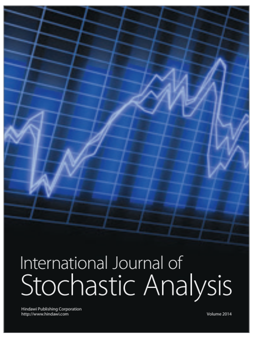
[12] J. Yang, G. Chen, and M. Li, "Extraction of affine invariant features using fractal," *Advances in Mathematical Physics*, vol. 2013, Article ID 950289, 8 pages, 2013.





Hindawi

Submit your manuscripts at
http://www.hindawi.com