Communications on Applied Electronics (CAE) – ISSN : 2394-4714
Foundation of Computer Science FCS, New York, USA
Volume 4– No.7, March 2016 – www.caeaccess.org

Effective and Faster Retrieval of Images from Large Database by using Binary Tree Implemented with Map Reduce

Radhakrishnan B.
Assistant Professor
Baselios Mathews II College of Engineering,
Sasthamcotta
Kollam, Kerala

Anver Muhammed K.M.
Systam Administrator
Baselios Mathews II College of Engineering,
Sasthamcotta
Kollam, Kerala

ABSTRACT
Effective searching of image from large image data base is definitely a tedious task. Searching images linearly will cost a lot of time. A distributed approach using map reduce concept is proposed in this paper. Rather than comparing two images, similarity features between images are searched for. The features are stored in different machines which are implemented using two dimensional binary tree. The tree constitutes the root and leaf machine which des the necessitated search.

General Terms
Large scale image searching, Big Data, Feature Detection, Map

Keywords
CBIR, Map Reduce, Feature vectors.

1. INTRODUCTION
The future image databases will abandon the matching paradigm of images , and rely on similarity searches. For looking for similarity, there is no need to look for the existence of a target image in the database. Rather, images are searched with respect to similarity with the query, given a fixed similarity criterion. Content-Base Image Retrieval (CBIR) has been proposed in the early 1990's.

Visual features are used in CBIR systems to represent the image content. The information used during retrieval process should have to be consistent, so CBIR systems are favorable since the features are computed automatically. To search for something similar, the query image is provided, or selects a prototype image. The result is a list of images sorted in decreasing values of similarity to the query image. Comparing two images in the big image data is time consuming and tedious task. So to reduce the complexity it is practical to measure the similarity using low level image properties, ie features in an image. This goal is usually performed using index structures on the image content descriptors. The feature vector of each image is stored and indexed in the data base. So that at query time the feature vector of the query image is computed and searched for the most similar feature in the data base using the distance functions.

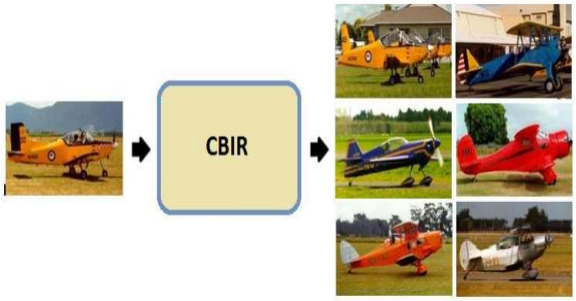

Fig 1. Result of query image in CBIR.

2. FEATURE EXTRACTION, SIMILARITY MODELS AND QUERY PROCESSING.
The features should be "simple enough" to allow the design of automatic extraction algorithms, yet "meaningful enough" to capture the image content. Several CBIR systems used the *global features*, like color and texture, which possess a rich semantic value. Under this view, each image is typically represented by a high-dimensional *feature vector*, whose dimensionality depends on the number and on the type of extracted features, and similarity between images is assessed by defining a suitable distance function on the resulting feature space.

2.1 Image Retrieval by Color Representation
In an image, Histogram is usually used to represent the distribution of colors. Each pixel of an image I[*x*, *y*] constitutes of three *color channels* $I = (I_r, I_g, I_b)$, which represents red, green, and blue components respectively. Using a transformation matrix T_r these channels are transformed, into hue, brightness, and saturation (HSV color space) ie the natural components of color perception. The three channels are then quantized, by using a quantization matrix Q_r, into a space consisting of a finite number of colors. The n^{th} component of the histogram, $h_r[n]$ is given by:

$$h_c[m] = \sum_x \sum_y \begin{pmatrix} 1 & yQ_r(T_rI[x,y] = n \\ 0 & otherwise \end{pmatrix} \quad \textbf{(1)}$$

In Fig 2, color histogram are calculated for two images and the similarity comparison is performed between the two vectors **p**1 and **p**2.

Communications on Applied Electronics (CAE) – ISSN : 2394-4714
Foundation of Computer Science FCS, New York, USA
Volume 4– No.7, March 2016 – www.caeaccess.org

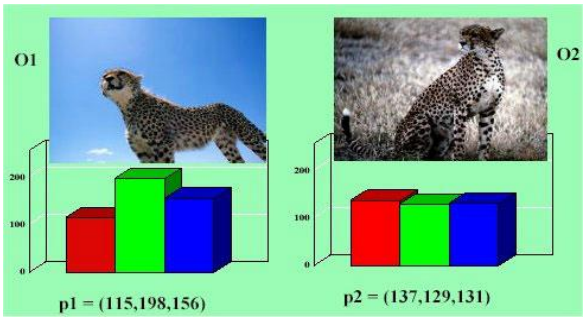

Fig 2. Color histogram extractions using 3 colors.

2.2 Image Retrieval by Texture Representation

Textures are homogeneous patterns or spatial arrangements of pixels that cannot be sufficiently described by regional intensity or color features. Based on the extraction of information on coarseness, contrast, and direction the texture properties can be globally represent. **A**n alternate method to represent texture property is Gabor filter. So, for each scale and direction, the luminance information is transformed with the corresponding Gabor filter and mean and variance are computed for each scale and direction. A common Gabor filter approach uses 5 directions and 3 scales, determining a feature vector defined in a 3 dimensional space. Manhattan distance is used to compare two images.

2.3 Image Retrieval by Shape Representation.

Shape representation techniques fall in two major categories:

1. The feature vector approach.
2. The shape through transformation approach.

The choice of a particular representation is driven by application needs, like characteristics of the shapes being analyzed, robustness against noise, and possibility of indexing. The feature vector approach is widely employed in information retrieval and allows effective indexing. A shape is represented as a numerical vector using a parametric internal method, or a parametric external method. The Euclidean distance is the most used distance function to compare two shapes. On the other hand, shapes can be also compared computing the effort needed to transform one shape into the other. In this case, similarity is computed by way of a transformational distance. The main disadvantage of this approach, however, is that it does not support indexing, due to the fact that the method used to assess similarity does not satisfies metric postulates.

3. MAP REDUCE

Map/Reduce [11] is a "programming model and an associated implementation for processing and generating large data sets". In order to gain reasonable amount of time in case of large computations, distributed processing using hundreds or thousands of machines are required. But carefully distributing the data requires effective partitioning and parallel computation and Map/Reduce was designed for that. It lets the programmer to write simple units of work as map and reduce functions. A typical Map Reduce application consists of three functions: map function, partition function and reduce function. This frame work can then distribute the data to different machines by partitioning the data and executing the task in parallel.

map (k1, v1) -> k2,v2

reduce (k2, list(v2)) -> v3

Map Reduce can be done in the following steps:

1. The Map/Reduce frame work first splits the input data into n pieces of fixed size and then passed to the participating machines in the cluster.
2. One of the nodes in the cluster is the master and rest are slaves which performs the work assigned by the master.
3. The slave reads the content and parses key/value pairs and passes to the map function. The intermediate key/value pairs are buffered in memory and periodically written to local disk and partitioned by the partitioning function and pass to reduce.
4. When a reduce worker has read all intermediate data, it sorts it by the intermediate keys (**k2**) so that all occurrences of the same key are grouped together.
5. Next, the reduce worker iterates over the sorted intermediate data and for each unique intermediate key encountered, it passes the key and the corresponding set of intermediate values to the users **reduce** function. The output of the **reduce** function is appended to a final output file for this reduce partition.
6. When all map tasks and reduce tasks have been completed, the master wakes up the user program. At this point, the Map/Reduce call in the user program returns back to the user code.

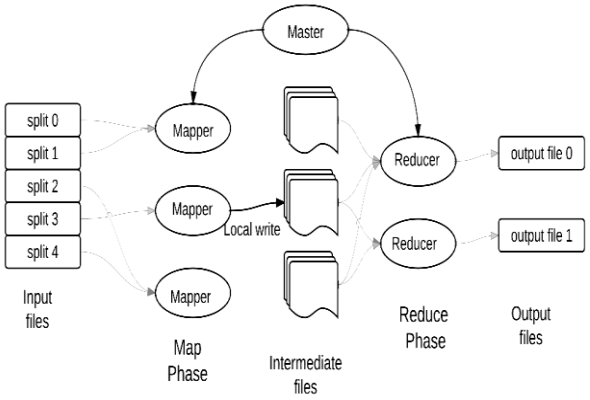

Fig 3.Map Reduce

The intermediate key/value pair from the map task is passed on to a practitioner which in turns calls the practitioner function as shown in Figure 4. **I**t takes as input the key/value pair and returns the reducer to which this key/value pair should be sent. The number of partitions is equal to the number of reduce tasks for the job. The amount of data received from each mapper to a reducer and the total size of data to be processed by the reduce task will only be known after the map tasks complete execution.

Communications on Applied Electronics (CAE) – ISSN : 2394-4714
Foundation of Computer Science FCS, New York, USA
Volume 4– No.7, March 2016 – www.caeaccess.org

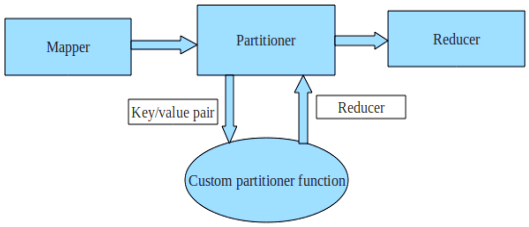

Fig 4. Map Reduce Job

4. IMPLEMENTING MAP REDUCE USING TWO DIMENSIONAL BINARY SEARCH TREE

The images are divided equally among different machines, each building its own Tree from its chunk of images. The tree is divided into a "root sub tree" that resides on a root machine, and several "leaf sub trees", each residing on a leaf machine[7] as shown in Fig 5. At query time, the root machine directs the features into the appropriate leaf machines depending on where they exit the tree on the root machine. The leaf machines compute the nearest neighbors within their sub tree and send them back to the root machine, which performs the counting and outputs the final sorted list of images.

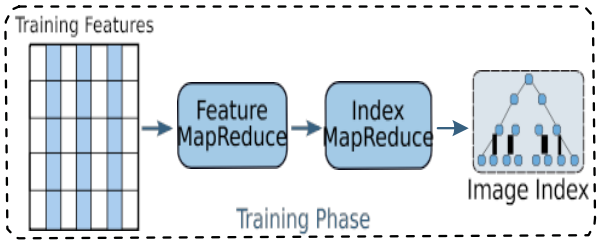

Fig.5. The Training MapReduce distributes the features among the different machines.

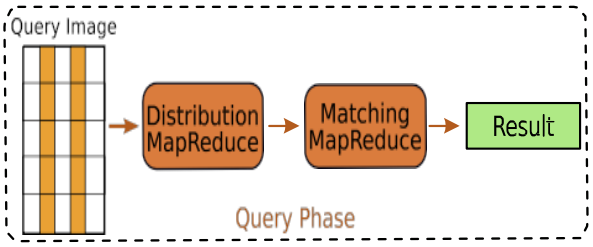

Fig.6. In the query phase, the query image is first routed through the Distribution Map reduce

The main advantage of distributed binary tree is that a single feature will only go to a small subset of the leaf machines, and thus the leaf machines will be processing multiple features at the same time[10]. Most of the computations are done in the leaf machines., When the number of leaf machines increases the bottle neck can be avoided in the leaf machine by having multiple copies of the root machine. The problems faced are: (a) If the tree contains billions of features it is difficult to built the tree since it does not fit on one machine. (b) In case, if back tracking is needed how to perform that.

These two problems are solved by noticing the properties of Trees: (a) The 2-D tree are not only built on one machine; rather build a feature "distributor" that represents the top part of the tree, on the root machine. Since it is not possible to fit all the features in the database in one machine, simply subsample the features and use as many as the memory of one machine can take. (b) Backtracking is performed only in the leaf machines, and not in the root. To decide which leaf machines to go , test the distance to the split value, and if it is below some threshold S_t, include the corresponding leaf machine in the process[7].

The MapReduce architecture for implementing tree is as shown in Figure 5 and Figure 6. It proceeds in two phases:

1. Training Phase: The Feature MapReduce directs the training features into the different machines, which then build the two dimensional binary tree with the features assigned to it during the Index MapReduce.
2. Query Phase: The Distribution MapReduce directs the query features into the appropriate machines, which perform the Matching MapReduce.

Given M machines, the top part of the Kdt should have $\lceil \log 2M \rceil$ levels, so that it has at least M leaves. The Feature Map subsamples the input features by emitting one out of every input *skip* features, and the feature Reduce builds the tree with those features. The Index MapReduce builds the M bottom parts of the tree, where the Index Map directs the database features to the tree that is going to own it, which using depth first search identifies as the first leaf of the top part. The Index Reduce then builds the respective leaf trees with the features it owns. At query time, the Distribution MapReduce dispatches the query features to zero or more leaf machines, depending on whether the distance to the split value is below the threshold S_t . The Matching MapReduce then performs the search in the leaf trees and the counting and sorting of images.

5. EXPERIMENTAL RESULTS

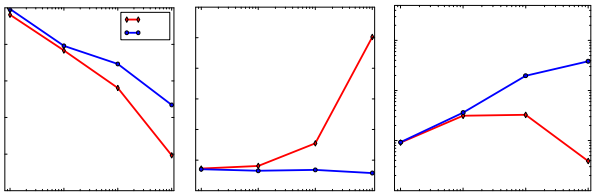

Fig.7 Effect of Number Images. The X-axis depicts the number of images in the database. The Y-axis depicts precision@1 (left), CPU time (center), and Throughput (right).

6. CONCLUSION

The image search system takes query input in the form of image and retrieves relevant images from huge database. A novel Internet Search system have been implemented which only requires one-click user feedback. Rather than comparing image as a whole, features are extracted. The Feature Vectors are stored in a two dimensional binary tree thereby reducing the searching time. Feature vectors are used to compare the similarity between images which reduce the search time tremendously. The search is performed in parallel using distributed machines arranged using map reduce concept.

7. REFERENCES

[1] Ilaria Bartolini, Paolo Ciaccia, an d Marco Patella. A sound algorithm for region-based image retrieval using an index. In proceedings of the 4th International Workshop on Query Processing and Multimedia Issue in Distributed Systems (QPMIDS'00), pages 930–934, Greenwich, London, UK, September 2000.

[2] Ilaria Bartolini, Paolo Ciaccia, and Florian Waas. Feedback Bypass: A new approach to interactive similarity query processing. Technical Report CSITE-09-01, CSITE–CNR, 2001. Available at URL http://www-db.deis.unibo.it/ MMDBGroup/TRs.html.

[3] Stefan Berchtold, Daniel A. Keim, and Hans-Peter Kriegel. The X-tree: An index structure for high-dimensional data. In Proceedings of the 22nd International conference on Very Large Data Bases (VLDB'96), pages 28–39, Mumbai (Bombay), India, September 1996.

[4] Christos Faloutsos, Will Equitz, Myron Flickner, Wayne Niblack, Dragutin Petkovic, and Ron Barber. Efficient and effective querying by image content. Journal of Intelligent Information Systems, 3(3/4):231–262, July 1994.

[5] Yong Rui, Thomas S. Huang, Michael Ortega, and Sharad Mehrotra. Relevance feedback: A power tool for interactive content-based image retrieval. IEEE transaction on Circuits and Systems for Video Technology, 8(5):644–655, September 1998.

[6] Mohamed Aly, Mario Munich, and Pietro Perona. Indexing in large scale image collections: Scaling properties and benchmark. In WACV, 2011.

[7] Ulrich Buddemeier and Alessandro Bissaccoo. Distributed kd-tree for efficient approximate nearest neighbor search, 2009.

[8] M. Muja and D. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In VISAPP, 2009.

[9] J. Zobel and A. Moffat. Inverted files for text search engines. ACM Comput. Surv., 2006. ISSN 0360-0300.

[10] T.Cormen,C.Leiserson, R. Rivest, and C. Stein. Introduction to Algorithms. McGraw- Hill, 2001.

[11] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. In OSDI, 2004.

[12] H. Jégou, M. Douze, C. Schmid, and P. Pérez. Aggregating local descriptors into a compact image representation. In CVPR, 2010.

[13] S. Arya, D.M. Mount, N.S. Netanyahu, R. Silverman, and A.Y. Wu. An optimal algorithm for approximate nearest neighbor searching. Journal of the ACM, 45:891–923, 1998.

8. AUTHOR PROFILE

Radhakrishnan B is working as Asst. Professor in Computer Science department. He has more than 14 years experience in teaching and has published papers on data mining and image processing. His research interests include image processing, data mining, and image mining .

Anver Muhammed K.M is working as System Administrator in Computer Science department. He has more than 10 years experience in System Administration and Implementation. His research interests include image processing and network security.