# Content Based Image Retrieval ang Image Processing using Hadoop

Project report submitted in partial fulfillment
of the requirements for the degree of

*Bachelor of Technology*
*in*
*Computer Science and Engineering*

by


| Names of Students | Roll No |
|---|---|
| Ashish Agarwal | Y11UC062 |
| Ayush Kasliwal | Y11UC069 |
| Mahak Jain | Y11UC128 |


Under Guidance of
Prof. Ravi Prakash Gorthi
Prof. Vikas Bajpai



Department of Computer Science Engineering
The LNM Institute of Information Technology, Jaipur


May 2015

The LNM Institute of Information Technology
Jaipur, India

# CERTIFICATE

This is to certify that the project entitled Private Content Based Image Information Retrieval using Map-Reduce submitted by Ashish Agarwal (Y11UC062), Ayush Kasliwal (Y11UC069), Mahak Jain (Y11UC128) in partial fulfillment of the requirement of degree in Bachelor of Technology (B. Tech), is a bonafide record of work carried out by him/her at the Department of Computer Science and Engineering, The LNM Institute of Information Technology, Jaipur, (Rajasthan) India, during the academic session 2014-2015 under my supervision and guidance and the same has not been submitted elsewhere for award of any other degree. In my/our opinion, this thesis is of standard required for the award of the degree of Bachelor of Technology (B. Tech).

Adviser: Prof. Ravi Prakash Gorthi

Adviser: Prof. Vikas Bajpai

Date

To SOMEONE

# Acknowledgments

I would like to express my sincere gratitude towards **Prof. Ravi Prakash Gorthi** and **Prof. Vikas Bajpai**, our research guides, for their patient guidance, enthusiastic encouragement and useful critiques on this research work. Without his invaluable guidance, this work would never have been a successful one.

I would also like to thank my classmate for continuous and detail knowledge sharing on Hadoop and other aspects from installation to working. I would also like to thank all my B.Tech friends for unconditional support and help.

Last but not least, my sincere thanks to all my teachers who directly or indirectly helped me to learn new technologies and complete my post graduation successfully.

Although there may be many who remain unacknowledged in this humble note of gratitude there are none who remain unappreciated.

# Abstract

Today, Role of Information technology is changing and this change is leading to real life software application developments i.e. Information technology has entered every part of our day to day life like traffic signals, power grids, our day to day financial transactions are handled by Information technology. But this technology comes with a cost to pay for it. As this technology is touching each and every layer of life and community, there is a need to make this technology available to each and every layer of community.

Cloud computing is an emerging technology which reduces this cost while providing very good and efficient information technology platform as well as service. Cloud reduces the overload on hard disk, operating system and eventually on CPU processing. Cloud providers provide infrastructure, platform, database etc. for you to install your operating system, your required softwares, to store your data on their cloud. But this service comes at a cost. Cloud providers provide it on pay per hour basis. So any application which can be deployed on cloud must be fast, efficient and secure.

As Internet is expanding, creation of data over it is also increasing exponentially. Maximum part of this data contains images. Today large amount of variety image data is produced through digital cameras, mobile phones, and photo editing softwares etc.

For other tasks related to large-scale data, humans have already turned towards distributed computing as a way to side-step impending physical limitations to processing hardware by combining the resources of many computers and providing programmers various different interfaces to the resulting construct, relieving them from having to account for the intricacies stemming from its physical structure. An example of this is the MapReduce model, which - by way of placing all calculations to a string of Input-Map-Reduce-Output operations capable of working independently - allows for easy application of distributed computing for many trivially parallelised processes. With the aid of freely available implementations of this model and cheap computing infrastructure offered by cloud providers, having access to expensive purpose-built hardware or in-depth understanding of parallel programming are no longer required of anyone who wishes to work with large-scale image data. In this thesis, I look at the issues of processing two kinds of such data - large data-sets of regular images and single large images - using MapReduce.

By further classifying image processing algorithms, we present a general analysis on why different combinations of algorithms and data might be easier or harder to adapt for distributed processing with MapReduce. Finally, we describe the application of distributed image processing on large data-set of photographs. Both preliminary analysis and practical results indicate that the MapReduce model is well suited for distributed image processing.

There are some domains which uses content based image retrieval applications. These applications should be fast enough to carry out user functionalities efficiently and secure in order to protect user data. If application is to be deployed on cloud, then it should be supported by cloud structure. One technology which has support from cloud and does distributed parallel computing is Map Reduce.

I am proposing a system which will allow user to upload, search and retrieve some of users personal images depending upon one particular image, efficiently from his continuously expanding image database. This system is incorporated with Map-Reduce technique for efficiently upload, search and retrieval of images over large dataset of user images.

*Proposed solution will be a system to upload, search and query images among massive image data storage using content based image retrieval scheme using Map Reduce technique.*

**Keywords** - Information technology, Cloud computing, Big Data, Map Reduce.

# Contents

# List of Figures

# Chapter *1*

# Introduction

## 1.1 The Area of Work

### 1.1.1 Cluster Computing

A computer cluster is a single logical unit consisting of multiple computers that are linked through a LAN. The networked computers essentially act as a single, much more powerful machine. A computer cluster provides much faster processing speed, larger storage capacity, better data integrity, superior reliability and wider availability of resources. Computer clusters are, however, much more costly to implement and maintain. This results in much higher running overhead compared to a single computer.

### 1.1.2 Hadoop

The Apache Hadoop is a collection open-source software projects for reliable, scalable, distributed computing. Software libraries of Hadoop specify a framework that allows distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from single to thousands of machines, each offering local computation and storage.

Some modules of Hadoop are listed below:

1. *Hadoop Common*: The common utilities that support the other Hadoop modules.

2. *Hadoop Distributed File System (HDFS)*: A distributed file system that provides high-throughput access to application data.

3. *Hadoop YARN*: A framework for job scheduling and cluster resource management.

4. *Hadoop MapReduce*: A YARN-based system for parallel processing of large data sets.

5. *HBase*: A scalable, distributed database that supports structured data storage for large tables.

#### 1.1.2.1 MapReduce

Map reduce is a framework for processing parallel problems across huge datasets using a large number of commodity computers (nodes). The computation performed on the nodes is independent of the physical location of nodes i.e. whether all nodes are on same network and use similar hardware (called as cluster) or nodes are shared across geographically and administratively distributed systems, and use more heterogeneous hardware (called as grid). It is also independent of location of data. Data can be stored into cluster at any node.

The computation takes a set of input key/value pairs, and produces a set of output key/value pairs. The user of the MapReduce library expresses the computation as two functions: map and reduce.

*Map function*, written by the programmer, takes an input pair of key/value and produces a set of intermediate key/value pairs. The MapReduce library groups together all values associated with the same key and passes them to the Reduce function.

*Reduce function*, also written by the programmer, accepts pair of key/value which is generated by map function. It merges together these values to form a possibly smaller set of values. Typically just zero or one output value, which is actual output of the user program.

map (k1, v1) — list(k2, v2)
reduce (k2, list(v2)) — list(k3, v3)

Computational processing can occur on data stored either in a file system (unstructured/HDFS) or in a database (structured/HBase).

**Map**: The master node takes the input, divides it into smaller sub-problems, and distributes them to worker nodes. A worker node may do this again in turn, leading to a multi-level tree structure. The worker node processes the smaller problem, and passes the answer back to its master node.

**Reduce**: The master node then collects the answers to all the sub-problems and combines them in some way to form the output  the answer to the problem it was originally trying to solve.

MapReduce allows for distributed processing of the map and reduction operations. Provided each mapping operation is independent of the others. All maps can be performed in parallel  though in practice it is limited by the number of independent data sources and/or number of CPUs near each source. Similarly, a set of reducers can perform the reduction phase provided all outputs of the map operation that share the same key are presented to the same reducer at the same time, or if the reduction function

is associative.

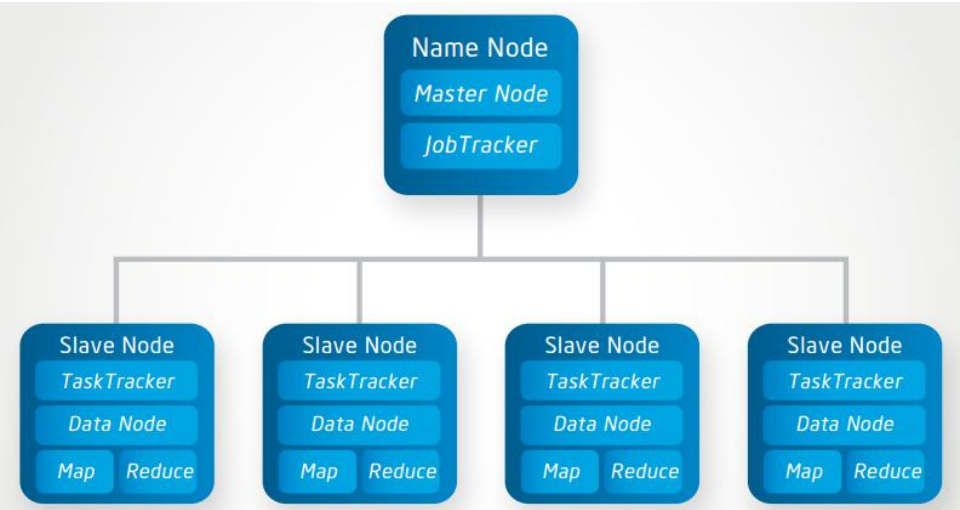


**Figure 1.1** Hadoop Architecture

**Example**

Lets look at a simple example. Assume you have five files, and each file contains two columns (a key and a value in Hadoop terms) that represent a city and the corresponding temperature recorded in that city for the various measurement days. Of course weve made this example very simple so its easy to follow. You can imagine that a real application wont be quite so simple, as its likely to contain millions or even billions of rows, and they might not be neatly formatted rows at all; in fact, no matter how big or small the amount of data you need to analyze, the key principles were covering here remain the same. Either way, in this example, city is the key and temperature is the value.

Toronto, 20
Whitby, 25
New York, 22
Rome, 32
Toronto, 4
Rome, 33
New York, 18

Out of all the data we have collected, we want to find the maximum temperature for each city across all of the data files (note that each file might have the same city represented multiple times). Using the MapReduce framework, we can break this down into five map tasks, where each mapper works on one of the five files and the mapper task goes through the data and returns the maximum temperature for

each city. For example, the results produced from one mapper task for the data above would look like this:

(Toronto, 20) (Whitby, 25) (New York, 22) (Rome, 33)

Lets assume the other four mapper tasks (working on the other four files not shown here) produced the following intermediate results:

(Toronto, 18) (Whitby, 27) (New York, 32) (Rome, 37) (Toronto, 32) (Whitby, 20) (New York, 33) (Rome, 38)(Toronto, 22) (Whitby, 19) (New York, 20) (Rome, 31) (Toronto, 31) (Whitby, 22) (New York, 19) (Rome, 30)

All of these output streams would be fed into the reduce tasks, which combine the input results and output a single value for each city, producing a final result set as follows:

(Toronto, 32) (Whitby, 27) (New York, 33) (Rome, 38)

As an analogy, you can think of map and reduce tasks as the way a census was conducted in Roman times, where the census bureau would dispatch its people to each city in the empire. Each census taker in each city would be tasked to count the number of people in that city and then return their results to the capital city. There, the results from each city would be reduced to a single count (sum of all cities) to determine the overall population of the empire. This mapping of people to cities, in parallel, and then combining the results (reducing) is much more efficient than sending a single person to count every person in the empire in a serial fashion.

#### 1.1.2.2 HDFS

The Hadoop Distributed File System (HDFS) is a another module of Hadoop Project. It is a distributed file system, which is designed to run on commodity hardware. HDFS is highly fault-tolerant and works on low-cost hardware. HDFS provides high throughput access to application data and is suitable for applications that have large data sets.

#### 1.1.2.3 HBase

HBase actually works on HDFS. It is a distributed column oriented database built on top of HDFS. It is a Hadoop application which is used, when user require real-time read/write random-access to very large datasets. HBase can scale linearly just by adding nodes. It is not relational and does not support SQL, but given the proper problem space, it is able to do what an RDBMS cannot: host very large, sparsely populated tables on clusters made from commodity hardware.

## 1.2 Problem Addressed

It is common knowledge that a computer consists of a processor, memory and a hard drive. The processor performs calculations on the data stored in memory, which has previously been read from a hard drive. It is important to note here that since very many computers are also connected to the Internet, the hard drive in question may reside in a different physical location than the processing unit and memory. Now, it is also known that the data transfer speed between the processor and memory is generally orders of magnitude faster than between memory and hard drive. Similarly, reading from a local hard drive is faster than accessing data from storage in a different computer, due to overhead added by having to communicate over a network.

Therefore, as the size of the data to be processed by one algorithm increases so that the computer no longer can hold all the information in memory, there is a significant decrease in processing speed. Similarly, if the data does not fit on the local hard drive, the processing speed drops due to having to wait for it to be sent in from another computer. While this can be alleviated somewhat by using buffering techniques, the general rule remains the same: its best if the problem fits within memory, worse if it fits on the local hard drive and worst if the data has to be read across the network.

Processing a large image is an example of such a problem. In this case we are dealing with a microscope image, where each pixel is made up of 3 values - red, green and blue. Assuming that each of these values is stored as a 32-bit precision floating point number, the total memory consumption of storing this data in an uncompressed way is very huge. However, our aim is to find out whether it is possible to process this kind of images using commodity computers in such a way that all the necessary data is stored within memory and can be processed in an efficient way.

It is clear that processing the classes of large images or large data sets of regular images can not be done on a single personal computer, because first, they do not fit into memory, and second, one computer can not process them fast enough. Neither of these issues can be expected to be solved by advances in computing power, because CPUs are already reaching their theoretical physical limitations and the scale of data is increasing faster than the processing capabilities of single commodity computers.

Also at the start, image retrieval is mainly dependent on the text based retrieval of an image. This technology is widely used, the current mainstream search engines Google, Baidu, Yahoo, etc. mainly used this method to search image. In this technique, name of image file was used to compare and retrieve the mage from database. But it has drawbacks, researchers often need to manually mark with text for all images, and this mark text cant objectively and accurately describe the visual information of the images.

A solution for these problems is turning towards distributed computing, where limitations of a single computer are overcome by combining the resources of many computers to perform one large task. While this approach is not new - supercomputers and computing clusters have existed for many years already -

it is only recently that techniques of using commodity computers for distributed processing have gained popularity. In the following we will explain more thoroughly how these technologies could be used to solve image processing tasks.

Content-Based Image Retrieval, which can extract visual features from image automatically, and then retrieval image by their visual features. Since CBIR intuitive, efficient, and can be widely used in information retrieval, medical diagnosis, trademarks and intellectual property protection, crime prevention and other areas, it has a very high applied value.

## 1.3 Existing System

This project is focused on using the MapReduce model for performing image processing, we will now describe some of the issues that stem from the limitations of this model with regard to images.

Generally speaking, the MapReduce parallel computing model follows what can be called a divide-and-conquer strategy of parallelised computing. That is, instead of joining together physical resources like processing power, memory and hard drive storage in order to allow the processing software to see these combined devices as one monolithic entity, the problem is divided into independent parts which are then processed separately - usually in different physical or virtual computers - and later joined together to form the output.

Local, with regard to image processing, denotes that the computation is performed as a series of small calculations on fixed subsets of the image: typically this means that the value of a pixel in focus is recalculated using the values of its neighbouring pixels. It is easy to see how problems like this can be parallelised by virtue of splitting the image into parts, performing the processing, and later putting it back together. Gaussian blurring, is an example of a local processing algorithm. Contrary to local, non-local problems involve larger parts of the image. A good example of non-local processing is object recognition. The solution of splitting the image into parts to allow for parallel processing now requires special attention in order to avoid splitting objects into unrecognisable fragments, and in the worst case could be entirely inapplicable if the object to be classified takes up the whole image.

In the first case of processing a image, it immediately be comes obvious that parallelising a non-local type of algorithm is going to be a non-trivial task, as we lack even the ability to store it in memory on regular computers. Even when assuming that the algorithm can continue to function if the input image is split to pieces, if communication is also required between the computers working on separate pieces, then there is a good chance that network speed will become a bottleneck and slow down the computation.

The second case, however, is far more suited for processing with the MapReduce model. Even though the total size of the data is several times bigger than in the previous case, since it consists of compara-

tively small images which easily fit into memory even when taking any algorithm-specific overhead into account. Moreover, because we do not have to split any images into pieces or worry about communication between worker computers, classification of the algorithms into the aforementioned four groups does not matter. Therefore, looking at the issues with regard to analysing this sort of data lets us make conclusions that apply to a wider range of problems.

At this point it is important to note that we have so far silently assumed that all the algorithms we classify only require one image as an input. Speaking from the perspective of distributed processing, this means we assume that the algorithm only requires data from one image at a time. For example, with this clause we exclude any processing that needs to compare two or more images with each other from this discussion. The reason behind this is somewhat related to the implementation of the MapReduce model in Apache Hadoop and our approach to parallelising image processing tasks by dividing images into manageable pieces. It can be summarised as follows: if an algorithm requires access to more images than the local storage of the computer allows, communication between computers is needed. However, since a MapReduce calculation has only one step where the computing nodes exchange information, the only way to satisfy this need without resorting to another processing model is to run the MapReduce calculations themselves iteratively This, in turn, has been shown to be very slow in actual performance, especially as the number of iterations increases.

## 1.4 Motivation

In the previous sections, we have presented some general analysis with regard to the general feasibility of using the MapReduce model of distributed 8computing to solve image processing tasks. In this part we will summarise the main motivation behind choosing MapReduce and its implementation in the form of Apache Hadoop, and briefly outline alternative ways how one could approach distributed image processing.

First, what are the alternatives? Batch processing on the PC is feasible for only small amounts of data, and since only a part of this data fits into memory at given time, computation will suffer from a decrease in speed due to slow hard drive access. Trying to counter this by running the batch process on several computers simultaneously is a solution, but it creates a need for job monitoring, mechanisms for data distribution and means to ensure that the processing completes even when some computers experience failures during work. This is more or less exactly the problem that both Google MapReduce and Apache Hadoop were designed to solve. Another approach is treating the problem like a traditional large-scale computing task which requires specialised hardware and complex parallel programming.

Cluster computers built on graphics processing units (GPU) are an example of this, and while maintaining a purpose-built computer cluster has been shown to be a working solution for many kinds of problems, it is interesting to know whether the same issues can be tackled with simpler and cheaper systems without much decrease in efficiency.

## 1.5 Creation of bibliography

Use sampleBib.bib file to save your bib format citations. Use the command [**?**] for referring to a particular article.

# Chapter *2*

# Literature survey

In this chapter we will describe and summarise relevant work that has been done in the field of distributed image processing, then describe the MapReduce computing model with regard to Apache Hadoop and Hadoop Distributed Filesystem.

Overall Hadoop has shown its utility for large scale medical image computing. The three usecases reflect the various challenges of processing medical visual information in clinical routine: parameter optimization, indexation of image collections with hundreds of thousands images, and multi dimensional medical data processing. In all tasks very positive results could be obtained helping the projects to scale with limited local resources available and moderate efforts to adapt the software.

In an experiment, they processed sequences of video frames with MapReduce to create grayscale images and extracted some features of the video images. In the process of creating the grayscale images, each video frame was divided into multiple parts. In the extraction, frame numbers were used as the key numbers to extract some features of the video images. It was proposed in future, it is necessary for us to build a distributed environment that combines multiple machines, conduct large-scale experiments involving sequential video images, evaluate the performance speed of the implementation with MapReduce, and verify the efficient key-value pairs.

This paper shows a comparative study of the Apache Hadoop and the Apache Storm focusing on the efficiency of the image feature extraction in order to prepare the data for the content-based retrieval methods. The evaluated Big Data frameworks have been compared in terms of scalability from workload and resource-size points of view. Moreover, an evaluation processing time was also measured in order to test the frameworks with different problem sizes.

The proposed module consists of two parts: storage system (HDFS) for image data and MapReduce program with JAI library for image transcoding. The proposed module can process image data in distributed and parallel in cloud computing environment. Thus, our proposed module can minimize overhead of computing infrastructure. In this paper, we explain how to implement the proposed module

with Hadoop and JAI. In addition, we evaluated our proposed module in terms of processing time.

Content-Based Image Retrieval, the following unified call CBIR, and different from TBIR, it can extract visual features from image automatically, and then retrieval image by their visual features. Since CBIR intuitive, efficient, and can be widely used in information retrieval, medical diagnosis, trademarks and intellectual property protection, crime prevention and other areas, it has a very high applied value.

A Distributed Image Retrieval System(DIRS) is a system in which images are retrieved in a content based way, and the retrieval among massive image data storage is carried parallely by utilizing MapReduce distributed computing model.

While the above shows that there has been a lot of work in this area the question remains whether (and how well) Hadoop is suited for large scale image processing tasks, because as evidenced by this brief overview, there are only a few cases where image processing has been done with MapReduce.

## 2.1 Image Retrieval

An image retrieval system is designed to browse, search and retrieve images from large database of digital images. Most traditional and common methods of image retrieval utilize method of adding metadata such as captioning, keywords, or descriptions to the images so that retrieval can be performed over the annotation words. Manual image annotation is time-consuming, laborious and expensive; to address this, there has been a large amount of research done on automatic image annotation.[**?**] Annotated images are searched based on Imagess matadata.

*Image Meta search:* Search of images based on associated metadata such as keywords, text, etc.

*Content-based image retrieval (CBIR):* The application of computer vision to the image retrieval. CBIR aims at avoiding the use of textual descriptions and instead retrieves images based on similarities in their contents (textures, colors, shapes etc.) to a user-supplied query image or user-specified image features.

## 2.2 Content Based Image Retrieval

Since 1970s, image retrieval has been an active research area. In the beginning, research was concentrated to text based search only. It was new framework at that time, which used names of image files as a search criteria. In this framework, images were need to be annotated first and then from database management system, images were retrieved. This framework was having two limitations firstly size of

image data that can fit into database and secondly extensive manual annotation work. There was need to do research work on retrieval which would not be having limitation of manual entry and big size data.

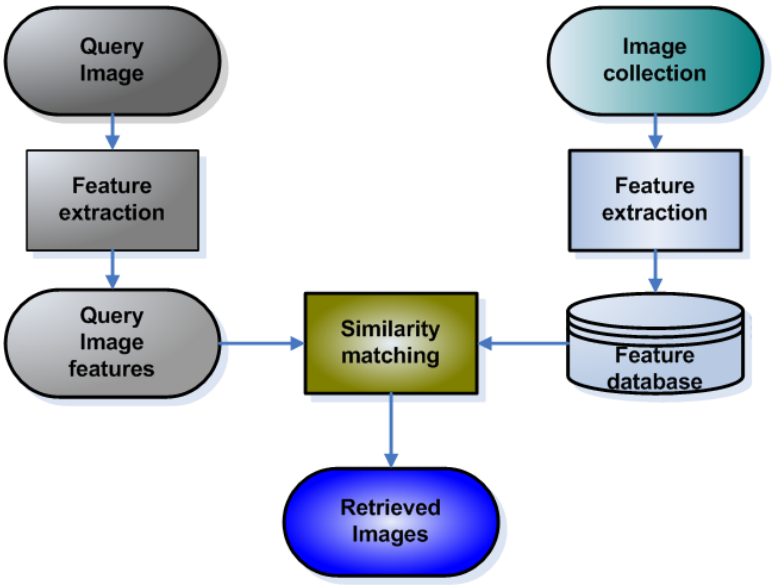


**Figure 2.1** CBIR Architecture

It lead to Content Based Image Retrieval technique. Content-based image retrieval (CBIR), also known as query by image content (QBIC). Content based image retrieval (CBIR) is a technique in which content of an image is used as matching criteria instead of images metadata such as keywords, tags, or any name associated with image. This provides most approximate match as compared to text based image retrieval.

The term content in this context might refer to colors, shapes, textures, or any other information that can be derived from the image itself. Image content can be categorized into visual and semantic contents. Visual content can be very general or domain specific. General visual content include color, texture, shape, spatial relationship, etc. Domain specific visual content, like human faces, is application dependent and may involve domain knowledge. Semantic content is obtained either by textual annotation or by complex inference procedures based on visual content.

**COLOR:** Image retrieval based on color actually means retrieval on color descriptors. Most commonly used color descriptors are the color histogram, color coherence vector, color correlogram, and color moments [9]. A color histogram identifies the proportion of pixels within an image holding specific values which can be used to find similarity between two images by using similarity distance measures.It tries to identify color proportion by region and is independent of image size, format or orientation. Color moments contains calculation of the first order (mean), the second (variance) and the third order (skewness) of an image.

**TEXTURE:** Texture of an image is actually visual patterns that an image possesses and how they are spatially defined. Textures are represented by texels which are then placed into a number of sets, depending on how many textures are detected in the image. These sets not only define the texture, but also where in the image the texture is located. Statistical method such as co-occurrence Matrices can be used for quantitative measure of the arrangement of intensities in a region.

**SHAPE:** Shape in image does not mean shape of an image but it means that shape of a particular region or an object. Segmentation and edge detection are prominent techniques that can be used in shape detection.

There are several components that are calculated from image content such as color intensity, entropy, mean of image etc. which are useful in creation of feature vector. Feature vector of every image is calculated and is stored in database.

## 2.3 Image Processing Techniques/Methods

The technique we used in this is extracting the features of the image. This mainly contains the following algorithms:

Histogram, Edge detection (Canny and Sobel), Gaussian blur, Face Detection, rgb2grey conversion, etc. Taking into account the color histogram algorithm is widely used, its feature extraction and similarity matching are more easier, this method will be used as our target color algorithm.

### 2.3.1 RGB to Gray Conversion

In photography and computing, a grayscale or greyscale digital image is an image in which the value of each pixel is a single sample, that is, it carries only intensity information. Images of this sort, also known as black-and-white, are composed exclusively of shades of gray, varying from black at the weakest intensity to white at the strongest.

### 2.3.2 Histogram

An image histogram is a type of histogram that acts as a graphical representation of the intensity distribution in a digital image. It plots the number of pixels for each intensity value. By looking at the histogram for a specific image a viewer will be able to judge the entire intensity distribution at a glance.

### 2.3.3 Edge Detection

Edge detection is an image processing technique for finding the boundaries of objects within images. It works by detecting discontinuities in brightness. Edge detection is used for image segmentation and

data extraction in areas such as image processing, computer vision, and machine vision. Common edge detection algorithms used are Canny, Sobel etc.

### 2.3.4 Gaussian Blur

The Gaussian smoothing operator is a 2-D convolution operator that is used to 'blur' images and remove detail and noise. In this sense it is similar to the mean filter, but it uses a different kernel that represents the shape of a Gaussian ('bell-shaped') hump.

The idea of Gaussian smoothing is to use this 2-D distribution as a 'point-spread' function, and this is achieved by convolution. Since the image is stored as a collection of discrete pixels we need to produce a discrete approximation to the Gaussian function before we can perform the convolution. One could use the value of the Gaussian at the centre of a pixel in the mask. We integrated the value of the Gaussian over the whole pixel (by summing the Gaussian at 0.001 increments).

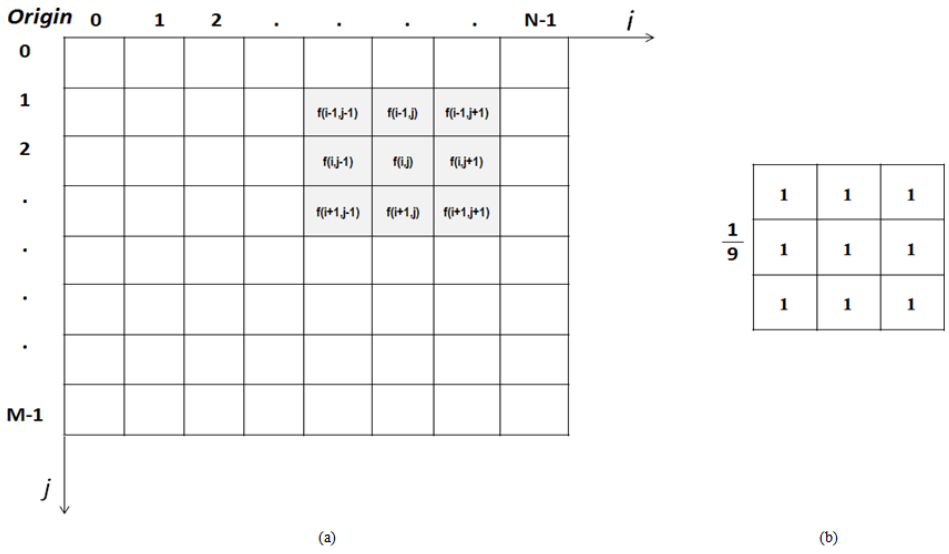


**Figure 2.2** Gaussian Blur

#### 2.3.4.1 Canny Edge Detection

The Canny edge detector is an edge detection operator that uses a multi-stage algorithm to detect a wide range of edges in images.
The Process of Canny edge detection algorithm can be broken down to 5 different steps:

1. Apply Gaussian filter to smooth the image in order to remove the noise
2. Find the intensity gradients of the image
3. Apply non-maximum suppression to get rid of spurious response to edge detection

4. Apply double threshold to determine potential edges
5. Track edge by hysteresis: Finalize the detection of edges by suppressing all the other edges that are weak and not connected to strong edges.

#### 2.3.4.2 Sobel Edge Detection

The Sobel operator performs a 2-D spatial gradient measurement on an image and so emphasizes regions of high spatial frequency that correspond to edges. Typically it is used to find the approximate absolute gradient magnitude at each point in an input grayscale image.

$$G_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * \begin{bmatrix} A_{i-1,i-1} & A_{i,i-1} & A_{i+1,i-1} \\ A_{i-1,i} & A_{i,i} & A_{i+1,i} \\ A_{i-1,i+1} & A_{i,i+1} & A_{i+1,i+1} \end{bmatrix}$$

$$G_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * \begin{bmatrix} A_{i-1,i-1} & A_{i,i-1} & A_{i+1,i-1} \\ A_{i-1,i} & A_{i,i} & A_{i+1,i} \\ A_{i-1,i+1} & A_{i,i+1} & A_{i+1,i+1} \end{bmatrix}$$

$$G = \sqrt{G_x^2 + G_y^2}$$

**Figure 2.3** Sobel Filter

# Chapter *3*

# Installation

## 3.1 Installing Hadoop on Ubuntu (Single Node)

In this section, we'll install a single-node Hadoop cluster backed by the Hadoop Distributed File System on Ubuntu.

### 3.1.1 Installing Java

Hadoop framework is written in Java.

```
user@user:~$ sudo apt-get update
user@user:~$ sudo apt-get install default-jdk
user@user:~$ java -version
```

### 3.1.2 Adding a dedicated Hadoop user

```
user@user:~$ sudo addgroup hadoop
user@user:~$ sudo adduser --ingroup hadoop hduser
```

### 3.1.3 Installing SSH

ssh has two main components:
*ssh:* The command we use to connect to remote machines - the client.
*sshd:* The daemon that is running on the server and allows clients to connect to the server.
The ssh is pre-enabled on Linux, but in order to start sshd daemon, we need to install ssh first. Use this command to do that :

```
user@user:~$ sudo apt-get install ssh
```

### 3.1.4 Create and Setup SSH Certificates

Hadoop requires SSH access to manage its nodes, i.e. remote machines plus our local machine. For our single-node setup of Hadoop, we therefore need to configure SSH access to localhost.

So, we need to have SSH up and running on our machine and configured it to allow SSH public key authentication.
Hadoop uses SSH (to access its nodes) which would normally require the user to enter a password. However, this requirement can be eliminated by creating and setting up SSH certificates using the following commands. If asked for a filename just leave it blank and press the enter key to continue.

```
user@user:~$ su hduser
Password:
user@user:~$ ssh-keygen -t rsa -P ""
hduser@laptop:/home/k$ cat $HOME/.ssh/id_rsa.pub >> $HOME/.ssh/authorized_keys
```

The second command adds the newly created key to the list of authorized keys so that Hadoop can use ssh without prompting for a password.

We can check if ssh works:

```
hduser@laptop:/home/k$ ssh localhost
```

### 3.1.5 Install Hadoop

```
hduser@laptop:~$ wget http://../../hadoop-2.6.0.tar.gz
hduser@laptop:~$ tar xvzf hadoop-2.6.0.tar.gz
```

We want to move the Hadoop installation to the /usr/local/hadoop directory using the following command:

```
hduser@laptop:~$ /hadoop-2.6.0$ sudo mv * /usr/local/hadoop
[sudo] password for hduser:
hduser is not in the sudoers file.  This incident will be reported.
```

This error can be resolved by logging in as a root user, and then add hduser to sudo:

```
hduser@laptop:~/hadoop-2.6.0$ su k
Password:
user@user:/home/hduser$ sudo adduser hduser sudo
[sudo] password for k:
```

Now, the hduser has root priviledge, we can move the Hadoop installation to the /usr/local/hadoop directory without any problem:

```
user@user:/home/hduser$ sudo su hduser
hduser@laptop:~/hadoop-2.6.0$ sudo mv * /usr/local/hadoop
hduser@laptop:~/hadoop-2.6.0$ sudo chown -R hduser:hadoop /usr/local/hadoop
```

### 3.1.6 Setup Configuration Files

The following files will have to be modified to complete the Hadoop setup:

1. /.bashrc
2. /usr/local/hadoop/etc/hadoop/hadoop-env.sh
3. /usr/local/hadoop/etc/hadoop/core-site.xml
4. /usr/local/hadoop/etc/hadoop/mapred-site.xml.template
5. /usr/local/hadoop/etc/hadoop/hdfs-site.xml

1. **/.bashrc:**
   Before editing the .bashrc file in our home directory, we need to find the path where Java has been installed to set the JAVA_HOME environment variable using the following command:

```
hduser@laptop update-alternatives --config java
```

   Now we can append the following to the end of /.bashrc:

```
hduser@laptop:~$ vi ~/.bashrc

#HADOOP VARIABLES START
export JAVA_HOME=/usr/lib/jvm/java-7-openjdk-amd64
export HADOOP_INSTALL=/usr/local/hadoop
export PATH=$PATH:$HADOOP_INSTALL/bin
export PATH=$PATH:$HADOOP_INSTALL/sbin
export HADOOP_MAPRED_HOME=$HADOOP_INSTALL
export HADOOP_COMMON_HOME=$HADOOP_INSTALL
export HADOOP_HDFS_HOME=$HADOOP_INSTALL
export YARN_HOME=$HADOOP_INSTALL
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_INSTALL/lib/native
export HADOOP_OPTS="-Djava.library.path=$HADOOP_INSTALL/lib"
#HADOOP VARIABLES END

hduser@laptop:~$ source ~/.bashrc
```

   Note that the JAVA_HOME should be set as the path just before the .../bin/:

```
hduser@ubuntu-VirtualBox:~$ javac -version
hduser@ubuntu-VirtualBox:~$ which javac
hduser@ubuntu-VirtualBox:~$ readlink -f /usr/bin/javac
```

2. **/usr/local/hadoop/etc/hadoop/hadoop-env.sh:**
   We need to set JAVA_HOME by modifying hadoop-env.sh file.

```
hduser@laptop:~$ vi /usr/local/hadoop/etc/hadoop/hadoop-env.sh
export JAVA_HOME=/usr/lib/jvm/java-7-openjdk-amd64
```

Adding the above statement in the hadoop-env.sh file ensures that the value of JAVA_HOME variable will be available to Hadoop whenever it is started up.

3. **/usr/local/hadoop/etc/hadoop/core-site.xml:**
   The /usr/local/hadoop/etc/hadoop/core-site.xml file contains configuration properties that Hadoop uses when starting up. This file can be used to override the default settings that Hadoop starts with.

```
hduser@laptop:~$ sudo mkdir -p /app/hadoop/tmp
hduser@laptop:~$ sudo chown hduser:hadoop /app/hadoop/tmp
```

Open the file and enter the following in between the ¡configuration¿¡/configuration¿ tag:

```
hduser@laptop:~$ vi /usr/local/hadoop/etc/hadoop/core-site.xml
```

```
<configuration>
 <property>
  <name>hadoop.tmp.dir</name>
  <value>/app/hadoop/tmp</value>
  <description>A base for other temporary directories.</description>
 </property>

 <property>
  <name>fs.default.name</name>
  <value>hdfs://localhost:54310</value>
  <description>
    The name of the default file system.
    A URI whose scheme and authority determine the FileSystem implementation.
    The uri's scheme determines the config property (fs.SCHEME.impl)
    naming the FileSystem implementation class.
    The uri's authority is used to determine the host, port, etc.
    for a filesystem.
  </description>
 </property>
</configuration>
```

4. **/usr/local/hadoop/etc/hadoop/mapred-site.xml:**
   By default, the /usr/local/hadoop/etc/hadoop/ folder contains /usr/local/hadoop/etc/hadoop/mapred-site.xml.template file which has to be renamed/copied with the name mapred-site.xml:

```
hduser@laptop:~$ cp /usr/local/hadoop/etc/hadoop/mapred-site.xml.template
 /usr/local/hadoop/etc/hadoop/mapred-site.xml
```

The mapred-site.xml file is used to specify which framework is being used for MapReduce. We need to enter the following content in between the ¡configuration¿¡/configuration¿ tag:

```
<configuration>
 <property>
  <name>mapred.job.tracker</name>
  <value>localhost:54311</value>
  <description>The host and port that the MapReduce job tracker runs
  at. If "local", then jobs are run in-process as a single map
  and reduce task.
  </description>
 </property>
</configuration>
```

5. **/usr/local/hadoop/etc/hadoop/hdfs-site.xml**

   The /usr/local/hadoop/etc/hadoop/hdfs-site.xml file needs to be configured for each host in the cluster that is being used. It is used to specify the directories which will be used as the namenode and the datanode on that host.

   Before editing this file, we need to create two directories which will contain the namenode and the datanode for this Hadoop installation. This can be done using the following commands:

```
hduser@laptop:~$ sudo mkdir -p /usr/local/hadoop_store/hdfs/namenode
hduser@laptop:~$ sudo mkdir -p /usr/local/hadoop_store/hdfs/datanode
hduser@laptop:~$ sudo chown -R hduser:hadoop /usr/local/hadoop_storeite.xml.t
 /usr/local/hadoop/etc/hadoop/mapred-site.xml
```

   Open the file and enter the following content in between the ¡configuration¿¡/configuration¿ tag:

```
hduser@laptop:~$ vi /usr/local/hadoop/etc/hadoop/hdfs-site.xml


<configuration>
 <property>
  <name>dfs.replication</name>
  <value>1</value>
  <description>Default block replication.
  The actual number of replications can be specified when the file is created
  The default is used if replication is not specified in create time.
  </description>
```

```
  </property>
  <property>
    <name>dfs.namenode.name.dir</name>
    <value>file:/usr/local/hadoop_store/hdfs/namenode</value>
  </property>
  <property>
    <name>dfs.datanode.data.dir</name>
    <value>file:/usr/local/hadoop_store/hdfs/datanode</value>
  </property>
</configuration>
```

### 3.1.7 Format the New Hadoop Filesystem

Now, the Hadoop file system needs to be formatted so that we can start to use it. The format command should be issued with write permission since it creates current directory under /usr/local/hadoop_store/hdfs/namenode folder:

```
hduser@laptop:~$ hadoop namenode -format
```

Note that hadoop namenode -format command should be executed once before we start using Hadoop. If this command is executed again after Hadoop has been used, it'll destroy all the data on the Hadoop file system.

## 3.2 Running Hadoop

```
su hduser
hduser@laptop:~$ ifconfig
hduser@laptop:~$ sudo gedit /etc/hosts
Add you IP address as 172.22.29.225 user user

hduser@laptop:~$ sudo rm -Rf /usr/local/hadoop_store/hdfs/datanode/*
hduser@laptop:~$ sudo rm -Rf /app/hadoop/tmp/*
hduser@laptop:~$ hadoop namenode -format
hduser@laptop:~$ start-dfs.sh
hduser@laptop:~$ jps
        19654 NameNode
        19809 DataNode
        20123 Jps
        20005 SecondaryNameNode
hduser@laptop:~$ start-yarn.sh
```

```
hduser@laptop:~$ jps
        20666 Jps
        19654 NameNode
        20526 NodeManager
        19809 DataNode
        20200 ResourceManager
        20005 SecondaryNameNode

hduser@laptop:~$ /usr/local/hadoop/sbin/mr-jobhistory-daemon.sh start historyserv
hduser@laptop:~$ hadoop namenode -format
hduser@laptop:~$ /usr/local/hadoop/bin/hadoop dfsadmin -safemode leave
hduser@laptop:~$ hdfs dfs -rm -R /usr/local/hadoop/input
hduser@laptop:~$ hdfs dfs -rm -R /usr/local/hadoop/output
hduser@laptop:~$ hdfs dfs -mkdir -p /usr/local/hadoop/input
hduser@laptop:~$ sudo nautilus
        Copy data to sample folder /home/hduser/sample/image.jpg
hduser@laptop:~$ hdfs dfs -copyFromLocal /home/hduser/sample/image.jpg /usr/local
hduser@laptop:~$ hadoop jar example.jar /usr/local/hadoop/input /usr/local/hadoop
hduser@laptop:~$ hadoop dfs -cat /usr/local/hadoop/output/part-0000
```

# Chapter *4*

# System Architecture

## 4.1 Complete Architecture

System provides a command line interface to receive task request. After receiving request, system performs specific operations on the images whether it is upload or feature extraction. Core of this system is divided into two phases. One phase is to upload images and other phase is to extract image features. Both modules are deployed on Hadoop cluster, which follows the MapReduce paradigm.

Upload process will read input data from HDFS, will process it using map and reduce functions, and then write output results to HDFS again. Similarly features extraction process will use map and reduce functions for retrieving similar images depending upon queried image. These images will be given to user on local file system.

### 4.1.1 Upload Images

Upload Images is an independent process where user can upload his image or photo information to the systems database. The process of uploading images is carried out with the help of Map/Reduce techniques which parallelize the process of upload. In upload, there are three sub process. First is splitting of an image, second is feature extraction. Upload Image process is parallelized with the help of Map Reduce technique for each image. When user uploads his private photos or images to the database, he might wants to upload an image or many images at a single point of time. So if user wants to upload many images at a single point of time, he has to provide only path of image folder where images are stored and map stage will parallelize the upload process instead of uploading one image at a time. Architecture of upload process can be seen in figure. For better understanding, explanation of upload process is split into three distinct phases.

These three phases actually specifies image splitting, image feature extraction and encryption part.

*Phase 1:* An image which is going to be uploaded is first split into small images. This step is taken to disperse data across nodes and attacker should not be able to track data from one location. Map/Re-

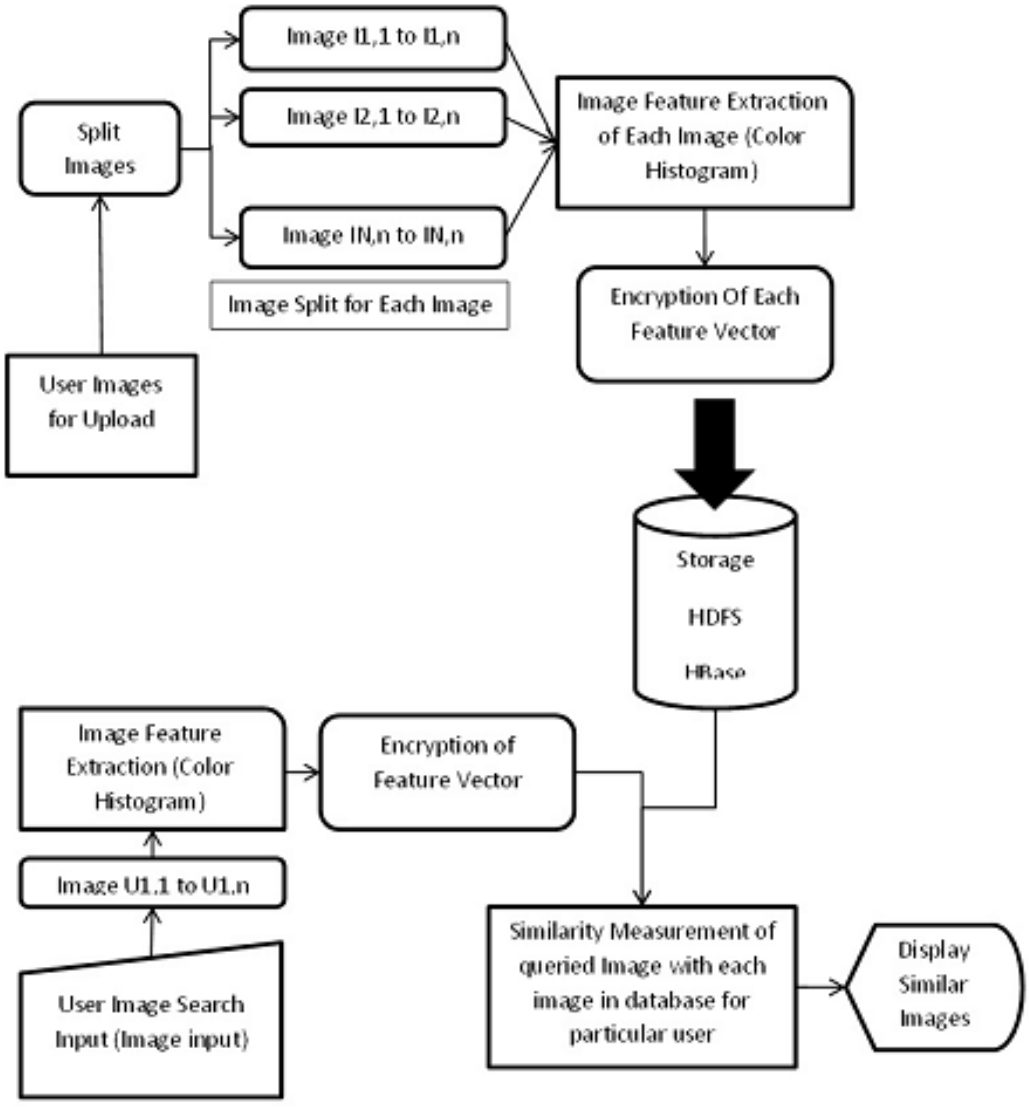


**Figure 4.1** System Architecture

duce technique works on its distributed file system called as Hadoop Distributed File System(HDFS). specifically built for parallelization. All image files are stored into file system. The information about all split images of every image are stored into HBase table. HBase works on top of HDFS. Table in HBase will store path of image and other features of that image.

*Phase 2:* These images are given to the image processing part. In this phase, features of each image is calculated. This includes Histogram, Edge detection (Canny and Sobel), Gaussian blur, Face Detection, rgb2grey conversion etc will provide different feature values such as entropy, intensity, mean, median etc.

*Phase 3:* The values calculated above will be used for the similarity calculation of images i.e. distance between input image and images from HDFS.

## 4.2 Feature Extraction

## 4.3 Image Retrieval

This phase of the system provides user,a CLI to search and retrieve images. Here user will upload one image for which he wants to find similar images. When user wants to search images based on particular image, he uploads it to system. Processes included in upload phase are also carried out on this image. Feature vector of each image is calculated from color histogram. This procedure of searching, comparing and retrieving images is also one Map reduce process.
In this process, when user uploads image, feature vectors of input image i.e. feature vectors of all split image files along with all image feature vectors goes to Map process.

*Map Stage:* This map process multiplies all feature vectors of particular image and emits total feature vector of image. This process does this in parallel for all images present into system. It will create one mapper for one image and emits total feature vector after combining 16 feature vectors.

*Reduce Stage:* Vectors of all images are collected in reduce step. Each vector is compared with feature vector of input image. Comparison is done on the basis of similarity measures. This similarity measure is calculated from the different standard formulas such as Euclidean Distance, Cosine Similarity. In proposed protocol, euclidean distance between color histograms of images is calculated.

## 4.4 System Requirement Specifications

**Hardware:** Minimum one Desktop system with Intels or AMDs processor and 4Gb of RAM. Latest technology will provide better results. i.e. Cluster of nodes having Intels core series processor(i3,i5,i7) of high frequency with 4Gb RAM will definitely perform better than node cluster having older processor version working on less frequency and less RAM. Cluster should have nodes having same configuration.

If it has nodes having different RAM capacity, processor frequency then cluster will not perform up to capacity.

**Software:**

1. Operating System: Any open source Linux operating system.
2. Coding Language: Hadoop map reduce technology/software Java language (JDK)
3. IDE: Eclipse environment

# Chapter *5*

# Implementation

## 5.1 Image Processing

### 5.1.1 Image Format Change

T

```
public void map(NullWritable key, BufferedImageWritable value, Context context)
                throws IOException, InterruptedException
{
    Configuration conf = context.getConfiguration();
    value.setFormat(conf.get("format"));
    context.write(NullWritable.get(), value);
}
```

### 5.1.2 Image Gray Scale Conversion

```
public void map(NullWritable key, MBFImageWritable value, Context context)
                throws IOException, InterruptedException
{
        color_image = value.getImage();
        if (color_image != null)
        {
            gray_image = color_image.flatten();
        fiw.setFormat(value.getFormat());
        fiw.setFileName(value.getFileName());
        fiw.setImage(gray_image);
        context.write(NullWritable.get(), fiw);
    }
}
```

### 5.1.3 Image Blurring

```
public void map(NullWritable key, MBFImageWritable value, Context context)
                throws IOException, InterruptedException
{
        color_image = value.getImage();
        if (color_image != null)
        {
                color_image.processInplace(new FGaussianConvolve(2f,3));
        fiw.setFormat(value.getFormat());
        fiw.setFileName(value.getFileName());
        fiw.setImage(color_image);
        context.write(NullWritable.get(), fiw);
    }
}
```

### 5.1.4 Image Filtering

```
public void map(NullWritable key, MBFImageWritable value, Context context)
                throws IOException, InterruptedException
{
        color_image = value.getImage();
        if (color_image != null)
        {
                color_image.processInplace(new CannyEdgeDetector());
                fiw.setFormat(value.getFormat());
        fiw.setFileName(value.getFileName());
        fiw.setImage(color_image);
        context.write(NullWritable.get(), fiw);
    }
}
```

### 5.1.5 Face Detection

```
public void map(NullWritable key, MBFImageWritable value, Context context)
                throws IOException, InterruptedException
{
        image = value.getImage().flatten();
        if (image != null)
        {
```

```
        FaceDetector<DetectedFace,FImage> fd = new HaarCascadeDetector(40);
        List<DetectedFace> faces = fd.detectFaces(image);
        for( DetectedFace face : faces )
        {
                value.getImage().drawShape(face.getShape(), 3, RGBColour.RED);
        }
        context.write(NullWritable.get(), value);
        }
}
```

### 5.1.6 Color Histogram

```
protected void map(NullWritable key, BufferedImageWritable value, Context context
{
BufferedImage colorImage = value.getImage();
int[] pixel = new int[10];
int[] sbins = new int[256];
IntWritable[] result = new IntWritable[256];
if (colorImage != null)
{
        double d = 0.0;
        double d1;
        int k = 0;
        Color c = new Color(k);
        StringBuilder sb = new StringBuilder();
        for (int x = 0; x < colorImage.getWidth(); x++)
        {
                for (int y = 0; y < colorImage.getHeight(); y++)
                {
                        System.out.println(x + "␣" + y);
                    colorImage.getRaster().getPixel(x, y, pixel);
                    d=(0.2125*pixel[0])+(0.7154*pixel[1])+(0.072*pixel[2]);
                    k=(int) (d);
                    sbins[k]++;
                }
        }
        for (int inte : sbins)
                sb.append(inte).append("␣");
        context.write(NullWritable.get(), new Text(sb.toString()));
```

```
}
}
```

## 5.2 Content Based Image Retrieval

```
public class ImageSearch extends Configured implements Tool {
public int run(String[] args) throws Exception {
    if (args.length != 2) {
    System.err.printf("Usage: %s [generic options] <input> <output>\n",
        getClass().getSimpleName());
    ToolRunner.printGenericCommandUsage(System.err);
    return -1;
}

    Job job = Job.getInstance(super.getConf(), "Image Search");
        DistributedCache.addCacheFile(new URI("/cachefile1"),job.getConfiguration
    job.setJarByClass(getClass());
        job.setOutputKeyClass(NullWritable.class);
        job.setOutputValueClass(Text.class);
        job.setInputFormatClass(TextInputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);
        job.setMapOutputKeyClass(DoubleWritable.class);
        job.setMapOutputValueClass(Text.class);
        job.setOutputFormatClass(TextOutputFormat.class);
    job.setMapperClass(ImageSearchMapper.class);
        job.setReducerClass(ImageSearchReducer.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    return job.waitForCompletion(true) ? 0 : 1;
    }

    public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    int exitCode = ToolRunner.run(conf, new ImageSearch(), args);
    System.exit(exitCode);
    }

    public static class ImageSearchMapper extends Mapper<LongWritable, Text, Doub
```

```
    private MBFImage color_image;
        private Text fileName = new Text();
    private FImageWritable fiw = new FImageWritable();*/
    private final static DoubleWritable one = new DoubleWritable();
        private Text word = new Text();

        public void map(LongWritable key, Text value, Context context) throws IOE
        Path[] cacheFiles = context.getLocalCacheFiles();
        FileInputStream fileStream = new FileInputStream(cacheFiles[0].toString()
        StringBuilder builder = new StringBuilder();
        int ch;
        while((ch = fileStream.read()) != -1){
    builder.append((char)ch);
        }
        String mainfile = builder.toString();
        String line = value.toString();
        String[] tokens = line.split("\t");
        String[] tokens1 = mainfile.split("\t");
        long diff = 0;
        long squaresumdiff = 0;
        word.set(tokens[0]);
        for(int i=1;i<257;i++)
        {
        diff = Integer.parseInt(tokens[i]) - Integer.parseInt(tokens1[i]);
        squaresumdiff = squaresumdiff + diff*diff;
        }
        double diff1 = Math.sqrt(squaresumdiff);
        one.set(diff1);
        context.write(one,word);
}
}
public static class ImageSearchReducer extends Reducer<DoubleWritable, Text, Null
private Text result = new Text();
public void reduce(Text key, Iterable<Text> values, Context context)
throws IOException, InterruptedException {
        String translations = "";
        for (Text val : values) {
    translations += "\t"+val.toString();
```

```
}
        result.set(translations);
        context.write(NullWritable.get(), result);
    }
}
}
```

# Chapter *6*

# Results

## 6.1 Image Processing

### 6.1.1 Image Gray Scale Conversion

**Figure 6.1** Input: Image Gray Scale Conversion

**Figure 6.2** Output: Image Gray Scale Conversion

### 6.1.2 Image Blurring

**Figure 6.3** Input: Image Blurring

**Figure 6.4** Output: Image Blurring

### 6.1.3 Image Filtering

**Figure 6.5** Input: Image Filtering (Canny Edge Detection)

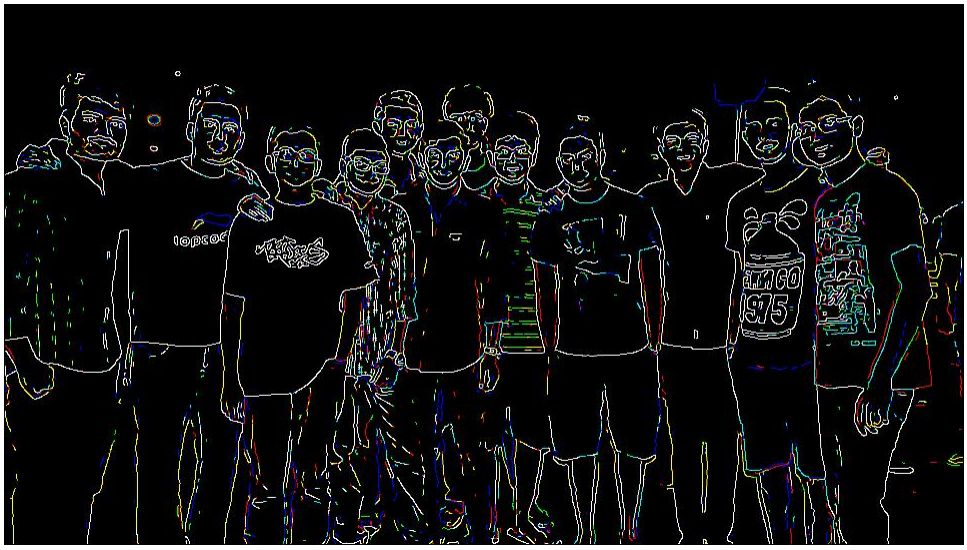

**Figure 6.6** Output: Image Filtering (Canny Edge Detection)

### 6.1.4 Face Detection

**Figure 6.7** Input: Face Detection

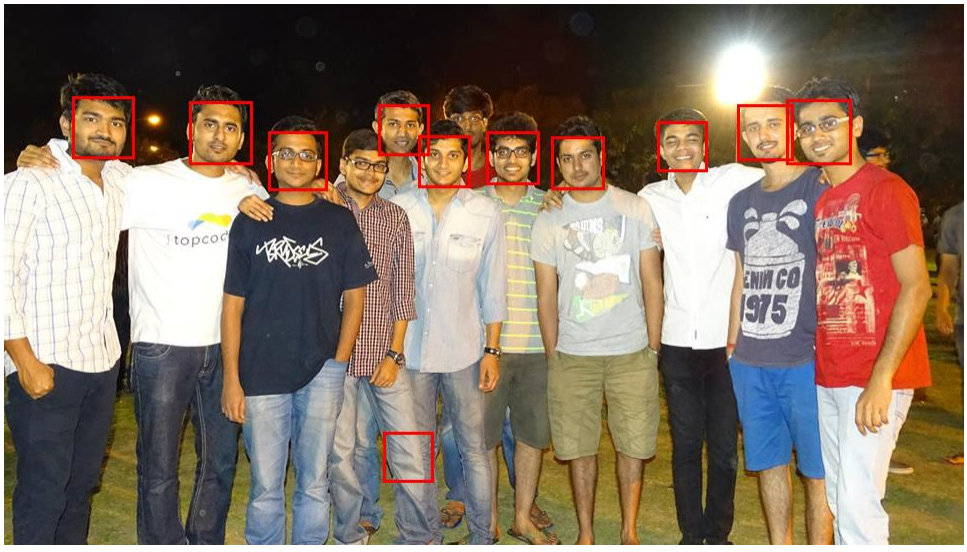

**Figure 6.8** Output: Face Detection

### 6.1.5 Color Histogram

**Figure 6.9** Input: Color Histogram

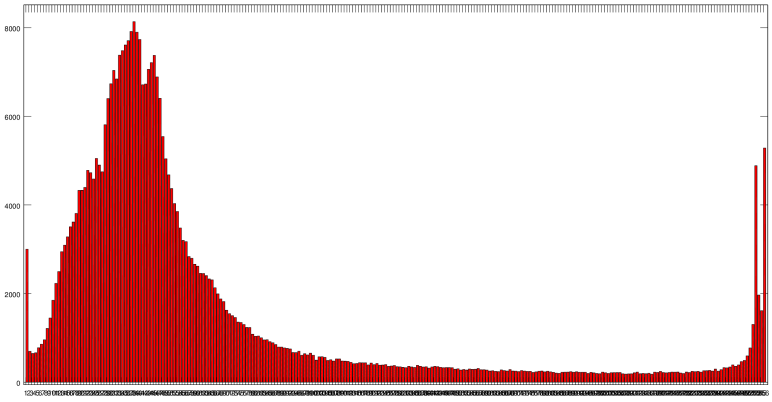


**Figure 6.10** Output: Color Histogram (Red)

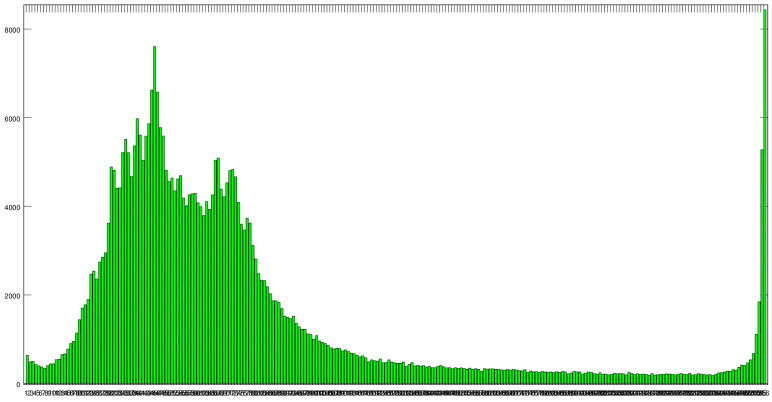


**Figure 6.11** Output: Color Histogram (Green)

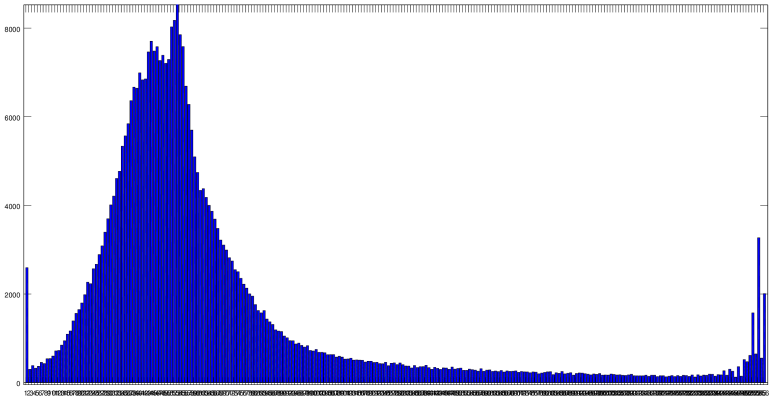


**Figure 6.12** Output: Color Histogram (Blue)

## 6.2 Content Based Image Retrieval

**Figure 6.13** Input: Content Based Image Retrieval

**Figure 6.14** Output: Match Percentage = 100

**Figure 6.15** Output: Match Percentage = 85

**Figure 6.16** Output: Match Percentage = 81

**Figure 6.17** Output: Match Percentage = 78

**Figure 6.18** Output: Match Percentage = 75

# Chapter *7*

# Conclusions and Future Work

## 7.1 Conclusion

We experimented and evaluated the proposed system with different aspects. One is its similarity retrieval, second is execution time complexity.
On similarity measurement front, application is successful in uploading, searching and retrieving similar images by precision of nearly 89 to 94%, for same format and same dimension of image data. Variation in dimension and format varies the output similarity precision.

For execution time complexity, application is tested on single node and multi node clusters configurations. Multi node clusters delivers high performance and increases time eciency nearly by 40%.

Although system has provided satisfactory results, we consider this as just a first step to provide internet/cloud users, a Private CBIR system for their use and it opens avenues for further research and developments in this.

## 7.2 Future Scope

This application is designed in such a way that it will try to cover different domains. It does not use detailed research work from any of the domain. So there is lot of scope to improve this application form different domains perspective. This application can be evolved in Information Retrieval, Content Based Image Retrieval and security domains.

This application has large scope in cloud domain and internet world because every domain servers are using cloud technology and they want to provide their customers a new cutting edge applications for imaging which are efficient

# Chapter *8*

# Bibliography

# Bibliography


[1] DT. Mayberry, E.-O. Blass, and A. H. Chan, Pirmap: Efficient private information retrieval for mapreduce, 2012, http://eprint.iacr.org/

[2] L. Shi, B. Wu, B. Wang, and X. Yan, Map/reduce in cbir application, in Com- puter Science and Network Technology (ICCSNT), 2011 International Conference on, vol. 4, dec. 2011, pp. 2465 2468

[3] P. R. Sabbu, U. Ganugula, S. Kannan, and B. Bezawada, An oblivious image retrieval protocol, in Proceedings of the 2011 IEEE Workshops of International Conference on Advanced Information Networking and Applications, ser. WAINA 11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 349354. [Online]. Available: http://dx.doi.org/10.1109/WAINA.2011.128

[4] J. Zhang, X. Liu, J. Luo, and B. Lang, Dirs: Distributed image retrieval system based on mapreduce, in Pervasive Computing and Applications (ICPCA), 2010 5th International Conference on, 2010, pp. 9398

[5] Y. Rui and T. S. Huang, Image retrieval: Current techniques, promising direc- tions and open issues, Journal of Visual Communication and Image Representation, vol. 10, pp. 3962, 1999.

[6] C.-C. Chen and H.-T. Chu, Similarity measurement between images, in Proceedings of the 29th annual international conference on Computer software and applications conference, ser. COMPSAC-W05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 4142. [Online]. Available: http://dl.acm.org/citation.cfm?id=1890517.1890538

[7] R. B. Gulfishan Firdose Ahmed, A study on different image retrieval techniques in image processing.

[8] J. Eakins, M. Graham, J. Eakins, M. Graham, and T. Franklin, Content-based image retrieval, Library and Information Briefings, vol. 85, pp. 115, 1999.

[9] Mapreduce. [Online]. Available: http://wiki.apache.org/hadoop/MapReduce

[10] Apache tomcat. [Online]. Available: http://tomcat.apache.org/

[11] Apache tomcat. [Online]. Available: http://en.wikipedia.org/wiki/Apache Tomcat

[12] Java server pages overview. [Online]. Available: http://www.oracle.com/technetwork/java/overview-138580.html

[13] Eclipse. [Online]. Available: http://www.eclipse.org/ 51[21] Java platform, enterprise edition. [Online]. Available: http://en.wikipedia.org/wiki/ Java Platform, Enterprise Edition