Content Based Image Retrieval for Big Visual Data using Map Reduce

Neeraj Kumar

A Thesis Submitted to

Indian Institute of Technology Hyderabad

In Partial Fulfillment of the Requirements for

The Degree of Master of Technology

Department of Computer Science and Engineering

June 2015

Declaration

I declare that this written submission represents my ideas in my own words, and where ideas or words of others have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be a cause for disciplinary action by the Institute and can also evoke penal action from the sources that have thus not been properly cited, or from whom proper permission has not been taken when needed.

_____

(Signature)

_____

(Neeraj Kumar)

_____

(Roll No.)

Approval Sheet

This Thesis entitled Content Based Image Retrieval for Big Visual Data using Map Reduce by Neeraj Kumar is approved for the degree of Master of Technology from IIT Hyderabad

---

(Dr. C Krishna Mohan) Adviser
Associate Professor
Dept. of Computer Science and Engineering
IIT Hyderabad

Dedication

I dedicate my thesis to my parents for whom a educated child is more important than hefty amount of money in bank account.

Abstract

Keywords : *Content based image retrieval, clustering, k-means, top-k, data sets, MapReduce framework, distributed computing model, apache hadoop , cyclic redundancy check.*

With high availability of portable and low cost digital cameras and improvement in image capture technology, huge amount of visual data (photos and videos) is being generated everyday. The processing capability of standalone devices is insufficient to handle such massive data also known as big data. Apache Hadoop, a distributed computing model that is based on MapReduce framework is an easy to use solution for managing big data with freely available implementation models. Hadoop is mainly designed for cost-effective commodity hardware or inexpensive cloud computing infrastructure. Hence, access to expensive hardware or in-depth understanding of parallel programming is no longer required to work on big data. A hadoop cluster comprises of a master node and many computing nodes (slaves). The master node's main purpose is to interact with users and monitor the status of the slaves, keep track of load balancing and other background tasks. Such a cluster is well capable of working independently for many trivially parallelised processes which makes it suitable for the task of clustering data. The focus of this Thesis is content based image retrieval in the context of visual big data where clustering is a necessary preprocessing step. The input images are organised into clusters by using standard clustering algorithms like k-means implemented in hadoop. While searching, the top-K similar images are retrieved from the cluster closest to the query image. The performance of the setup was measured on two data sets with 397 image categories. The first one had 1,08,679 images and the second one comprised of 80 million tiny images. It was shown that using hadoop, significant speedup on search and retreival can be obtained on such large datasets.

Contents

Chapter 1

Introduction

With the rapid growth of information technology, a new class of problems has emerged. As we have achieved more computation and processing power we are able to tackle more intense tasks efficiently. However the scale of data to be processed is increasing exponentially. To have an idea about scale of data, let's look at data of famous social networking site 'Facebook' : It handles 300 million photos each day (2012 stats). It is virtually impossible (also inefficeint) to store and process such large amount of data on single machine. Solution to this problem can be storing and processing data in distributed manner.

In recent time smartphones are able to solve some problems which was not possible even for supercomputers many years ago. However as computer architectures are moving close to some physical limitations, systems that are distributed are getting more and more popular. The main reasons to which popularity of distributed computed can be attributed are : i) Physical limitations of processors ii) Scalability iii) Fault Tolerance iv) Latency.

Currently there are many popular ways for parallal computation like multi-core GPU's, Map Reduce etc. Both approaches have their own advantages and disadvantages in terms of speed, portability and cost.

With the exponential growth in amount of data, it is becoming more and more complex task to find the relevant content. Let's take example of google images : Google database contains a huge amount of images. While searching with a image or just a text phrase (signifying class of images), it will return relevant images. Such huge databases can not be stored and processed on single machines. There is a need to store and process this data in a distributed manner. Specifically speaking Google uses GFS to store it's content over the servers.

1.1 Problem Statement

Content based image retrieval(CBIR) is about to get the similar content images from the database to a image query. In this thesis we tried to implement Content based image retrieval in detributed

framework using Hadoop. Our intention is to speedup the performance of content based image retrieval, starting from the feature extraction to image searching.

1.2 Content Based Image Retrieval

Content-based image retrieval (CBIR) is the application of computer vision techniques to the image retrieval problem, which involves searching visually similar images to a query image from a large pool of data.

Visual contents like color,texture, shape of images are used by CBIR system to identify the similarity. Generally in CBIR the visual contents of the images in the database are extracted and described by multi-dimensional feature vectors. These feature vectors form a feature database. Query image provided by user to the CBIR system to retrieve the similar images. Distances between the feature vectors of the query image and images present in the database are calculated and retrieval is carried out with the help of indexing scheme. [1]

1.2.1 Traditional Content Based Image Retrieval Model

Content based image retrieval [CBIR] is mainly about developing a technique that extract the similar images to a query image based on their content from a large data set. Query image is given as input to the CBIR system, then the system will process the query and the images from data set and retrieve similar images as output.

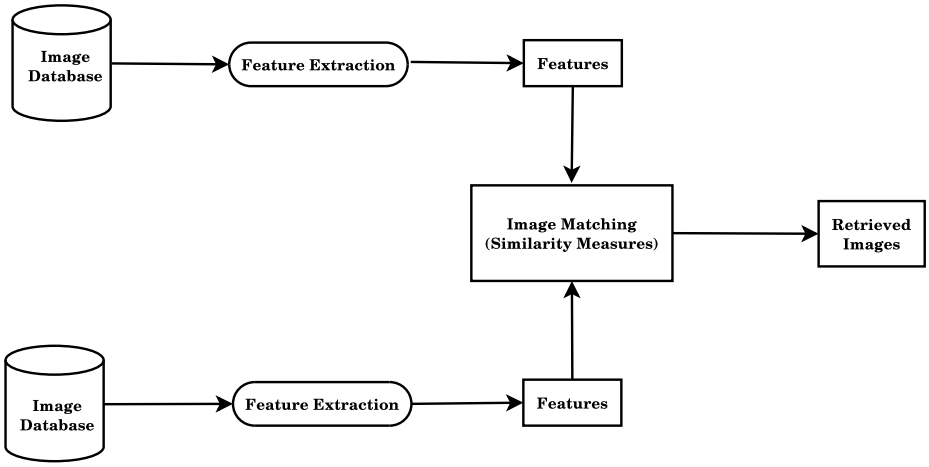


Figure 1.1: CBIR Process Model.

Retrieval of similar content images is based on processing of features. An image can have many features like color, texture, edge information etc.

1.2.2 Image Content Descriptors

A good visual content descriptor should be insensitive to the variation of the illuminant of the scene. Between the invariance and the discriminative power of visual features,There is a tradeoff, since a very wide class of invariance loses the ability to discriminate between essential differences. Invariant description has been largely investigated in computer vision (like object recognition), but is relatively new in image retrieval [2] [1].

1.2.2.1 Color

Color is a important feature and many popular use this feature. Image can be a RGB (Red, Green, Blue) color space or spaces such as HSV (Hue, Saturation, and Value). Differences in color spaces are very much close to the differences in color that human recognise. There are many color spaces are available which are beneficial for different purposes. However color feature is very much dependent on illuminance, shadows etc.

1.2.2.2 Texture

Another important feature for CBIR is texture. Visual patterns with the properties of homogeneity that do not occur from the presence of only a single color or intensity can be considered as texture.Texture representation methods can be defined into two categories: 1).structural 2). statistical. In CBIR systems coarseness, and regularity information provided by texture content is very useful.

1.2.2.3 Shape

Edge information and corner (intersection of edges) is very much suitable for CBIR. Edge directionn and gradient magnitude are well tested features of image to get the similar content in CBIR system. We have used Histograms of oriented gradients (HOG) in our system to get the edge direction feature of images.

1.2.3 Retrieval of Similar Content Images

Images organised in clusters so most similar images grouped in same cluster and can be differentiated with other images. For making cluster of images we have used k-means clustering algorithm based on euclidean distance. TreeMap [3] algorithm used to get top k number of imgaes from the cluster based on content similarity.

1.2.3.1 TreeMap

TreeMap is very similar to the HashMap because it stores key, value pairs. Difference between both is treemap sort data in ascending order. Natural ordering is used for the sorting purpose. TreeMap implementation takes log(n) time to do operations like **Put,Get,Remove** etc. TreeMap is based on red-black tree operations.

To understand the operations of red-black tree, the knowledge of it's properties is very essential. Description of properties given below -

- Color of every node will be red or block.

- Root must be black color
- Red node can not be neighbor
- From root to leaf all the paths should have same number of black nodes.

Chapter 2

Overview of approaches for Content Based Image Retrieval

There is a lot of work has been done in field of content based image retrieval but not by using MapReduce framework.

In Web-Scale Computer Vision using MapReduce for Multimedia Data Mining [4], Brandyn White et al. present a case study of classifying and clustering billions of regular images using MapReduce. No mention is made of average image dimensions or any issues with not being able to process certain images because of memory limitations. However, a way of pre-processing im- ages for use in a sliding-window approach for object recognition is described. Therefore one can assume that in this approach, the size of images is not an issue, because the pre-processing phase cuts everything into a manageable size. The question still remains whether a sliding window approach is ca- pable of recognizing any objects present in the image that do not easily fit into one analysis window, and whether the resource requirements for image classification and image processing are significantly different or not. An Architecture for Distributed High Performance Video Processing in the Cloud [5] by Rafael Pereira et al. outlines some of the limitations of the MapReduce model when dealing with high-speed video encoding, namely its dependence on the NameNode as a single point of failure, and lack of possibility for the more sophesticated issues. An alternative - optimized - implementation is proposed for providing a cloud-based IaaS (Infrastructure as a Service) solution. However, considering the advances of distributed computation technology within the past two years and the fact that the processing of large images was not touched upon, the problem posed in this work still remains.

A description of a MapReduce-based approach for nearest-neighbor clustering by Liu Ting et al. is presented in Clustering Billions of Images with Large Scale Nearest Neighbor Search [6].

In Parallel K-Means Clustering of Remote Sensing Images Based on MapReduce [7], Lv Zhenhua et al. describe using the k-means algorithm in conjunction with MapReduce and satellite/aerophoto images in order to find different elements based on their color (i.e. separate trees from build- ings). Not much is told about encountering and overcoming the issues of analyzing large images besides

mentioning that a non-parallel approach was unable to process images larger than 1000x1000 pixels, and that the use of a MapReduce-based parallel processor required the conversion of TIFF files into a plaintext format.

Case Study of Scientific Data Processing on a Cloud Using Hadoop [8] from Zhang Chen et al. describes the methods used for processing sequences of microscope images of live cells. The images and data in question are rela- tively small - 512x512 16-bit pixels, stored in folders measuring 90MB - there were some issues with regard to fitting into Hadoop DFS blocks which were solved by implementing custom InputFormat, InputSplit and RecordReader classes. No mention was made about the algorithm used to extract data from the images besides that it was written in MATLAB and MapReduce was only involved as a means distribute data and start the MATLAB scripts for processing.

While the above shows that there has been a lot of work in this area the question still remains : Is Hadoop well suited for large scale visual recognition tasks, because as evidenced by this brief overview, there are only a few cases where image processing has been done with MapReduce.

Chapter 3

Hadoop Architecture and Programming Model Used for CBIR

3.1 Why Hadoop?

Hadoop is well known for its Map Reduce and distributed file system and the system well tested for its feasibility to process images. Data processing in batches on PC is feasible but only for small amount of data and computation will suffer since only a part of this data fits into memory at given time and also slow hard drive access. Now solution lies in running the data simultaneously on several computers but it require job monitoring, mechanisms for data distribution and means to ensure that the processing completes even when some computers experience failure in process. Apache Hadoop was designed to solve. Another approach is treating the problem like a traditional large-scale computing task which requires specialized hardware and complex parallel programming. Cluster computers built on graphics processing units (GPU) are an example of this. It is interesting to know whether the same issues can be tackled with simpler and cheaper systems without much decrease in efficiency.

3.2 Hadoop Processing

3.2.1 Hadoop Cluster

Cluster mainly consist of master computer and any number of computing nodes purpose of master is to interact with users and monitor the status of computing nodes, keep track of load balancing and other background tasks[9][10]. Computing node deals with storing and processing of data. Execution happens in MapReduce in following manner-

- User uploads input data to Hadoop distributed file system (HDFS), which in turn distributed and stored among computing nodes [11].

- Users starts the job by specifying MapReduce programme to execute along with input and output path and other parameters.
- Master node sends a copy of programme along with its parameters to every computing nodes and starts execution.
- Computing nodes starts the map phase first and process data in their local storage, fetching more data from other computers if necessary and possible (based on master node decision).
- After all map tasks are finished, there output is sorted in a way, that for every distinct key reduce task process all pairs with that key.
- Once reduce phase finished and its output written back to HDFS the user can retrieves the resulting data.

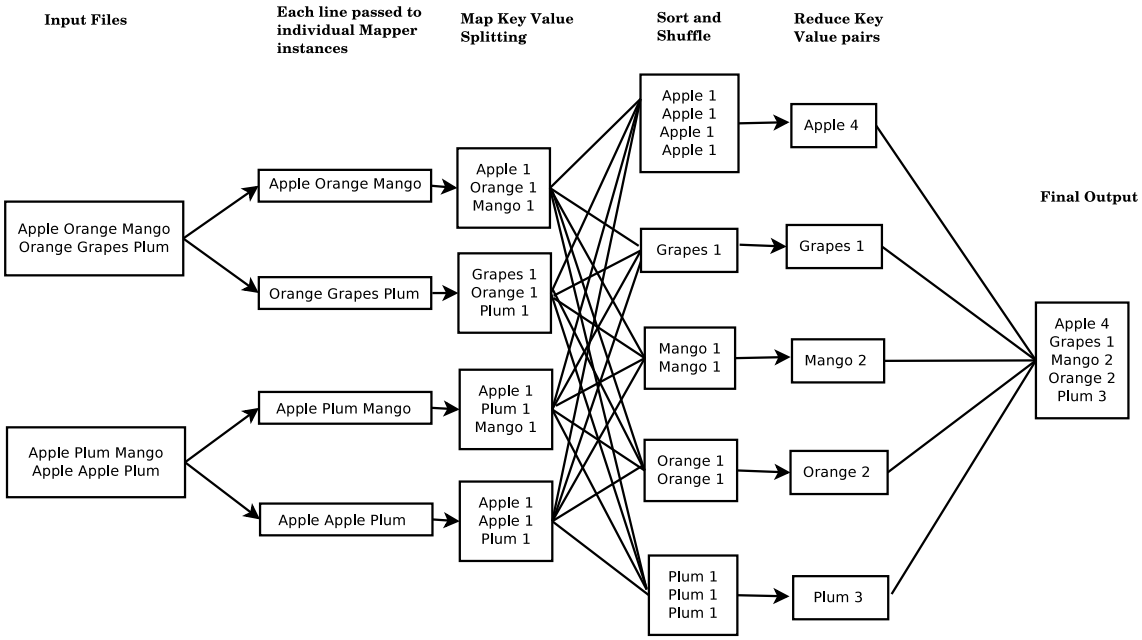
Hadoop provides a fairly straightforward implementation of the MapReduce model. In order to write a complete a MapReduce job, a programmer has to specify the following things:

- An InputFormat class, which handles reading data from disk and converting it to Key-Value pairs for the Map function.
- A Mapper class, which contains the map function that accepts the $\langle Key, Value \rangle$ pairs from InputFormat and outputs $\langle Key, Value \rangle$ pairs for the Reduce function.
- A Reducer class with a reduce function that accepts the $\langle Key, Value \rangle$ pairs output from the Mapper class and returns $\langle Key, Value \rangle$ pairs.
- An OutputFormat class, which takes $\langle Key, Value \rangle$ pairs from the Reducer and writes output to disk.

3.2.2 MapReduce

MapReduce is a programming model for data processing. MapReduce works by breaking the processing into two phases: the map phase and the reduce phase. Each phase has key-value pairs as input and output, the types of which can be chosen. The programmer also specifies two functions: the map function and the reduce function. I will focus on describing the general philosophy and methodology behind this model. MapReduce computation can be describe as following [11]:

- Input from the disk in $\langle Key, Value \rangle$ pair.
- Map function process each pair separately and gives out put in $\langle Key, Value \rangle$ pair.
- For each distinct key, the Reduce function processes all $\langle Key, Value \rangle$ pairs with that Key, and - similarly to Map returns Key- Value pairs.
- After processing output of reduce function written into disk in $\langle Key, Value \rangle$ pair.



3.2.3 Hadoop Distributed File System (HDFS)

Hadoop Distributed File System is integral part of hadoop cluster. Its purpose is to provide fault tolerant storage structure capable of holding large amount of data. It provides fast access of data and provide a way for MapReduce to perform computations on same location as the data.

Importance of HDFS with respect to image processing is approach of storing files in block storage. Default block size in HDFS is 64 MegaBytes, but it can be customized. There are reasons for this kind of design. Firstly, as the blocks are written to physical storage in a contiguous manner, they can also be read with minimal disk seeking times. Secondly, file system is geared towards storing very large files, a larger block size ensures that storage of meta-data such as read/write permissions and physical locations of individual blocks creates less overhead.

Block size is somewhat important with regard to processing images, since if an image that is too big is uploaded to HDFS, there is no guarantee that all of its blocks would be stored in the same physical location. Since a Map or Reduce task would then have to retrieve all of its blocks before processing, the idea behind executing tasks that are local with regard to the data is lost, the speed of reading input data now depends on the network. Therefore, in order to ensure optimal processing speed, images should fit inside the HDFS block size. This is not a problem with most regular images, as it is easily possible to configure the cluster with a block size of even 128 megabytes or more, however increasing this parameter past a certain point may not have the desired effects. Also, as discussed before, processing very large images sets considerable memory requirements to the computers. For these reasons, splitting large images into manageable parts is the best solution.

On the other hand, when dealing with a data-set of many small images, simply uploading them to HDFS results in the creation of a separate block for each file. Since a given Map or Reduce task operates so that it uses its defined InputFormat to read data one block at a time, having many small blocks increases the overhead with regard to these operations. In these cases, it is standard practice to first store the data using a SequenceFile. Drawback: There is a caveat, however, with

regard to the files that are located on the "edge" of the split. In order to illustrate this, I uploaded a SequenceFile with 3 images - 30 megabytes each - to HDFS with a configured block size of 50 megabytes Quering the uploaded file with the Hadoop fsck tool, I found that instead of writing the file as three blocks, each containing a full image, it was split into two blocks, so that one image ended up divided into two. This could negatively affect the performance of a job, since a Map or Reduce task would need to read both blocks to assemble the full image. If I set block size 64 MB then there will be no problem in reading image from a single block and there will be no overhead so optimal block size is important to get the best out of Hadoop.

3.3 Data Integrity in Hadoop

It is very necessary to preserve the integrity of data in overall processing to make the system reliable. It is rightly said that hadoop provides data integrity. It uses the cyclic redundancy check 32 (CRC-32). Hadoop calculates checksum of input data and also also claculate checksum of data after data passess through any network. If checksum calculated have no difference that means there is no corruption in data and hadoop preserves integrity.[11]

3.3.1 Data Integrity In HDFS

HDFS calculates checksum before reading the data and also calculate the difference to make sure that data is not corrupted. Checksum is calculated for every byte of data and default size is 512 bytes. Actually CRC-32 has size of 32 so overall overhead is approximately less than 1 percent.

Primary responsibility of calculating and checking the validity of checksum is of datanodes. Datanodes recieve data from clients and other datanodes during replication then datanode check the integrity of data by calculating checksum. Each datanodes maintains the list of checksum that it had calculated previously. Each datanode runs **DataBlockScanner** to scan each block in background and make sure that every byte of data is error free. Some replicas of blocks maintained by HDFS so if any block gone corrupted it can be replaced by new one that can be produced by replicas.[11]

3.3.2 Client Side Data Integrity

Client side calculation of checksum is very important to make sure that data is noncorrupt. Client side data checksum is calculated by hadoop **LocalFileSystem**. For example if we give a name to a file like featuredata then local file system automaticaly create one another file name featuredata.crc that will contain all checksum calculation of every chunk of that file.

To calculate the checksum **LocalFileSystem** uses **ChecksumFileSystem**. When ever ChecksumFileSystem detects the erroe it calls its report function.

3.4 Small File Problem

The dataset that we are using have all the small images. In hadoop small size of file means the size of file is significantly less thah the default size of block in HDFS. 64 MB is the default size of block in HDFS. Size of images in our dataset is in range of kilobytes only.

3.4.1 Small File Problem with HDFS

Hadoop works better with a small number of large files than a large number of small files. One reason for this is that FileInputFormat generates splits in such a way that each split is all or part of a single file. If the file is very small (small means significantly smaller than an HDFS block) and there are a lot of them, then each map task will process very little input, and there will be a lot of them (one per file), each of which imposes extra overhead.[11]

Example : Every file, directory and block in HDFS is represented as an object in the namenodes memory, each of which occupies 150 bytes. So 10 million files, each using a block, would use about 3 gigabytes of memory. Scaling up much beyond this level is a problem with current hardware. Certainly a billion files is not feasible.

3.4.2 Small File Problem With MapReduce

There is also a small file problem with Map Reduce, Map will process very little data because of small size of files, so there will be a lot more map task, each of which will impose extra overhead.

Example : Compare a 1GB file broken into 16, 64MB blocks, and 10,000 or so 100KB files. The 10,000 files use one map each, and the job time can be tens or hundreds of times slower than the equivalent one with a single input file.

3.4.3 Solution : Sequence Files

Idea here is that use the file name as the key and file content as the value [12]. Going back to the 10,000, 100KB files, write a program to put them into a single SequenceFile and then process them in a streaming fashion (directly or using MapReduce) operating on the SequenceFile. A sequence file is a persistent data structure for binary key-value pairs. Sequence files have sync points included after every few records that align with record boundaries, aiding the reader to sync. The sync points support splitting of files for mapreduce operations. Sequence files support record-level and block-level compression. SequenceFiles are splitable so MapReduce can break them into chunks and operate on each chunk independently. They also support compression. It is perfectly possible to create a collection of SequenceFiles in parallel.

3.5 Iterative Algorithms in Hadoop

For any programme to execute hadoop initialize itself. To initialize cluster master node destribute the copy of programme to every datanode and also regulate the memory management in the cluster. Task of initialization takes some time, this time depends upon the strength of network like bandwith and organization of network and the time also depends upon the number of nodes in the cluster.

As the number of nodes increases the task of replication of data for the master node and network congestion also increses. So larger the cluster, larger is the initialization time.

After every iteration hadoop stores data in hard drive and in next iteration hadoop will initialize itself (In our case in 6 node cluster it takes 9 to 10 seconds for initialization) and take data from hard drive as input so it becomes time consuming affair. If the data is small and aglorithm used in process require many iteration then hadoop will initialize itself many time and overall implementation becomes inefficient. So if data is big and algorithm needs less number of iteration then hadoop implementation will surely give efficient result. This shows that hadoop is designed only for bigdata problems.

Chapter 4

Features Used for Clustering and Image Retrieval

We have used two features namely Histograms of Orientations (HOG) and Scale Invariant Feature Transformations (SIFT) to identify the similarity among the pictures. We extracted both the features together and stored in sequence file having image name as a key and features as value. Dimensions of a feature is separated by commas so that it is easier to use it in next steps.

4.1 Histograms of Oriented Gradients (HOG)

4.1.1 Introduction

HOG features, proposed in 2005 descriptors are a widely used technique for object recognition in field of computer vision. Main idea behind HOG descriptors is that local object appearance and shape within an image can be described by the distribution of edge direction (intensity gradients). Histogram of gradient direction is computed by dividing image into cells. The algorithm is explained through a simple example below.

4.1.2 Algorithm

- **Gradient Computation :** Divide image in to cells. Compute horizontal and vertical gradients using derivate masks.
- **Orientation Binning :** Compute gradient orientation and magnitude. Based of strategy of weighted votes cell histogram is calculated.
  - For color image pick the color channel with highest gradient magnitude for each pixel.
- **Descriptor Blocks:** Gradient strengths are locally normalized to tackle affects change in contrast and illumination. It is done by grouping the cells together into larger, spatially connected blocks. The HOG descriptor is then the concatenated vector of the components of the normalized cell histograms from all of the block regions.
- **Block Normalization :** Finally blocks are normalized using l_k norm.

Example :

Consider an image with size 64x128

Divide the image in 16x16 blocks of 50% overlap

7x15 = 105 blocks in total.

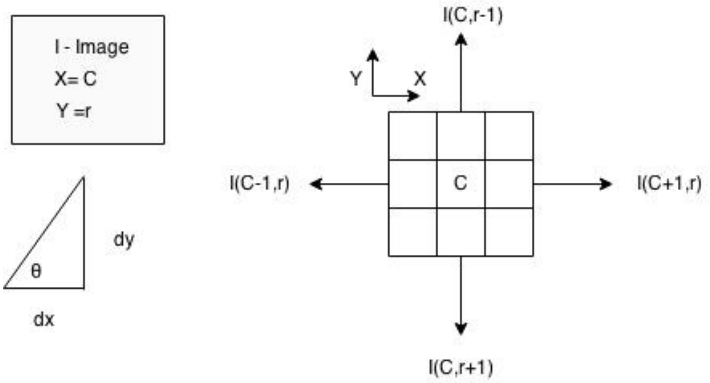
Each block will consist of 2x2 cell with size 8x8.

Quantize the gradient orientation into 9 bins.

Bi-linear interpolation between bin centers in its neighborhood.

Vote is used with Gaussian to downweight the pixel center near the edge block.

Concatenate the histograms (Feature dimension is : 105x4x9)



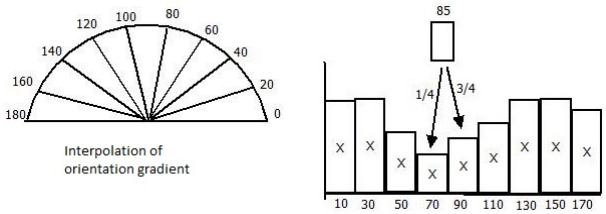
$dy = I(C, r-1) - I(C, r+1)$

$dx = I(C+1, r) - I(C-1, r)$

$Gradient\ \ Magnitude\ \ (GM) = \sqrt{dy^2 + dx^2}$

$Gradient\ \ Orientation\ \ (GO) \quad \theta = tan^{-1}(\frac{dy}{dx})$

Example : if $\theta = 85$ Distance to the bin center bin 70 and bin 90 are 15 and 5 degree respectively Hence ratios are $\frac{5}{20} = \frac{1}{4}$ and $\frac{15}{20} = \frac{3}{4}$



4.2 Scale Invariant Feature Transform (SIFT)

There exist some algorithm like Harris Corner Detection, which are rotation invariant i.e. even if the image is rotated, this algorithm will still detect same corners. But if image is scaled up or down, Harris Corner Detection algorithm may fail i.e. it is scale invariant. Let's understand this through an example : When the image is scaled up the curvature of edges usually get changed. So the corner may not fit in the same window which was used earlier to detect it. In 2004, D. Lowe came up with an new algorithm Scale Invariant Feature Transform(SIFT) [**?**]. It focus on extracting Distinctive Image Features (desciptors) from Scale Invariant Points (keypoints).

4.2.1 Scalespace Extrema Detection

For corner detection window with same size can not be used, as we will need variable size window to detect a corner. For this purpose scalespace filtering is used. Here corners are detected at various scales. Laplacian of Gaussian operator is used to detect blobs, with variation in parameter corner of various size are detected. So, we find the local maximas' across the scale and space which generates a list of values which means there is a potential keypoint at (x,y) at scale. Sometimes Difference of Gaussian is used as an approximation as LoG is costly.

4.2.2 Keypoint Localization

In previous step list of potential keypoints was generated, to get more accurate results a threshold is use to refine the list. Taylor series expansion of scale space is used for this purpose. It basically removed low contrast keypoints.

4.2.3 Orientation Assignment

Next step is to assign direction to each keypoint. A neighborhood is taken around the keypoint location depending on the scale, and the gradient magnitude and direction is computed. Histogram is created with 36 bins which covers 360 degrees. This step contributes to stability of matching. It is weighted by gradient magnitude and gaussianweighted circular window with equal to 1.5 times the scale of keypoint. The highest peak in the histogram is taken and any peak above 80 percent of it is also considered to calculate the orientation.

4.2.4 Keypoint Descriptor

Next step is to get keypoint descriptor. A 16x16 neighborhood is considered around the keypoint. It is divided into 16 subblocks of 4x4 size. For each subblock, histogram with 8 bins is used. Hence total 128 bins are available, a vector is used for representation. In addition to this, several measures are taken to attain robustness against factors such as rotation, variation in illumination etc.

4.2.5 Matching

To identify nearest neighbors, only keypoints need to be matched. In some cases, when matches are very close, then ratio of closest distance to second closest distance is used. If that is more than 0.8, they are skipped.

Chapter 5

Implementation

5.1 Introduction

We have implemented our process in two phases. In preliminary phase data is preprocessed and then features are extracted. These features are stored as clusters which are created based on similarity of images. In phase II, given a query image we have to search similar images based on the content. First the most close cluster is detected and then top k most similar images are returned as result.

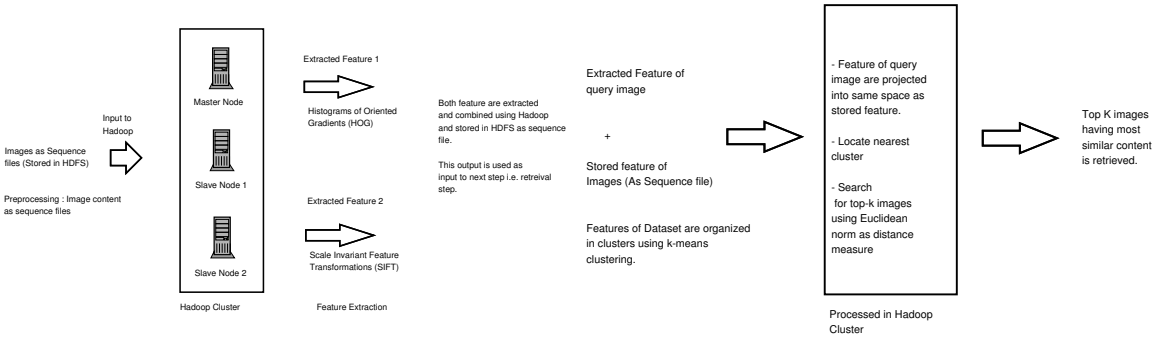


Figure 5.1: Phase 1.

Figure 5.2: Phase 2.

5.2 Preprocessing

5.2.1 Dataset

We are using SUN397 for this purpose [13]. This dataset contains 108679 images of variable size belonging to 397 classes [14] .

Database Size = 39 GB

Number of classes = 397

Total number of mages = 108679

5.2.2 Ideal input for Hadoop : Sequence File

As we have discussed the problem of small files with HDFS and MapReduce in previous section and we also mentioned the one of the solution lies in sequence files. Taking all the images into consideration we created sequence file having image name as a key and binary content as value. It took around 2 hour and 22 minutes. Although it can be generated in parallel in cluster but we created in a single node. Now onwards we will take this sequence file as a input and also we will store output in sequence file.

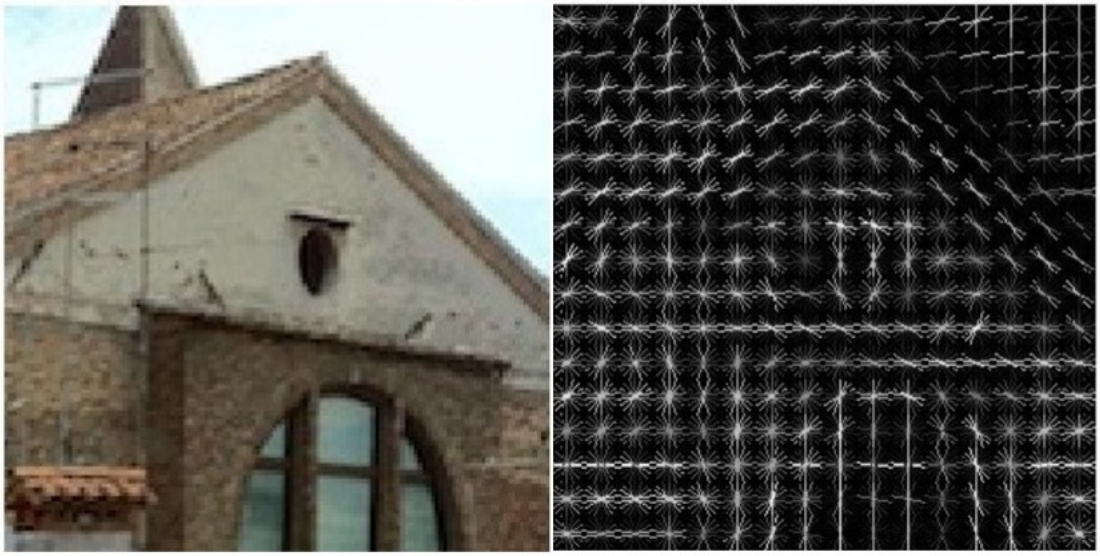
5.3 Feature Extraction

5.3.1 HOG

To get orientation gradient of the images HOG features have been extracted. These features captures edge direction along with depth by which image has been taken. Edge direction is proven to be suitable to identify similarity among images. Following things we have considered while extracting the HOG features :

Size of block : 2x2
Number of blocks : Variable because of variable size of images
Feature vector size : 6300 (maximum)
Number of bins : 9

To visualize HOG features :



5.3.2 SIFT

To get Scale Invariant Feature Transformation interest points and their descriptors are required. As we have already discussed SIFT features are scale and rotation invariant. have considered while extracting the HOG features :

Size of Feature Vector : 6300 (maximum)

Figure 5.3: Query Image.

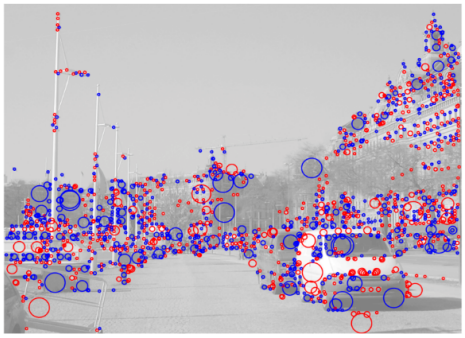


Figure 5.4: Retreived Images.

5.4 Dimensionality Reduction

As we saw in earlier section, that due to variation in dimension of images, feature vector may vary too. So feature vector size is restricted to 6300 dimension to maintain a constant vector size for easier computations.

'Curse of Dimensionality' [15] is a well known problem in computer vision. Higher dimension of feature vectors doesn't provide full proof solution for discrimination among classes. So we have used Principal Component Analysis (PCA) technique to reduce the dimension of feature vectors.

We have used PCA with an goal of maintaining 95% of variance in the data i.e. Data in lower dimension have 95% variance with respect to origninal data. 455 dimensions for SIFT and 976 dimensions for HOG were selected to maintain 95% variance.

While implementing PCA we have used approach with eigen vectors rather than SVD. The k components with highest k eigen values are selected for new k-dimension space. Both training and testing images were projected in this space for later.

5.5 Clustering of Images

To make the searching more efficient we divided the data into clusters. Clustering is mainly about grouping data on the basis of selected features. We organised data in k number of groups where k is a positive integer number.

5.5.1 Clustering Algorithm : K-Means Clustering

Motivation behind k-means clustering is to grouping data in clusters. Here we are clustering images in k groups according to similarity in features.The grouping is done by minimizing the sum of squares of distances between data and the corresponding cluster centroid.

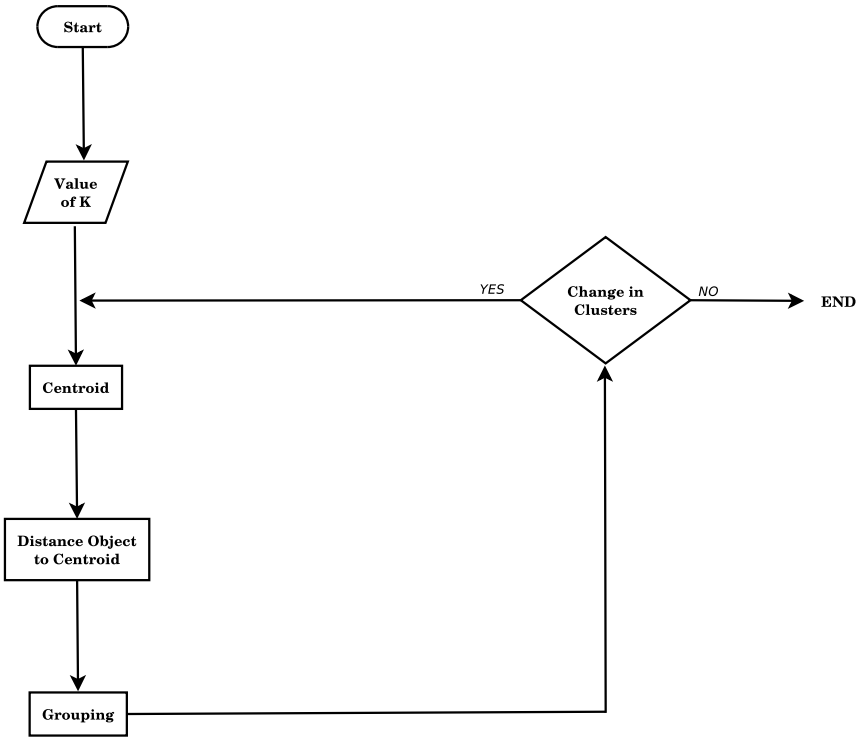
Initially k centres are randomly choosen, where k is number of clusters

Iterate until last (No more changes in clusters):

(i) *Centroid coordinate Determination.*

(ii) *Calculate distance between each object and the centroids.*

(iii) *Based on minimum distance, group the object* (find the closest centroid).



Example : Lets take a example to understand the working of k-means clustering. weight and lenght of a commodity are displayed in table , weight and length are considered as features.

Table 5.1: Commodity Table

Commodity	Weight	Length
A	1	1
B	2	1
C	4	3
D	5	4

Iteration (0) : Here we consider the features of commodity A and commodity B as initial center (C1 (1,1) and C2 (2,1)).

Now calculate distance from C(4,3) to C(1,1).

$$\sqrt{(4-1)^2+(3-1)^2} = 3.61$$

$Also calculate distance from C(4,3) to C(2,1)$.

$$\sqrt{(4-2)^2+(3-1)^2} = 2.81$$

The *Object centroid distance* calculated as

$$D(0) = \begin{vmatrix} 0 & 1 & 3.61 & 5 \\ 1 & 0 & 2.83 & 4.24 \end{vmatrix}$$

Now The *Object clustering* written as

$$D(0) = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \end{vmatrix}$$

Iteration (1) : Now in this iteration we calculated new centers these new centers are C1(1,1) and C2(2+4+5/3, 1+3+4/3). So the two bcenters are C1(1,1) and C2(11/3,8/3).

As illustrated in iteration (0) again we can get the distance of object from the center and we can construct matrix on calculated values.

The *Object centroid distance* calculated as

$$D(0) = \begin{vmatrix} 0 & 1 & 3.61 & 5 \\ 3.14 & 2.36 & 0.47 & 1.89 \end{vmatrix}$$

Now The *Object clustering* written as

$$D(0) = \begin{vmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{vmatrix}$$

If the object clustering matrix of this iteration is same as previous which indicates clusters are stable.

Chapter 6

Experimental Results

6.1 Time Required to Extract Image Features

Both features (HOG and SIFT) have been extracted using hadoop. Statistics are given in table below.

Cluster	Start Time	End Time	Total Time	Splits	Maps	Reducer Tasks
5 Node	0:47:55	0:59:10	0:11:15	5	5	1
6 Node	10:43:14	10:54:21	0:11:07	5	5	1

6.2 Time Required for Clustering

Clustering has been done by using k-means clustering algorithm. We organised data in many k showing number of images present in one cluster.

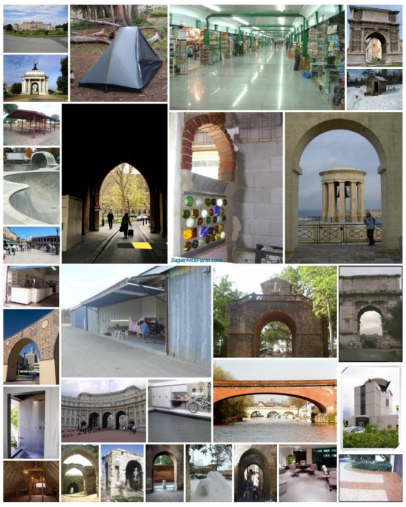


Figure 6.1: cluster Visualisation.

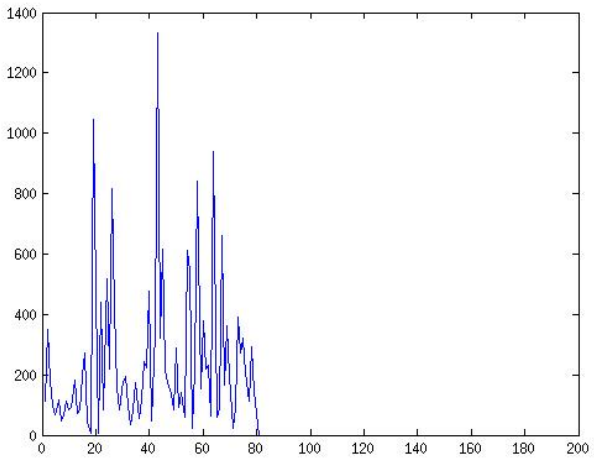


Figure 6.2: 80 clusters.

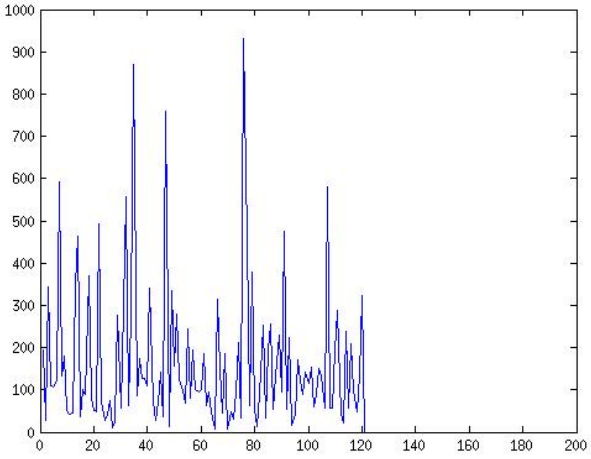


Figure 6.3: 120 clusters.

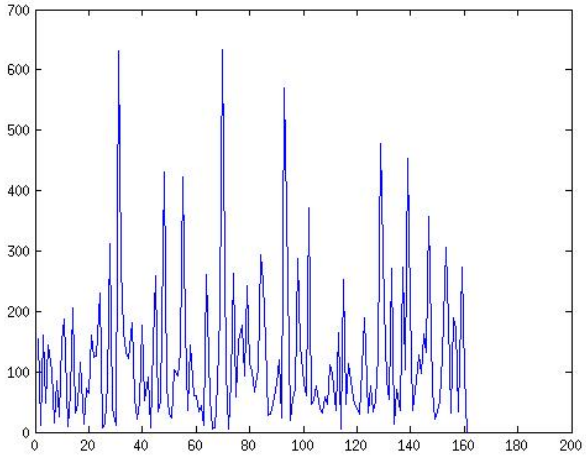


Figure 6.4: 160 clusters.

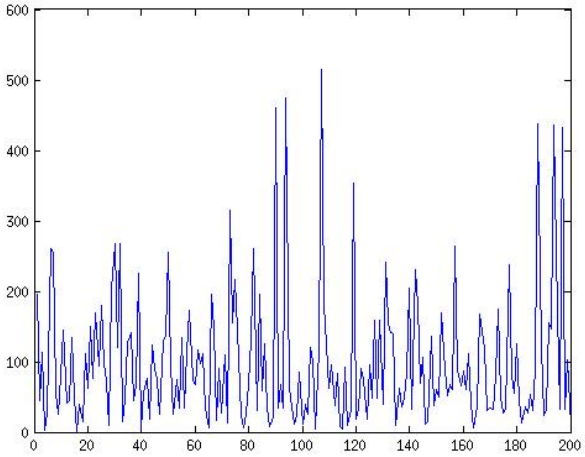


Figure 6.5: 200 clusters.

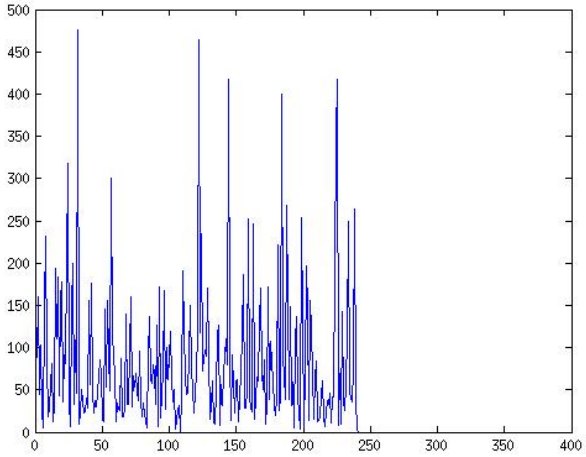


Figure 6.6: 240 clusters.

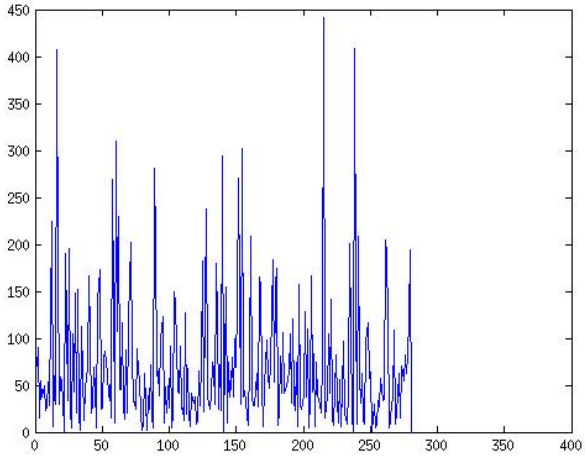


Figure 6.7: 280 clusters.

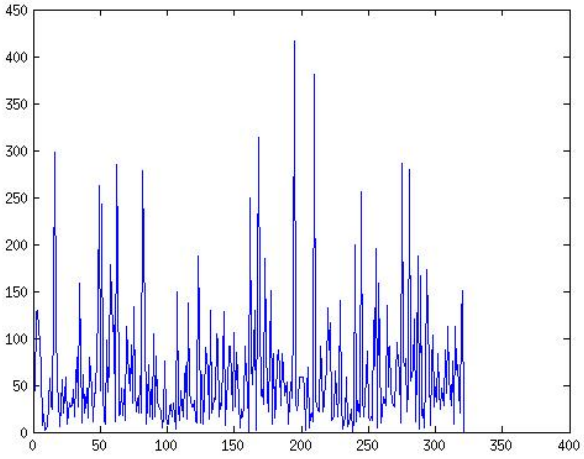


Figure 6.8: 320 clusters.

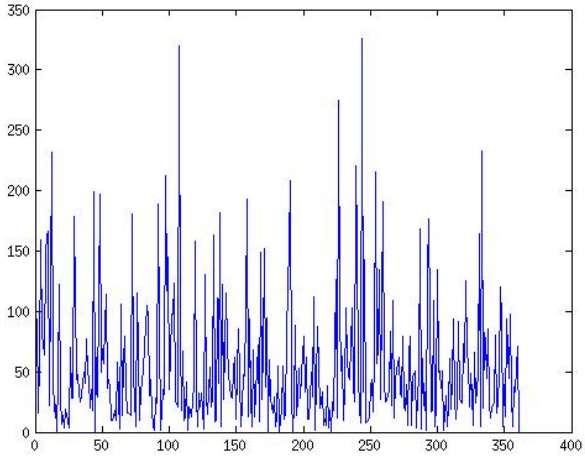


Figure 6.9: 360 clusters.

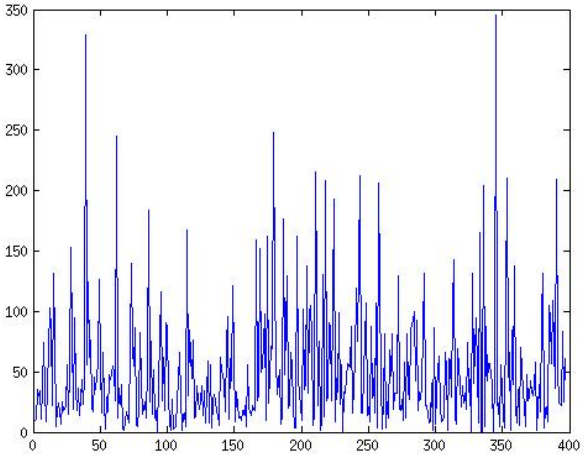


Figure 6.10: 397 clusters.

6.3 Time Required to Form Clusters : A comparison

It is not necessary that time will increase with number of clusters. Time in clustering depends upon number of iterations required for k-means clustering. Less clusters may require more iteration to find out the shortest distance from the centroid and it is also possible to get clustering done in less time if it finished in few iterations.

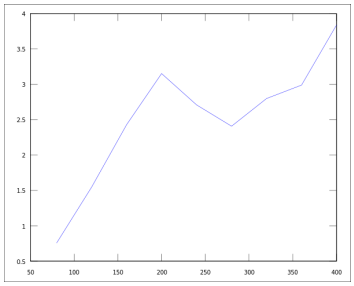


Figure 6.11: number of clusters vs time (in hrs) required to form clusters.

6.4 Searching Time in Hadoop

Figure 6.12: Query Image.

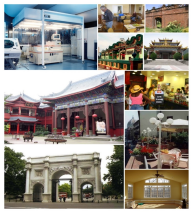


Figure 6.13: Retreived Images.

Searching in clusters depends upon the number of images in the cluster. If the query image falls in the cluster which has large number of images that other clusters then it will take more time as compare to others. As shown in figure 397 clusters take less time than 360 clusters. Time may change when query image changes.

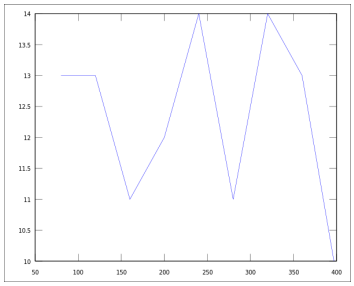


Figure 6.14: Time required to get similar images.

Chapter 7

Conclusions

We have described a different approach for distributed image processing using the Hadoop model and we also provided a practicl application using Apache hadoop framework. We also organised images of large scale database into clusters and explained some of the basic issues that should be taken into account when considering methods of parallelisation in hadoop. When discussing all of these subjects, I have focused on two-dimensional images with three color channels, which is essentially the vast majority of data that is commonly thought of as an "image".

In the case of working with a data-set of small/big images, there are almost no insurmountable issues with adapting any kind of algorithm to the MapReduce model. The divide-and-conquer approach of splitting up the data-set for independent processing works well in frameworks such as Hadoop. As we explained earlier there are some issues related to small files and the same can be solved with using sequence files having key as image name and valus as image binary content. Hadoop is well suited for non-iterative algorithms as it stores the data in hard drive after each iteration and again initialise itself for the next iteration. For example, clustering algorithms like k-means which need to compare images to each other. Another restriction stems from the Hadoop framework itself: no matter the size of input data,the start-up time of a job remains at roughly 10 to 11 seconds. With local non-iterative algorithms, it is enough to partition the input, process the pieces, and then assemble the final output image. It is the delay in initiating a MapReduce job that makes this approach unattractive for any algorithm involving many short iterations. With algorithms that have less iterations or iterations that last longer, adaptation to MapReduce might be an option.

In conclusion, I would say that when considering the feasibility of using MapReduce as a means for large-scale distributed image processing, the nature of the data determines the algorithms that can be used. With a data-set of many images, there are almost no issues to speak of, as parallelisation of the processing in this case is simply a more fault-tolerant, efficient and automated way of dividing up the data amongst several computers, doing the calculations and later merging the result back together.

References

[1] Dr. Fuhui Long, Dr. Hongjiang Zhang and Prof. David Dagan Feng. FUNDAMENTALS OF CONTENT-BASED IMAGE RETRIEVAL, .

[2] H. Burkhardt, and S. Siggelkow, "Invariant features for discriminating between equivalence classes," Nonlinear Model-based Image Video Processing and Analysis , John Wiley and Sons, 2000. .

[3] http://javahungry.blogspot.com/2014/06/how-treemap-works-ten-treemap-java-interview-questions.html, .

[4] Brandyn White, Tom Yeh, Jimmy Lin, and Larry Davis. Web-scale com- puter vision using mapreduce for multimedia data mining. In Proceed- ings of the Tenth International Workshop on Multimedia Data Mining, MDMKDD 10, pages 9:19:10, New York, NY, USA, 2010. ACM .

[5] Rafael Pereira, Marcello Azambuja, Karin Breitman, and Markus Endler. An architecture for distributed high performance video pro- cessing in the cloud. In Proceedings of the 2010 IEEE 3rd International Conference on Cloud Computing, CLOUD 10, pages 482489, Washing- ton, DC, USA, 2010. IEEE Computer Society .

[6] Clustering Billions of Images with Large Scale Nearest Neighbor Search, 2007 .

[7] Zhenhua Lv, Yingjie Hu, Haidong Zhong, Jianping Wu, Bo Li, and Hui Zhao. Parallel k-means clustering of remote sensing images based on mapreduce. In Proceedings of the 2010 international conference on Web information systems and mining, WISM10, pages 162170, Berlin, Heidelberg, 2010. Springer-Verlag. .

[8] Chen Zhang, Hans De Sterck, Ashraf Aboulnaga, Haig Djambazian, and Rob Sladek. Case study of scientific data processing on a cloud using hadoop. In Proceedings of the 23rd international conference on High Per- formance Computing Systems and Applications, HPCS09, pages 400 415, Berlin, Heidelberg, 2010. Springer-Verlag. .

[9] http://www.kishorer.in/2014/10/setting-up-hadoop-241-multi-node.html .

[10] Dhruba Borthakur. The hadoop distributed file system: Architecture and design. Hadoop Project Website, 11:21, 2007. .

[11] Tom White. Hadoop: The definitive guide. OReilly Media, Inc., 2012. .

[12] http://thinkbiganalytics.com/hadoop-sequence-files-and-a-use-case/ .

[13] http://vision.princeton.edu/projects/2010/SUN/ .

[14] J. Xiao, J. Hays, K. Ehinger, A. Oliva, and A. Torralba. SUN Database: Large-scale Scene Recognition from Abbey to Zoo. IEEE Conference on Computer Vision and Pattern Recognition (CVPR) .

[15] Dimensionality Reduction A Short Tutorial Ali Ghodsi Department of Statistics and Actuarial Science University of Waterloo Waterloo, Ontario, Canada, 2006 .