Image Search by MapReduce

COEN 241 Cloud Computing
Term Project Final Report
Team #5

Submitted by:
   Lu Yu
   Zhe Xu
   Chengcheng Huang

Submitted to:
Prof. Ming-Hwa Wang

09/01/2015
Preface

Currently, there’s no efficient way to do image search both online and offline. Therefore, we think why not try Image search with Cloud Computing. With the powerful computing ability provided by cloud, maybe we can enhance the performance of Image Search. In this project, we will mainly focus on local disk search. So there’s no well-known competitor for us to compare with. But we will still try our best to find the optimized solution.

Acknowledgement

We first want to thank Prof. Ming-hwa Wang for his kindly support and guidance. Then, we need to thank those authors of the paper in the bibliography for their helpful supplement materials to our project. Third, we should thank Santa Clara University Library for offering our discussion room and Design Center for project coding and implementation. Finally we want to thank each of our team members for their hard working on the project.
# Table of Contents

1. **Introduction**  
   - Objective  
   - What is the Problem  
   - Why this project related to the class  
   - Why other approaches are not good  
   - Why our approach is better  
   - Statement of the problem  
   - Scope of Investigation  

2. **Theoretical Bases and Literature Review**  
   - Definition of the Problem  
   - Theoretical Background of The Problem  
   - Related Research To Solve The Problem  
   - Advantage/Disadvantage Of Those Research  
   - Our Solution To Solve This Problem  
   - Where Our Solution Different From Others’ Solutions  
   - Why Our Solution Is Better  

3. **Hypothesis**  
   - Multiple Hypothesis  

4. **Methodology**  
   - How to Generate/Collect Input Data  
   - How to Solve the Problem  
   - How to Generate Output  
   - How to Test Against Hypothesis  
   - How to Proof Correctness(Required by Dissertation)  

5. **Implementation**  
   - Code  
   - Design Document and FlowChart  

6. **Data Analysis and Discussion**  
   - Output Generation and Analysis  

7. **Conclusions and Recommendations**  
   - Summary and Conclusions  
   - Recommendation for Future Studies  

8. **Bibliography**  

9. **Appendices**  
   - Sample code
List of Tables and Figures:
Please refer to the Section 9 Appendices.

Abstract
In this project proposal, we propose to introduce an accurate and efficient image search by MapReduce application based on Hadoop Image Processing Interface(HIPI) to search the most similar photos in your local library.
1. Introduction

Objective

We design this specialized data search focusing on image search by MapReduce that allows us to search the most similar photo(s) compare with the target photo you initially input in the system, in order to fulfill your search request. We Aims for providing an accurate and efficient image search by MapReduce.

What is the Problem

Some image searching function can be found online, however often we are looking for searching one or more similar photos in your own library, in other words, on a local disk. With currently large image dataset level, it is impossible to do a search by brutal force, It is also difficult to find an existing application to solve this problem because the lack of limited application and platform available for searching similar images in local. Therefore the problem of how to find the most similar photo comes to our attention and we decided to build an application to complete the image search function by searching locally. To our knowledge so far, we are one of the first few people to provide this application.

Why this project related to the class

The recent evolutions makes the use of MapReduce as well as distributed and parallel frameworks dedicated to grid and cloud computations such as Hadoop. Our project is based on Hadoop Image Processing Interface (HIPI), which stores and processes images efficiently based on Hadoop MapReduce platform. We also implement MapReduce, which processes and generates large data sets with a parallel, distributed algorithm on larger clusters.

Why other approaches are not good

As far as our research goes, we found that all other’s approaches are limited to small data scale with very slow search speed. We cannot apply these approaches to perform our project in image search, so our solution will include HDFS/HIPI and MapReduce. We will discuss this more in detail in Section 2.

Why our approach is better

As mentioned in the previous paragraph that others’ solution cannot be applicable to large set of datas. Usually they are target to process approximately 10 photos. However, our solution can process up to 10,000 photos with a faster speed. Moreover, for the future works, our solution can be extendable to online fetching images and comparing images.
**Statement of the problem**

The availability for local image search function/application is very limited, we will build an application to perform local request on finding the most similar photo in a large dataset.

**Scope of Investigation**

Our project will focus on implementing Image Search algorithm in large dataset level. And we will also focus on how to turn the existing search algorithm into MapReduce Code. In the following sections, we will describe theoretical bases and literature review, hypothesis and methodology in details.
2. Theoretical Bases and Literature Review

Definition of the Problem

Image search is a specialized data search used to find images. To search for images, a user may provide query terms such as keyword, image file/URL, or click on some image, and the system will return images "similar" to the query image. The similarity used for search criteria can be meta tags, color distribution in images, region/shape attributes, etc. Nowadays, most image search solution do not support local disk search. So we are trying to find a method to do image search locally by the powerful cloud.

Theoretical Background of The Problem

The major interface we used to solve this problem is Hadoop Image Processing Interface (HIPI). It provides a format for storing images for efficient access within the MapReduce pipeline, and simple methods for creating such files. By providing a culling stage before the mapping stage, it gives the user a simple way to filter image sets and control the types of images being used in their MapReduce tasks. Finally, it provides image encoders and decoders that run behind the scenes and work to present the user with float image types which are most useful for image processing and vision applications.

Related Research To Solve The Problem

Here we found two related algorithm to do image search. One is SIFT and the other is PHash. SIFT stands for Scale-invariant feature transform and PHash is Perceptual Hash. Since both of them are belong to Image Processing, we will not spend too much words on them. Please refer to the URLs in Reference part if you are interested in the details of them.

Advantage/Disadvantage Of Those Research

Advantage: The result from SFIT is very accurate. PHash is less accurate but it is still has a decent performance and it is easy to understand and written in Java.

Disadvantage: The speed of SIFT is very slow. Neither SIFT nor PHash are never implemented with cloud computing before. So they are not scalable. Moreover, SIFT is not easy to understand and implement in Java. Its implementation process is complicated.

Our Solution To Solve This Problem

Our Hadoop MapReduce program design steps as follows:
Use HIPI interface to package the large photo datasets to two HIB bundle files, which are several GBs source photo files and a single user input photo data, also called the target photo. The HIB bundle file format will consist of two files: a data file containing concatenated images and an index file.

We write one Mapper class for calculating distance between one source file from HIB and the target photo using Naive Comparison Method. Then, in order to reference each photo in HIB, the Mapper will identify each photo with their hex hash value. Finally, Mapper will generate distance/hex hash value pairs to the reducer.

Our Reducer function is very simple. It will only use the internal sort to sort all the distance in the descending order.

After get all those distances, we output the top ten smallest distance images for result. The smaller score means the more similar image, so the smallest distance will be the better result.

Where Our Solution Different From Others’ Solutions

The main difference is we use new technologies such as HDFS/HIPI and MapReduce.
<table>
<thead>
<tr>
<th></th>
<th>Others (w/o Cloud)</th>
<th>Our Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Support Multiple Nodes</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Search Speed</td>
<td>Slow</td>
<td>Fast</td>
</tr>
<tr>
<td>Data Scale</td>
<td>Small</td>
<td>Large</td>
</tr>
</tbody>
</table>

*Table 1. Where we are different*

**Why Our Solution Is Better**

Others’ solution cannot be applicable to large set of datas. Usually they are target to process ~10 photos. However, our solution can process ~10,000 photos with a faster speed. Moreover, for the future works, our solution can be extendable to online fetching images and comparing images.
3. Hypothesis

**Multiple Hypothesis**

The goal of this project is to introduce an accurate and efficient image search by MapReduce application based on Hadoop Image Processing Interface (HIPI) to search the most similar photos in your local library. Also, we assume the Cloudera Hadoop in VMware can handle the size of the sample data and run the MapReduce Program smoothly.

4. Methodology

**How to Generate/Collect Input Data**

Due to the limitation of the platform we have, we could not test very large dataset, since we only have single-node MapReduce. Hence, the goal of our project is to find the most similar pictures in our 2.2 GB datasets matching the picture offered by the user. The whole dataset contains 10,000 image files, which is not suitable for MapReduce. MapReduce prefer single large files, so we need to use HIPI to transform this dataset into one or two single large file(s), before we can use it on Hadoop.

**How to Solve the Problem**

**Tools and Algorithm:**

Typically image size is small comparing to HDFS block size (64M by default). MapReduce tasks run more efficiently when the input is one large file as opposed to many small files. We use a third-party open source Hadoop tools to reach the requirement to improve the run time performance. HIPI, stands for Hadoop Image Processing Interface, provides a solution for storing a large collection of images on the Hadoop HDFS and make them available for efficient distributed processing, that means by using HIPI we can manipulate the large photo dataset like just one file, so when we were doing the MapReduce programming, it is just simple as like writing the words count MapReduce program.

We use Naive Comparison Algorithm to calculate the similarity between two images. The key idea of this algorithm is to get 9 specified positions of data from target images and calculate the average distance according to these 9 positions.
For more details of HIPI and algorithm, please see the Related Work.

**Language Used**  
Since Hadoop is written in Java and we only have to write a MapReduce program, the only programming language we use is Java.

**Tools Used**  
HIPI, MapReduce.

**How to Generate Output**  
The expected outcome of our project should be like: we build a GUI, where user can upload an image. Then the user will get back a sequence of images in the order of descending similarity.

**How to Test Against Hypothesis**  
In order to test our project, we will choose some images as target files. With these images, we will create several variants (such as crop, scale, change color, adding some noise, change the angle or print and re-scan). We will use these files to search in our dataset, and try to find out the correct rate and the running time of the search. We may also do some other research on the CPU load, memory usage or other interesting things.

**How to Proof Correctness (Required by Dissertation)**  
In order to test our project, we will choose some images as target files. With these images, we will create several variants (such as crop, scale, change color, adding some noise, change the angle or print and re-scan). We will use these files to search in our dataset, and try to find out the correct rate and the running time of the search. We may also do some other research on the CPU load, memory usage or other interesting things.
5. Implementation

Code (Details in Appendix)

MapReduce Program Description:

Functions

Rescale()
CalcSignature()
CalcDistance()

Input

BufferedImage Reference;
BufferedImage Target;

Output

Distance Between Two Images (Double)

Design Document and FlowChart

Flowchart

The flowchart of our program is nothing but the flowchart of a typical MapReduce Program. First step is map, then reduce.

Infrastructure:

The infrastructure we use is Cloudera Hadoop MapReduce. We totally write one Mapper class and one Reducer class.

Mapper class input is the large HIB image bundle data. Each Mapper class use pHash(BufferedImage image) function to do the image processing and generate the perceptual hash value as the value in <key, value> for every image.

In the Reducer class, we use the intermediate hash results to calculate the Hamming distance between the source photos and target photos to generate the score of similarity. The score is cumulative from each comparison with target photo. Finally, generate a list of photos with top ten minimum score. Main function is to get location of those result photos in the huge size source photo bundle file (TEST.hib).
Setup:

Since we use the HIPI, we need to first setup HIPI in our Hadoop system. For brevity, we won’t show the detailed steps how to install HIPI in Hadoop. Here’s all the steps to install:(http://hipi.cs.virginia.edu/gettingstarted.html).

All the source and target images are stored on Local Disk. (This is unnecessary in the future, please see the Future Works).

6. Data Analysis and Discussion

Output Generation and Analysis

As Figure 3 shows, our result will be the top ten most similar photos and the corresponding scores. The lower the score is, the more similar the photo is. We wrote a linux script to download these 10 photos from HDFS and with the name of the photo(such as 6e32b6549…) we can check if the photo we find the correct one or not.

<table>
<thead>
<tr>
<th>Photo Name</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>6e32b6549013ae6ca06576213d37a858c87d6c</td>
<td>763</td>
</tr>
<tr>
<td>fabbacb539fcd2dbb62279b1c9874f19fbbe</td>
<td>778</td>
</tr>
<tr>
<td>379b67f13de64c8e55c2fb948d6e55bca1d9bb4b</td>
<td>779</td>
</tr>
<tr>
<td>ab99a12130a4baafce8c71d3bb652c8e94ae</td>
<td>781</td>
</tr>
<tr>
<td>486ec6a9489130f64b9eb2b563fa48e57f8</td>
<td>782</td>
</tr>
<tr>
<td>b0692fa6f44db15cedadfd49d3d89a1544b0d8</td>
<td>783</td>
</tr>
<tr>
<td>aa649536afcl3ea7b62bf6ae6010eeea98e2c</td>
<td>783</td>
</tr>
<tr>
<td>5a6ddef6a211787dec3b34a1c96df7c7c84b82</td>
<td>786</td>
</tr>
<tr>
<td>78d03db31d393ad10ddffcaae3d97c27f08b2c5</td>
<td>786</td>
</tr>
<tr>
<td>8daab5ff12df21dcd7d1449b953a36685a618</td>
<td>786</td>
</tr>
</tbody>
</table>

*Figure 2. Top Ten Most Similar Photos and their Scores*

Not like other projects that demoed in the class, our result is very obvious and straightforward to evaluate. If we didn’t find the one we want, it has low accuracy. If we do, it have a somehow high accuracy. So in the third position of the ten, we find the one and we find only one photo of the goal. But we mixed 7 photos into the dataset. So we can say our accuracy is 1/7 = 14.3\%.

Compare to the Google Image Search’s ~60\%. We are still far away from Google. However, we didn’t apply any machine learning algorithm in our search. If we have enough photos of goal(~1,000 photos), we could do data training for the machine learning and enhance our accuracy.

Expected Output

The expected output of our project will be very straightforward. Since our goal is to find the photo with the highest similarity within a large size of photo dataset, the expected output is we do find the photo we originally want.
7. Conclusions and Recommendations

Summary and Conclusions
In this project, we are trying to use Hadoop MapReduce to find the photo with the highest similarity in a large photo dataset (2.2 GB). The tools we use are from HIPI and the Algorithm to calculate the similarity of two photos we use is PHash. After we go deep in the project, we found PHash is not accurate enough to calculate the similarity. We finally switch to a better algorithm to calculate.

Recommendation for Future Studies
We noticed that the HIPI also includes a MapReduce Program called DistributedDownloader (http://hipi.cs.virginia.edu/examples/downloader.html). It can allow us to download our source images from given lists of URLs by scraping a website that contains photo collections like Flickr, Google Images, Bing, or Instagram... and create the HipiImageBundle File (HIB) from these images. So we don’t even need to store the photos dataset on our Local FS. Due to the time limitation, we cannot create hundreds of facebook/Flickr accounts and upload the images into albums to test our program. Hence in the future, we can upgrade our design to automatically retrieve our source images from the Social Networks, instead of only processing the images already stored in the disk. This makes our design more practical and natural.
Therefore, the next version of our program is like: One Flickr user posts an image, say Big Ben, naturally hundreds of people on the Facebook who visited or live England post their snapshots of a lot of architectures in England, including the same Big Ben. Our program distributedly download these images and compare them to the target Big Ben and finally find the same Big Ben we desire.

8. Bibliography


Journal of Emerging Trends in Computing and Information Sciences VOL. 3, NO. 4, April 2012

9. Appendices

Sample code
Too long...Will not show here. Please see the result from Submit.