Cloud Computing Project Report

Efficient Image Steganography using Hadoop

Adharsh Desikan
Vignesh Vijayabasker

1. MOTIVATION:

We worked on Image Steganography during our under graduation. We did our final course project on Image Steganography. Also we published an international paper in the same. Our interest on image steganography is the main driving force for selecting this project. We have implemented our paper (http://www.ijcaonline.org/volume5/number7/pxc3871305.pdf) using MapReduce paradigm.

2. OVERVIEW:

Image Steganography: It is a technique in which large data can be hidden inside a given image. The image in which the data is hidden is called a *cover*. The biggest challenge in Image Steganography is selecting a cover that gives least Mean Square Error for a given data.

The project is aimed to find the best cover possible for a given data, encrypt the data inside the image.

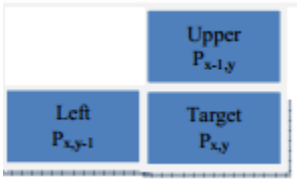
There are various algorithms for performing this encryption. We have chosen PIXEL VALUE DIFFERENCING (PVD) algorithm to accomplish the above discussed task. PVD is one in which the number of bits that are encrypted in a particular pixel depends on the neighboring pixels. So more bits will be encrypted when the neighboring pixels value differs in huge amount (rough area in the image) and less number of pixels will be encrypted when the neighboring pixels have almost same values (smooth area in the image). Inputs and Outputs:

3. ALGORITHM:

The Algorithm is discussed in detail in the below mentioned paper.

http://www.ijcaonline.org/volume5/number7/pxc3871305.pdf

The algorithm discussed in the paper is modified to benefit the parallel computing using the map and reduce paradigm.



Consider the target pixel $P_{x,y}$. The no. of bits to be embedded in the pixel is determined by the difference of the Upper and Left pixel of the target pixel.

Encryption:

Step 1: Difference between the Upper and the Left pixel is calculated.

$$d=abs\ (P_{x,y-1} - P_{x-1,y})$$

Step 2: Calculation of n: the number of the insertion bits in a target pixel Px,y is calculated, using the following formula:

$$n = \begin{cases} \lfloor \log_2 d \rfloor & if \quad d > 3 \quad d = odd \\ \lfloor \log_{2d} \rfloor - 1 & if \quad d > 3 \quad d = even \\ 1 & if \quad d < 3 \end{cases}$$

Step 3: Calculate a temporary value $t_{x,y}$ using:

$$t_{x,y} = b - g_{x,y} \bmod 2^n$$

b- Decimal representation of the binary message bits

$g_{x,y}$- Pixel Value

Step 4: To make the quality of the image higher, select the nearest value to the target pixel's value of the cover image by optimal pixel adjustment process

$$tl = \begin{cases} t & if \quad -\lfloor (2^n - 1)/2 \rfloor \le t \le \lfloor (2^n - 1)/2 \rfloor \\ t + 2^n & if \quad -(2^n + 1) \le t < -\lfloor (2^n - 1)/2 \rfloor \\ t - 2^n & if \quad (2^n - 1)/2 \le t < 2^n \end{cases}$$

Step 5: The final pixel value is obtained by

$$g^* = g + tl;$$

Mean Square Error:

The Mean Square Error is calculated as

$$MSE = \frac{1}{MN} \sum_{i=1}^{M} \sum_{j=1}^{N} (X_{i,j} - Y_{i,j})^2$$

Where $X_{i,j}$= pixel value at the i,j position in the cover

$Y_{i,j}$ = pixel value at the i,j position in the output image containing hidden data

Using the above formula for MSE, we can compare the various images as cover for the same message.

4. INPUTS AND OUTPUTS

Preprocessor Task:

Input: A local folder containing images as argument to the Image.java.

Output: Returns a local folder that has text files which contains the pixel values of each pixel in the image along with the row number to which the row belongs. Part of an example output is shown below

```
1   255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
10  255 255 255 256 255 256 255 256 255 255 255 255 255 255 255 255 255
100 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
101 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
102 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 254
103 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 254
104 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 254
105 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
106 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
107 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
108 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 254
109 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 254
11  255 255 256 255 255 255 256 255 256 255 256 255 255 255 256 255 256
110 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 254
111 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 254
112 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 254
113 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
114 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
```

Image Steganography:

Input: Output of the preprocessor step along with the text file containing the message to be embedded

Output: Text files containing the pixel values of the Images with the message hidden and a text file containing the Mean Square values for all the images.

5. MAP-REDUCE STEPS:

Pre Processing step:

The given set of images is converted into text files containing the pixel values in a matrix form.

Task 1: Calculating the individual no. of bits that can be embedded in each row

Map Task:

- Key – When processing i^{th} line, keys will be i and i+1 where i is the row no.
- Value – Corresponding entire row values of i^{th} row.

Reducer Task:

- Key – row number
- Value – Total no. of bits that can be embedded on the line and the row value

Task 2: Calculating the cumulative no. of bits that can be embedded in each row. This is done to find the start index in the message stream

Map Task:

- Key – When processing i^{th} line, keys will be i+1, i+2,… n where i is the row no.
- Value – Total no. of bits that can be embedded on the line and the row value

Reducer Task

- Key – row number
- Value – Cumulative total no. of bits that can be embedded on the line and the row value

Task 3: Embedding message and calculating the MSE

Map Task:

- Key – When processing i^{th} line, keys will be i and i+1 where i is the row no.
- Value – Corresponding entire row values of i^{th} row and the cumulative no. of bits to be embedded.

Reducer Task:

- Key – row number
- Value – Embedded row and the MSE of that row.
- Key 2 – Image File Name
- Value 2 – MSE of the image File.

6. DATA SOURCES:

A folder containing number of images which is only of the type "*.png ". Because the other image formats will compress and the image steganography cannot be performed on them. The images can be of any size.

Normally the size depends on the size of the images used.

7. TIME TAKEN FOR EACH STEP:

We tested it with ten images.

Preprocessor: within a minute

Task 1: 3 min

Task 2: 7 min

Task 3: 5 min

8. REFERENCES:

ZIG-ZAG PVD – A Nontraditional Approach - http://www.ijcaonline.org/volume5/number7/pxc3871305.pdf

9. GOALS ACCOMPLISHED:

Mandatory Accomplishment:

Implementing the above algorithm for a set of images that are already present in a given input folder

Status : Accomplished.

Most Likely completion:

Implementing the above algorithm by getting the images from a Bing search and performing decryption for the best cover

Status : Accomplished.

Ideal completion:

Performing the above tasks for multiple images and multiple messages (The above algorithm is for multiple images and single message)

Status : accomplished.

10. INDIVIDUAL TASK:

Adharsh – Task 2, Preprocessing and Decrypting;

Vignesh – Task 1, 3, Getting images from bing;

Preprocessing and post processing both together; this is work division for mandatory accomplishment. For most likely Vignesh – Bing search crawling and Adharsh – decryption

11. SURPRISES ENCOUNTERED:

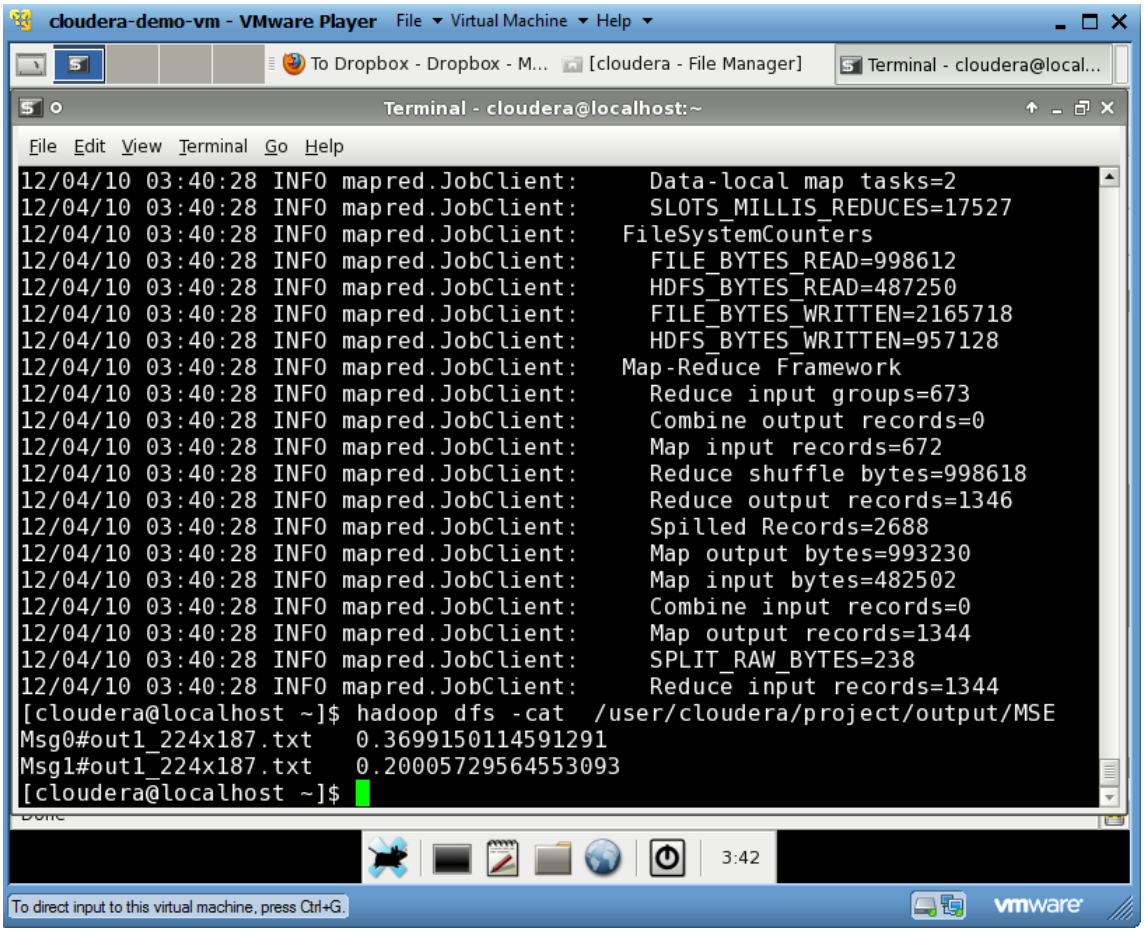
- Handling images was not as easy as our initial assumptions
- The Preprocessor and the post processing development took more time than initially estimated.
- Maintaining the row order of the images and the message bits.

12. Instructions to run the program.

Please find the instruction in the attached README text file.

13. Screen Shots:

Screen shot showing the output of the ideal condition. We have uses to different message and tried to embed on a single image (this can be performed in multiple images). The two different MSEs are show in the picture.



The Input and the output images (out1_224x187.txt) are shown below. The Image in the right side has the message embedded in it.

The MSE values for the 4 images that we tried.

Image Name	**MSE**
out0_166x140.txt	5.504776247848537
out1_224x187.txt	0.7656608097784567
out2_255x184.txt	2.3639386189258316

The reason for the blue and red spots near the white and black positions is that these are boundary values i.e. 255 or 0. For the boundary values, these problems will occur. In case of the

out1_224x187.txt

The other Images and their outputs

out2_255x184.txt

out0_166x140.txt

The above images have a boundary background (black and white). In case of color images that are not in the boundary backgrounds, the distortion visibly seen is far below.

13. Possible bugs:

While trying to embed the message in the image, we are using hash map to store the corresponding MSE values for different images. If two different reducers were trying to change the value in a same HashMap there may be a possibility to a MSE value for one row. But during testing we never dealt with such condition.

14. Appendix:

- At first we tried to construct the text file containing the pixel values from the image through map reduce. But unfortunately this went vain because BufferedImageReader is supported by map reduce.
- We are using JSOUP jar to get the DOM structure of the BING.
- We are allowed to get up to 15 images from BING.