Image Filtering with MapReduce in Pseudo-Distribution Mode

Tharindu D. Gamage*, Jayathu G. Samarawickrama†, Ranga Rodrigo‡ and Ajith A. Pasqual§
Department of Electronic & Telecommunication Engineering,
University of Moratuwa, Sri Lanka
Email: *tharindug@uom.lk, †jayathu@ent.mrt.ac.lk, ‡ranga@uom.lk, §pasqual@ent.mrt.ac.lk

***Abstract*—The massive volume of video and image data, compels them to be stored in a distributed file system. To process the data stored in the distributed file system, Google proposed a programming model named MapReduce. Existing methods of processing images held in such a distributed file system, requires whole image or a substantial portion of the image to be streamed every time a filter is applied. In this work an image filtering technique using MapReduce programming model is proposed, which only requires the image to be streamed only once. The proposed technique extends for a cascade of image filters with the constrain of a fixed kernel size. To verify the proposed technique for a single filter a median filter is applied on an image with salt and pepper noise. In addition a corner detection algorithm is implemented with the use of a filter cascade. Comparison of the results of noise filtering and corner detection with the corresponding CPU version show the accuracy of the method.**

Keywords*—*Image Filter; MapReduce; Hadoop

I. Introduction

The amount of image and video data increases every day. To store this massive amount of data, the use of a distributed file system is a must. A distributed file system facilitates the storage of large amount of data in many computer nodes. It splits the file in to blocks and replicates at multiple DataNodes [1]. It is a common practice to use the MapReduce programming model to process the data stored in a distributed file system.

Apache Hadoop, an implementation of the MapReduce model, is a framework that enables distributed processing of large data sets across computers. It provides distributed file system named HDFS. Along with HDFS, Hadoop MapReduce is provided. Hadoop MapReduce is an implementation of the paper [2]. With the availability of image and video data on distributed file systems it is important to consider image processing techniques using MapReduce programming model.

Image filtering is used by most of the image processing applications as a pre-processing step. Different filter kernels apply different operations on images, such as noise removal and feature detection. All these filters apply certain operation on an image for the neighborhood in question. This work is focused on image filtering with MapReduce programming model [2].

MapReduce programming model is used for processing large data sets. With the "text" mode of the MapReduce programming model, each line of the text file is considered as a key/value pair [2]. In the "text" mode, the data in the text file is processed line by line in parallel. It is different when it comes to image processing with MapReduce programming model. For example, to apply a filter on an image, the intensity values of the neighbouring pixels must be available.

One method of parallelizing an image processing task with MapReduce is to divide the image into multiple parts. The second method is processing multiple images at the same time. In the work of Yamamto and Kaneko [3] they have used both methods to process a video: first, processing video frames by dividing each frame into four parts, second, processing multiple frames without dividing the frame. The frame number is included in the key when multiple frames are processed [3]. When processing images by dividing in to multiple parts, the processed data is stored as the output. To process multiple video frames at the same time they have stored an image as the output [3]. Yan and Huang [4] integrated OpenCV image processing library [5] to MapReduce programming model. Hadoop Image Processing Interface also known as HIPI is a library performs image processing tasks in a distributed computing environment. It uses a combined set of images as the input type. The combination of images avoids handling of large number of small files. When combining multiple image files the layout is stored in metadata [6]. The work of Almeer [7] have shown how MapReduce programming model is used to process remote sensing images. Image processing algorithms applied on those images are Sobel filtering, image resizing, image format conversion, auto contrasting, image sharpening, picture embedding, text embedding, and image quality inspection. The image is loaded without splitting it. Almeer [7] modified the RecordReader class, responsible for loading the data from its source, to load the image to one Map instance. All the processing algorithms are implemented in the Mapper: the Reducer is not used for processing. In the above mentioned attempts of processing images with MapReduce programming model, the property of grouping key/value pairs of a filter neighbourhood is not used.

In this paper a method of image filtering with MapReduce programming model is proposed. The image is loaded to the Map instance by the same method used by [7]. The proposed technique in this paper uses a Reduce instance for each pixel. To apply a filter to a particular pixel the neighbourhood pixels are also sent along with each map and reduce process with a same key. The key/value pairs of the same key are grouped and passed to the reduce function. Use of this property in image filtering is a contribution of this paper. Another contribution is when image filters are applied in cascade the neighbours of a pixel are rearranged. As a result, proposed technique is

978-1-4799-1740-2/15/$31.00 ©2015 IEEE

effective if there is a cascade of image filters applied with in constraint for kernel size being fixed.

II. Methodology

There are two types of image filtering techniques proposed using MapReduce programming model: single filter, cascading filters. In both of these techniques image is loaded from the distributed file system as a byte stream in the map function. For the cascading filters image is only loaded in the first map function.

A. Single Filter

In the process of applying a single filter the image is stored in the Hadoop distributed file system which is configured on a single node. The map function [2] of the MapReduce programming model loads the image stored in the distributed file system as a byte stream. When the image is loaded its pixels are accessed through two nested iterations. Within the two nested iterations there must be another two nested iterations to access the neighbourhood of a particular pixel.

The corresponding pixel and its neighbouring pixels are emitted as a key/value pair with the same key. The key consists of the row index and the column index of the corresponding pixel. The value of the key/value pair consists of the intensity value of a pixel along with its index. The assignment of the pixel index for a 3×3 filter window is shown in Fig. 1. The image index and the intensity value of the pixel sent along with the value of the key/value pair is used to apply the filter kernel. The filter kernel is applied in the reduce function of the MapReduce programming model.

In the map function, the pixel in question and its neighbourhood are emitted with a same key. According to the MapReduce programming model key/value pairs of same key are grouped together and passed to the reduce function [2]. As an example, for a 3×3 neighbourhood, 9 key/value pairs are grouped together and passed to the Reduce function. The number of neighbourhood key/value pairs are less than 9 in the borders of the image for a 3×3 filter. Two examples of such occasions are shown in Fig. 3 and Fig. 4. A 3×3 neighbourhood of left border pixel is shown in Fig. 3 and neighbourhood of the bottom left corner pixel is shown in Fig. 4. Grouped key/value pairs of the reduce function contains the intensity values of the shaded pixels of Fig. 3 and Fig. 4. From the available key/value pairs it is possible to get the missing key/value pairs. The process of duplicating the key/value pairs is dependent on the filter. The output of the reduce function is the resultant value obtained by applying the filter kernel on the neighbourhood.

B. Cascading Filters

When there are cascading filters, a cascade of map reduce functions is used. The first map function of the cascade is the same as the map function of a single filter as described above. The duplication of key/value pairs at the borders of the image are also same as single filter. The considered pixel neighbourhood is grouped together and passed to the reduce function. For this neighbourhood the filter kernel is applied, and resultant value obtained. In this filter cascade, the resultant value is not sent to the output as in the single filter. The reduce function of the single filter needs to be modified as follows:

The main idea behind the cascade of filters in this work is to reuse the calculated neighbours of the reduce function in the subsequent filter. In each of the reduce functions in the filter cascade, the pixel indices of the neighbourhood and the row index and column index of the considered pixel are available. With the use of the pixel indices and the row and column

8	7	6
5	4	3
2	1	0

Fig. 1. Pixel indexes for a 3×3 filter window

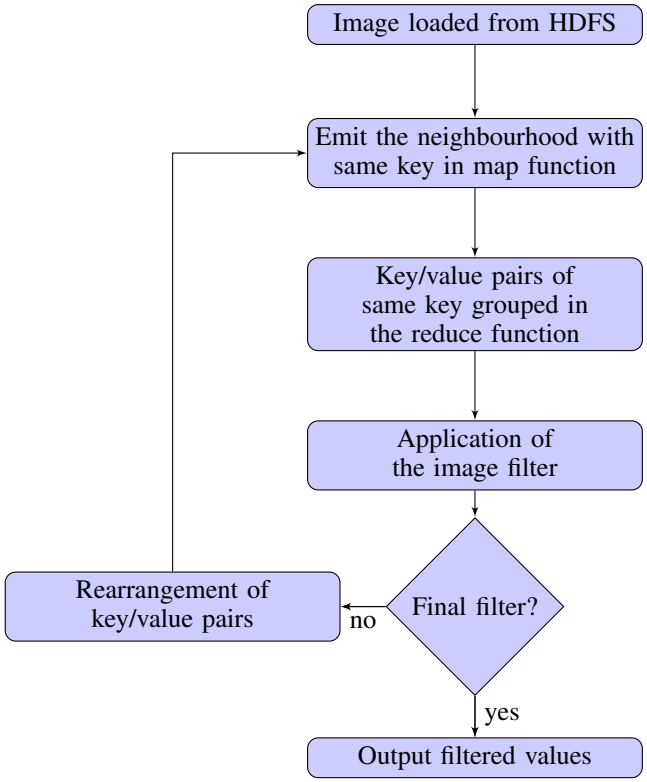

Fig. 2. Filtering process with MapReduce

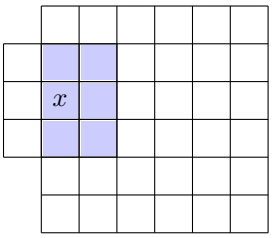

Fig. 3. 3×3 neighbourhood at left border of the image

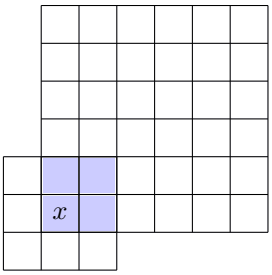

Fig. 4. 3×3 neighbourhood at bottom left corner of the image

TABLE I. PIXEL LOCATIONS BASED ON IMAGE INDICES FOR A 3×3 NEIGHBOURHOOD

Pixel_index	Pixel_ location (row index, column index)
0	(11,16)
1	(11,15)
2	(11,14)
3	(10,16)
4	(10,15)
5	(10,14)
6	(9,16)
7	(9,15)
8	(9,14)

indices of the considered pixel, it is possible to calculate the row and column indices of the neighbouring pixel. Pixel indices of a 3×3 neighbourhood is shown in Fig. 1. Row and column indices of the considered pixel corresponds to pixel index 4. The calculation of the pixel location for each pixel index is obtained from the Equations 1 and 2. The variable, i varies from -1 to $+1$. For each i, j varies from -1 to $+1$. In equation 1 the filter size is $k \times k$. In equation 2 the x and y are the row and column indices of considered pixel.

$$\text{Pixel\_Index}(i,j) = (i + \left\lfloor \frac{k}{2} \right\rfloor) \times k + (j + \left\lfloor \frac{k}{2} \right\rfloor) \quad (1)$$

$$\text{Pixel\_location}(i,j) = ((x-i),(y-j)) \quad (2)$$

For an example pixel location (10,15) the calculated pixel locations of a 3×3 filter is shown in the Table I.

In a filter cascade of 3×3 neighbourhood, pixel location corresponding to pixel index 4 becomes a neighbour to the pixel locations corresponding to pixel indices other than 4 in the next filter. With the use of that property the resultant value calculated by applying the first filter becomes a neighbor in the next filter. That process is shown in Fig. 5 where x is the considered pixel location having pixel locations a, b, c, d, e, f, g and h as its neighbours. In the next filter pixel location x becomes a neighbour for all the pixel locations from a to h. In the reduce functions of the filter cascade, the key/value pairs are rearranged to use the above mentioned property. The key is changed based on the pixel locations corresponding to pixel indices. The value is changed based on the rearranged pixel index and resultant value calculated by the first filter. The rearrangement of pixel indices of a 3×3 neighbourhood in the reduce functions is shown in the Table II.

The rearranged key/value pairs in the reduce function are emitted by the map function of the next filter without any change to the key/value pairs. The rearrangement of the

TABLE II. PIXEL INDEX REARRANGEMENT IN REDUCE FUNCTIONS FOR A 3×3 NEIGHBOURHOOD

Pixel index	Rearranged Pixel index
0	8
1	7
2	6
3	5
4	4
5	3
6	2
7	1
8	0

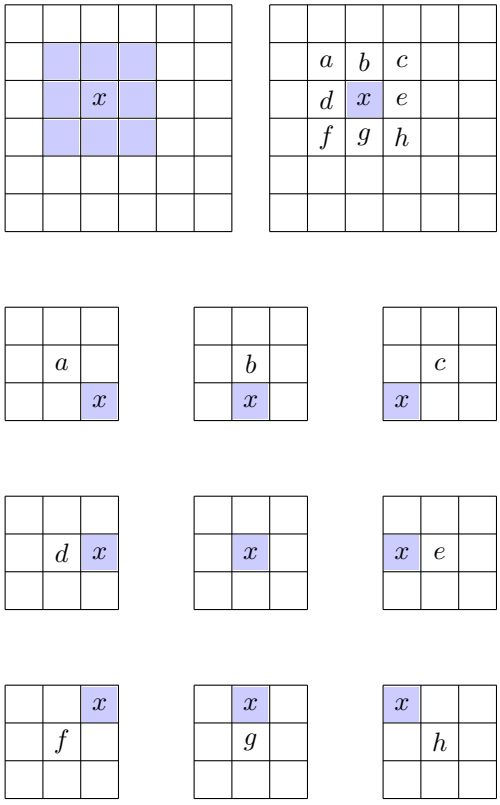

Fig. 5. Pixel index rearrangement of 3×3 neighbourhood

key/value pairs in the reduce function of the previous filter gives key/value pairs with same key. The key/value pairs which have the same key are grouped and passed to the reduce function. The corresponding filter kernel is applied on the grouped key/value pairs in the reduce function. If there is another filter available in the filter cascade, the key/value pairs need to be rearranged in the reduce function as described above. In the final reduce function of the filter cascade the output of the reduce function is the resultant value obtained by applying the filter kernel. The overall process is illustrated in Fig. 2.

III. RESULTS

The experiments are done in the following environment. Operating System: Ubuntu 14.04 LTS, CPU: Intel(R) Xeon(R) 2.80GHz and Memory: 6 GB. The installed Hadoop version is Hadoop 2.4.1. Hadoop is configured as pseudo-distributed mode.

Median filter is selected for the experiment of the single

Fig. 6. Image with salt and pepper noise

Fig. 7. Filtered Image by Median Filter

filter. Median filter is a nonlinear filtering technique used to remove noise. It is known to be effective in removing salt and pepper noise [8]. This Median filter is implemented on Apache Hadoop framework [9]. The filter is applied on a 3×3 neighbourhood. In the map function the key/value pairs are emitted giving the same key to the neighbourhood. The key/value pairs are grouped by the key and passed to the reduce function. The median filter is applied on the reduce function. The output of the reduce function is the resultant value obtained by applying the median filter. To verify the calculation of the median filter sum of squared differences (SSD) are calculated according to equation 3. In equation 3, h and w are height and the width of the image. The function mFilMR gives the filtered intensity values calculated by the median filter using MapReduce programming model. The function mFilMat gives the intensity values calculated by applying the median filter using the image processing toolbox [10].

$$SSD = \sum_{i=0}^{h-1} \sum_{j=0}^{w-1} \frac{(\mathrm{mFilMR}(i,j) - \mathrm{mFilMat}(i,j))^2}{h \times w} \qquad (3)$$

The calculated SSD value for the median filter in the above experiment is 0. Fig. 6 shows the image with salt and pepper noise and the Fig. 7 shows the filtered image by the median filter.

For the experiment of the image filter cascade sobel filter is applied as the first filter. Sobel filter is used to calculate image derivatives. It calculates horizontal changes Gx and vertical changes Gy of the image by using two filter kernels [8]. The kernels of size 3×3 is used in this work. The filter cascade is used in this experiment to detect corners in an image. In the reduce function of the first filter, Gx^2, Gy^2 and $Gx \times Gy$ are calculated based on values of Gx and Gy. The summation of Gx^2, Gy^2 and $Gx \times Gy$ values over a 3×3 neighbourhood is used to calculate the eigenvalues of the covariance matrix of derivatives. To verify the calculation of the eigenvalues sum of squared differences are calculated according to equation 4. In equation 4 h and w are height and the width of the image. The function eValMR gives the eigenvalues calculated by the filter

Fig. 8. Cameraman Image

TABLE III. EXECUTION TIME OF MEDIAN FILTER

No of images	Image size	Execution time(s)	Average execution time(s)
360	160×120	988.385	2.7455
49	512×512	994.435	20.2946

cascade using MapReduce programming model. The function eValCV gives the eigenvalues calculated by the OpenCV image processing library [5].

$$SSD = \sum_{i=0}^{h-1} \sum_{j=0}^{w-1} \frac{(\mathrm{eValMR}(i,j) - \mathrm{eValCV}(i,j))^2}{h \times w} \qquad (4)$$

The calculated SSD value for the eigenvalues in the above experiment is 2.4080×10^{-16}. If the minimum eigenvalue is greater than a certain threshold it is possible to consider that there exist a corner in that location [11]. The threshold is obtained experimentally to detect the corners. The filter cascade is applied on Fig. 8 to detect corners. The detected corners are shown in Fig. 9 in black dots.

The Median filter is applied on images with 3×3 neighbourhood. The execution time is shown in Table III. It is applied on 360 images of size 160×120. The average execution

Fig. 9. Detected corners as black dots

TABLE IV. EXECUTION TIME OF IMAGE FILTER CASCADE

No of images	Image size	Execution time(s)	Average execution time(s)
360	160×120	1873.478	5.2041
49	512×512	2966.362	60.538

time of one image is 2.7455s. The Median filter is applied on another set of 49 images of size 512×512. The average execution time of such image is 20.2946s.

The image filter cascade is applied on different images to detect corners. The execution time is shown in Table IV.

Image filter cascade is applied on 360 images of size 160×120. The average execution time of one image is 5.2041s. The filter cascade is applied on another set of 49 images of size 512×512. The average execution time of such image is 60.538s.

IV. Conclusion and Future Work

In this work the parallelized nature of the MapReduce programming model is used to create image filters. With this technique it is shown that it is possible to apply filters as a cascade and also as a single filter. One constrain with the filter cascade is all the filters must have the same kernel size. The filter cascade avoids image to be streamed every time a filter is applied. The median filter used to verify the proposed single filter, a corner detection algorithm is used to verify the filter cascade. There were memory limitations with pseudo-distributed mode when large images are processed. Dividing such large images in to multiple parts and applying the filter cascade on a fully-distributed mode of hadoop will be considered as future work.

Acknowledgment

The authors of this paper would like to thank the National Research Council of Sri Lanka for providing the funding and equipment to conduct this research. The Grant No is 12-18.

References

[1] (2015, Jan.) Apache Hadoop 2.4.1 - HDFS Architecture. [Online]. Available: http://hadoop.apache.org/docs/r2.4.1/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html

[2] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.

[3] M. Yamamoto and K. Kaneko, "Parallel image database processing with mapreduce and performance evaluation in pseudo distributed mode," *Int. J. Electron. Commerce Stud.*, vol. 3, no. 2, pp. 211–228, 2013.

[4] Y. Yan and L. Huang, "Large-scale image processing research cloud," in *Proc. 5th Int. Conf. on Cloud Computing, GRIDs, and Virtualization*, Venice, Italy, 2014, pp. 88–93.

[5] (2015, Jan.) Open source computer vision. [Online]. Available: http://opencv.org//

[6] (2015, Jan.) Hipi - hadoop image processing interface. [Online]. Available: http://hipi.cs.virginia.edu/

[7] M. H. Almeer, "Hadoop mapreduce for remote sensing image analysis," *Int. J. Emerging Technology Advanced Eng.*, vol. 2, no. 4, pp. 443–451, 2012.

[8] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 3rd ed. Upper Saddle River, NJ: Prentice-Hall, Inc., 2006.

[9] (2015, Jan.) Apache hadoop. [Online]. Available: http://hadoop.apache.org/

[10] (2015, Jan.) Image processing toolbox. [Online]. Available: http://in.mathworks.com/products/image/

[11] J. Shi and C. Tomasi, "Good features to track," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, Seattle, WA, 1994, pp. 593–600.