Parallel K-Means Clustering of Remote Sensing Images Based on MapReduce

Zhenhua Lv[1], Yingjie Hu[1], Haidong Zhong[1], Jianping Wu[1,*], Bo Li[2], and Hui Zhao[2]

[1] Key Laboratory of Geographic Information Science, Ministry of Education; Geography Department; East China Normal University, Shanghai, China
[2] Institute of Software Engineering, East China Normal University, Shanghai, China
zhenhualv@gmail.com,
jpwu@geo.ecnu.edu.cn

Abstract. The K-Means clustering is a basic method in analyzing RS (remote sensing) images, which generates a direct overview of objects. Usually, such work can be done by some software (e.g. ENVI, ERDAS IMAGINE) in personal computers. However, for PCs, the limitation of hardware resources and the tolerance of time consuming present a bottleneck in processing a large amount of RS images. The techniques of parallel computing and distributed systems are no doubt the suitable choices. Different with traditional ways, in this paper we try to parallel this algorithm on Hadoop, an open source system that implements the MapReduce programming model. The paper firstly describes the color representation of RS images, which means pixels need to be translated into a particular color space CIELAB that is more suitable for distinguishing colors. It also gives an overview of traditional K-Means. Then the programming model MapReduce and a platform Hadoop are briefly introduced. This model requires customized 'map/reduce' functions, allowing users to parallel processing in two stages. In addition, the paper detail map and reduce functions by pseudo-codes, and the reports of performance based on the experiments are given. The paper shows that results are acceptable and may also inspire some other approaches of tackling similar problems within the field of remote sensing applications.

Keywords: K-Means; remote sensing; parallel; MapReduce; Hadoop.

1 Introduction

The K-Means clustering is one of the most common methods of data analysis, as in the field of pattern recognition, data mining, image processing, etc. It is useful in the field of remote sensing analysis as well, where objects with similar spectrum values are clustered together without any former knowledge. It has become the basic algorithm of unsupervised classification, providing us an overview of objects easily and directly, with which our further analysis becomes clear. The time complexity of

* Corresponding author.

F.L. Wang et al. (Eds.): WISM 2010, LNCS 6318, pp. 162–170, 2010.

K-Means, however, is considerable, and the execution is time-consuming and memory-consuming especially when both the size of input images and the number of expected classifications are large. To improve the efficiency of this algorithm, many variants have been developed. It is commonly believed that there are two ways to reduce the time consumption, the first is concerned with optimizing the algorithm itself, whereas another one focuses on changing the proceeding of execution, that is migrate the sequential process to parallel environment. While in our opinion, optimization of sequential K-Means algorithm is important and has made much great success, this paper prefers to have the algorithm running under parallel environment, which will be considered as the appropriate way to process large amounts of data set. Different with traditional methods that implemented based on MPI [1] [2], we use MapReduce as our basic computing model, which is first proposed by Google Corporation in 2004 and has now been widely welcomed in many domains. Related work has been done, however they both have two faults that will be discussed in the third section.

The rest of this paper is organized as follows. In the next section we will briefly describe the traditional approach that clustering RS images sequentially. It includes color space we used and the clustering method K-Means. Then, at the third section we introduce the related work and discuss our implementation. In the fourth section, experiments are conducted. Finally, merits and demerits of this paper are pointed out.

2 Sequential Process

The input of clustering is a color remote sensing image, a set of pixels each of which represented by RGB value. We expect the output, after the K-Means algorithm, is k subsets of pixels, within each subset the pixels have the most similar color. The similarity in color is better understood in a transformed space called CIELAB rather than RGB. So the preparation for clustering is a transformation of each pixel from RGB-value to Lab-value, and the similarity of pixels is represented by the distance between Lab-values.

2.1 Transformation of Color

The L*a*b* color space (also called CIELAB) is derived from the CIE XYZ tri-stimulus values, and it consists of a luminosity 'L*' or brightness layer, chromaticity layer 'a*' indicating where color falls along the red-green axis, and chromaticity layer 'b*' indicating where the color falls along the blue-yellow axis [3]. This color space describes all colors visible to the human eye and is created to serve as a device-independent model. The formula for converting RGB into LAB is given by [4], and this conversion program implemented in MATLAB can be freely downloaded on the web [5]. In this paper, we have made use of another free function for color conversion that was written in Java and available on [6].

We use P(L, a, b) representing a pixel value in LAB color space. Since the color information exists in ‘a*b*’ space, we will measure the difference of two colors by ‘a*’ and ‘b*’ values, ignoring ‘L*’ values. The difference can be defined by Euclidean distance. The greater the distance of two pixels, the less similar they are or vice versa.

2.2 Overview of K-Means Clustering

Let P_1, P_2, ..., P_n be the set of N pixels, where P_i is the i^{th} pixel consisting of a (a, b) pair. In the pair, a, b are color components in L*a*b* color space. Both are calculated from RGB values the way we have discussed in previous section. Since the pixels in image files are arranged along widths and heights like rectangular shapes and each pixel has a two dimensions coordinates, a conversion of pixels to sequential pairs is required. Let C_1, C_2, ..., C_k be the K clusters, where K is a input parameters for the algorithm. Let m_1, m_2, ..., m_k be the centroids associated with their clusters, such that mi is the centroid of cluster C_i , for $1 \le i \le K$.

$$J_e = \sum_{j=1}^{K} \sum_{P_i \in C_i} \left\| P_i - m_i \right\|^2 \tag{1}$$

$$m'_i = m_i + \frac{1}{N_i - 1}[m_i - P_i] \tag{2}$$

$$m'_k = m_k + \frac{1}{N_k + 1}[P_i - m_k] \tag{3}$$

The quantity of clustering is measured by the error variance defined as (1) [7]. Ideally, the optimized solution of clustering is the one that makes J_e reaching the minimum. When P_i has been moved out its cluster C_i, there causes a decrease in the centroid m_i, generating the new centroid m'_i as (2). Similarly, when P_i has been moved into another cluster C_k, there cause an increase in the centroid m_k, generating the new centroid m'_k as (3).

Our work in this paper considers the K-Means algorithm executing as the following steps:

Step 1. Generate an initial K clusters.

Step 2. Move pixels between clusters when the sum of error variance is going to decrease. Select each pixel P, then let C_i be P's cluster, N_i be the number of pixels in this cluster and m_i be the centroid. If N_i =1, then ignore this pixel and select another one, otherwise calculate the variation by (4):

$$v_j = \begin{cases} \dfrac{N_j}{N_j + 1} \left\| P - m_j \right\|^2 & j \ne i \\ \dfrac{N_i}{N_i - 1} \left\| P - m_i \right\|^2 & j = i \end{cases} \tag{4}$$

In (4), i is the number indicating the cluster P belongs to, while j varies from 1 to K. v_j (j=i) is a decrease in the sum of error variance when P has be moved out of its cluster C_i. v_j becomes an increase in the sum of error variance when $j \ne i$ and on the assumption that P has be assigned to cluster C_j. If there exist a $j(j \ne i)$

satisfying $v_j < v_i$, that means moving P from C_i to C_j gives rise to a decrease in sum of total error variance, which is the purpose of this step, otherwise, jump to the beginning of this step. If there existing several j satisfying such condition, the one with the minimal increase would be chosen.

Step 3. Move the pixel from C_i to C_j and update m_i to m_j.

Step 4. Iterates this procedure for N times.

This algorithm doesn't stop until there is no variation of J_e took place or maximum of iteration specified by implementation is reached.

3 Parallel Process

MapReduce [8] is a programming model and Hadoop [9] is an open source system that implements the MapReduce model. The basic primitives of this model are map/reduce functions both are written by users.

3.1 Related Work

In paper [10], algorithm is divided into four steps briefly as follows.

Step 1: generate initial guess.

Step 2: Map: (null, data-instance) -> list (cluster-id, (data-instance, 1)), where the in-key is null, and the in-value is the data instance vector.

Step 3: Reduce: (cluster-id, (data-instance, count)) -> list (cluster-id, (sum-of-data-instances, number-of-instances))

In the output of reduce, the sum-of-data-instances divided by number-of-instances is the mean of the cluster. Simply put, their methods can be summarized as:

1. Random generate k points as initial k means.
2. Apply K-Means Map & Reduce.
3. While not done, go to Step 2.

The idea of paper [11] resembles the previous one. A slight difference is they add a combiner function to improve the performance. However, both the methods may have two faults in the view of processing RS images. For one thing, the final output is only the centroid representing the cluster, which is useless for RS classification. Instead, the output we expect should indicate the cluster of each pixel, such as appending a cluster identifier to each pixel instance. For another, dynamically updating the centroid is missing in their methods, and error variances are not considered either. Therefore, the final output will largely depend on the initial centroid specified by the user, giving rise to great deviations if inappropriate centroid is selected at the beginning.

3.2 Implements on Hadoop

In the parallel environment, dynamically updating centroid in each step needs global communication between computer nodes, which should be avoided. Hence, in our

method pixels are clustered partially. In general, there are two Hadoop jobs for calculations, one is to distribute each pixel into its initial cluster, and another is to move pixels minimizing the error variances.

To simply suit the framework, input images are translated into text file. Each line represents a pixel like (file_id, pixel_id, r, g, b), where file_id identifies the file, pixel_id is the line number in each file, and r, g, b are the values.

Our algorithm can be detailed as two steps. Each of them is represented by a job of Hadoop. The first step is to initial clustering, in which pixels closest to the centroid are clustered. To identify the owner cluster of a pixel, we insert the cluster number to a text line that represents the pixel. A job of the Hadoop has three functions, they are configure, map and close. In the configure function, the job reads some necessary variables from the JobConf object that are distributed by the name node. The variables are initial centroids specified by the user's description file. The map function performs chief calculations of clustering. The close function saves the final centroids to a file (called centroid file). Since there are many map tasks running on different computer nodes when the job is committed to Hadoop, the input pixels of one map task are only a subset. There are many centroid files, and they should be merged for the next step. The method that merges centroid files here is to simply take the mean of each cluster centroid as the new centroid.

The second step is to minimize the error variances. The method how to minimize the error variances is mentioned in the previous section and it is abided by (6). Reduce functions in both the jobs do no extra work but write the final results to files (called result file). Thus the results files contain cluster values indicating each pixel's cluster when all the jobs are accomplished.

It needs two job client objects in Hadoop to implement these two steps. The output of the first job is in turn regarded as the input of the second job. The two jobs run step by step.

Step 1. Initial clustering. The pseudo-codes describe the job 'Initial clustering' as shown in Table 1.

In 'Initial clustering', DETA is the increase of centroid when a new pixel is moved in its cluster. It can be calculated by the expression (3). The input 'CountsInCluster' is a counter list in which each item represents the number of pixels that a cluster contains. It is initialized at the configure function. The 'lab_distance' method is to calculate Euclidean distance of two pixels.

Step 2. Minimize the total error variance for each cluster. This job reads its input from the files that are produced by the first job. At the configure function of this job, it merges some partial results. For example, centroids of each cluster which are produced by each map task are collected together. And some 'CountsInCluster' values are also added together. The map function of the job 'Minimize Error' is shown in Table 2. The idea is to move the pixel to another cluster if the sum of error variance decreases. The error variance of a cluster will decrease when a pixel of it moves out and will increase then a pixel moves in. If the amount of increase is less than that of decrease, the pixel is of course worth moving.

Table 1. Initial Clustering

InitalClusteringJob.map

Input : (in_useless, in_pixel), where 'in_pixel' is (file_id, pixel_id, r, g, b);
Centroid{K} = getFromConfiguration(); // assign K initial centroid
CountsInCluster{K} = {1};// each cluster has 1 pixel at the beginning
Output : (out_cluster_id, out_ pixel)

```
rgbPixel = parseFrom(in_pixel);        labPixel = Transform(rgbPixel);
minDistance = Double.MAX_VALUE;   closestClusterId = -1;
for (i = 0; i < K; ++i) {
    dist = lab_distance(input, Centroid[i]);
    if (dist < minDistance) { minDistance = dist;    closestClusterId = i; }
}
// move pixel into the closest cluster and update the centroid
++ CountsInCluster [closestClusterId];
Centroid [closestClusterId] = Centroid [cid] + DETA;
out_cluster_id = closestClusterId;
output = {closestClusterId, file_id, pixel_id, labPix}
```

Table 2. Minimize Error

MinimizeErrorJob.map

Input : (cluster_id, labPixel)
Centroid{K}; //a set of centroids
Counts{K}; //Counts[i] indicates the count of pixels in cluster i
Output : (cluster_id, out_ pixel)

```
// move the pixel to another cluster if error variance decreases
icount =  Counts [cluster_id];    idestCluster = -1;
cen = Centroid [cluster_id];  double tempMin = decrease;
decrease = icount / (icount - 1) * lab_distance(labPixel, cen);
for (int k = 0; k < ClusterNumber; ++k) {
     if (k != cluster_id) {
       increase = icount / (icount+1) * lab_distance(input, cen);
     if (increase < tempMin) { idestCluster = k; tempMin = increase;} } }
 if ( idestCluster != -1) {
    cen = cen + (cen - labPixel) / (icount - 1);
    cen2 = Centroid[idestCluster];
    cen2 = cen2 + (labPixel – cen2) / (icount + 1);
    Centroid[cluster_id] = cen;
    Centroid[idestCluster] = cen2;
    --m_Counts[cluster_id]; ++m_Counts[idestCluster];
   output = { idestCluster, labPixel}  }
```

The K-Means algorithm requires that iterations for minimizing error variances run more than one times and select the best result. In our algorithm the iteration runs only once, because we found the first iteration decreasing the error variances immensely and the consequent iterations take trivial effects.

4 Experiments

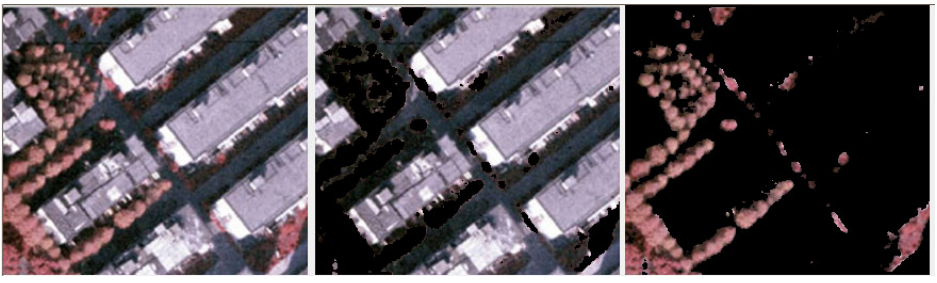
The hardware we used was a computer cluster which had one control node and eight computing nodes. The control node was composed of eight 2.0GHz-CPUs and 4GB memory, while each computing node was composed of eight 2.0GHz-CPUs and 8GB memory. All nodes were connected by a gigabit switch. The total capacity of the cluster is about 20TB.

We implemented sequential K-Means algorithm by using C++ and it ran well if the size of the input image is not large. In the experiment of sequential algorithm the size cannot exceed 1000 pixels both in width and height. We also implemented another version called parallel K-Means running on Hadoop in Java. The image used here contained vegetations and constructions as Fig. 1 (a) shown. The two images after clustering were divided as Fig. 1 (b) and (c) shown. The same image was also taken as the input of parallel K-Means on Hadoop, and the results were shown in Fig. 2 (b) and (c) separately. It is obvious that the results on different environments are very similar. Their corresponding error variances of these two algorithms were summarized, as Table 3 shown. The column deviation was calculated by taking the results of sequential K-Means as criterion.

Another kind of images was performed by the parallel K-Means, but we cannot visualize them because their sizes and amounts were very large. They cannot be examined by the sequential algorithm either. The data were aerial remote sensing images about some districts of Shanghai in China. The data were composed of many blocks and seemed like lattice, each of which was an image with tif format. Each image was 4000 pixels in width and 3200 pixels in height. The size of each image file was about 38 megabytes. But the amount of the file size soars to about 240 megabytes when the form of the pixels was changed into text. It was unwise to change the form, but there was no way to directly operate tif-format images on Hadoop. As Table 4 shown, the cost of time increased dramatically when the count of images became larger. This might be caused by the data transmission of network.

(a) (b) (c)

Fig. 1. Results of sequential K-Means, where K is two. (a) is the original image. (b) is one of its clusters and (c) is another cluster.



(a) (b) (c)

Fig. 2. Results of parallel K-Means on Hadoop, where K is two. (a) is the original image. (b) is one of its cluster and (c) is another cluster.

Table 3. Deviations of error variances

Cluster Name	Error Variance of Sequential K-Means	Error Variance of Parallel K-Means	Deviation
Vegetation	1013060.91	946741.75	-6.55%
Construction	3040686.81	3104197.87	2.09%

Table 4. Time cost

Image count	Size in pixels (width * height)	File size in bytes	Time cost (second)
1	4,000*3,200	248M	36
4	8,000*6,400	995M	125
20	20,000*12,800	4.85G	778
80	40,000*25,600	19.5G	5413

5 Conclusion and Future Work

In this paper, the pixels are firstly clustered in the partial scope, and then they are collected together and merged. In the view of mathematics, this slightly deviates from the origin of the algorithm. But, with the amount of input increasing, the deviations of this method slow down and can satisfy the concrete work. Since the Hadoop APIs have not supported to handle images as input directly by now, there need a transformation for images from binary into text files. This inevitably enlarges the size of input data and reduces the performance. In the near future it may provide a direct way processing images on Hadoop.

In fact, the final purpose of clustering RS images is to picture an overview of the data and provide us with approximate figures for further research. There is no technique that performs clustering RS images completely satisfying the need of work without artificial participation. Hence, instead of evaluating the results by error variances or other methods, visualizing the output and judging by human eyes directly

are more practical. We have developed a simple tool for visualizing, but it is not suitable for handling large images. And this is another point of improvement for our work.

References

1. Gursoy, A.: Data Decomposition for Parallel K-means Clustering. Parallel Processing and Applied Mathematics 14, 241–248 (2004)
2. Li-shun, J., Ding-sheng, L.: Research on K-Means Clustering Parallel Algorithm of Remote Sensing Image. Remote Sensing Information 01, 27–30 (2008)
3. Matlab R2008a Product Help. Demos/Toolboxes/Image Processing/Image Segmentation/ Color-Based Segmentation Using the L*a*b* Color Space
4. Kartikeyan, B., Sarkar, A., et al.: A segmentation approach to classification of remote sensing imagery. International Journal of Remote Sensing 19, 1695–1709 (1998)
5. MATLAB Central - File detail - RGB2Lab, http://www.mathworks.com/matlabcentral/fileexchange/24009
6. Color Inspector 3D - Color Space Conversions, http://www.f4.fhtw-berlin.de/~barthel/ImageJ/ColorInspector/HTMLHelp/farbraumJava.htm#rgb2lab
7. Zhaoqi, B., Xuegong, Z.: Pattern Recognition, 2nd edn., pp. 235–237. Tsinghua University Press, Beijing (2000)
8. Dean, J., Ghemawat, S.: Mapreduce: Simplified data processing on large clusters. Communications of the ACM 51, 107–113 (2008)
9. Hadoop Website, http://hadoop.apache.org/
10. Yao, K.T., Lucas, R.F., et al.: Data Analysis for Massively Distributed Simulations. Interservice/Industry Training, Simulation, and Education Conference(I/ITSEC) (Got from Google Scholar) (2009)
11. Zhao, W., Ma, H., et al.: Parallel K-Means Clustering Based on MapReduce. In: CloudCom 2009. LNCS, vol. 5931, pp. 674–679 (2009)