International Journal of Innovative
Computing, Information and Control
Volume **13**, Number **3**, June **2017**

 ISSN 1349-4198
pp. **1037–1046**

AN EFFECTIVE DETECTION OF SATELLITE IMAGES VIA K-MEANS CLUSTERING ON HADOOP SYSTEM

Mengzhao Yang, Haibin Mei and Dongmei Huang*

College of Information
Shanghai Ocean University
No. 999, Huchenghuan Rd., Nanhui New City, Shanghai 201306, P. R. China
mzyang@shou.edu.cn; *Corresponding author: dmhuang@shou.edu.cn

Received October 2016; revised February 2017

Abstract. *Nowadays detection of satellite images is generally difficult due to massive volume of images in Big Data era, so high processing speed has become an indispensable requirement for some special applications such as rapid response to disaster warning. In this paper, we present an effective detection of satellite images via K-Means clustering on Hadoop system. K-Means algorithm is one of the most popular and widely used clustering techniques to detect satellite images. We design the effective K-Means algorithm based on MapReduce programming model and Hadoop distributed file system. Two main operations in MapReduce: Map and Reduce, are realized to give an efficient implementation. The results show that we can acquire a fast detection speed and good scaleup while keeping accuracy both in training and testing.*
Keywords: Satellite images, Effective detection, K-Means clustering, Hadoop system

1. **Introduction.** Satellite image processing provides different services for analysis, and to support GIS (Geographic Information Systems) and other research applications. In recent years, the number of satellite images has grown considerably due to the growth of high resolution satellites. This rapid growth and huge volume of images have created a new field in image processing which is called Big Data and cloud computing [1] that nowadays is positioned among state-of-the-art technologies.

In the field of remote sensing images, deforestation, climate change, ecosystem and land surface temperature are some of the main research areas, where features need to be classified and clustered to provide research basis. Many clustering algorithms such as PCA (Principal Component Analysis) [2] and C-Means [3, 4] are based on the assumption that the image in question depicts one or more features and each of these features belongs to one of several distinct and exclusive classes. K-Means, a more effective algorithm over C-Means and PCA clustering, is one of the most popular and widely used clustering techniques across all fields, including remote sensing processing [5, 6]. However, when dealing with current massive data called Big Data, they have to spend much more time implementing K-Means algorithm. In order to improve the implementation speed, parallel clustering algorithms [7, 8] have been efficiently researched to meet the scalability and performance requirements. However, all these parallel clustering algorithms have the following drawbacks: (a) they assume that all objects can reside in main memory at the same time; (b) their parallel systems have provided restricted programming models and used the restrictions to parallelize the computation automatically.

Google as one of the leading companies in the field of Big Data, proposes the MapReduce programming model [9] which is designed to process large amounts of distributed

data efficiently. MapReduce is a programming model which is an associated implementation for processing and generating large data sets with a parallel, distributed algorithm on a cluster [10]. However, MapReduce is only a computing architecture, and has no sharing and storing files system. So distributed file system has to be combined into MapReduce model to construct an effective cloud computing environment. Over recent years, Hadoop system has become a highly popular solution to store and process a large amount of data for analysis purpose. Hadoop is an open source framework for processing, storage, and analysis of large amounts of distributed and unstructured data [11]. Sharing, storing, and retrieving large files on a Hadoop cluster is undertaken by its distributed file system called HDFS (Hadoop Distributed File System). To increase the reliability of the system, each part of the file is distributed among multiple computing nodes. Therefore, if a node stops working, its file can be retrieved again. Trying to explore the feasibility using MapReduce model for doing large scale image processing, Hadoop MapReduce for remote sensing image analysis [12, 13] aims to find how to design an efficient solution method for customized process within the Hadoop framework, which has become a research focus recently. Although there have been considerable researches utilizing the Hadoop platform for image processing, practical algorithms on Hadoop need to be explored further in implementation performance. Hence, the utilization of Hadoop for image processing has focused on some practical image processing algorithms. High computational complexity and long time computation are still main problems for detection of large size images. It greatly affects the response time of detection for disaster warning. Recently, one of practical algorithms is the parallelization of Fuzzy C-Means (FCM) [14], which is investigated using the MapReduce paradigm that is introduced by Google [9]. Although its implementation works correctly achieving competitive purity results compared to state-of-the-art clustering algorithms, K-Means implementation scales better than FCM algorithm since it is the less computationally expensive algorithm. Also FCM may be not well in accordance with the compact degree and the membership of the data.

In order to improve scaleup and handle massive volumes of satellite images quickly, a high processing speed has become an indispensable requirement when providing a rapid detection for some special applications such as warnings of disasters. In this paper, this is possible with the help of Big Data platforms: Hadoop system. We have designed parallel K-Means algorithm based on MapReduce on Hadoop system and find the optimum value of K by executing the K-Means algorithm with modified values of K in the same set of MapReduce iterations. The results show that we can obtain less execution time and higher speedup ratio, especially when the image data size is larger. Also scaleup can be reasonably reduced with increasing nodes number and test dataset size in proportion, which indicates the achievement of this proposed method.

The rest of this paper is organized as follows. The MapReduce programming model is given in Section 2. Design of parallel K-Means algorithm on Hadoop system is given in Section 3. Experimental results are discussed in Section 4, and finally conclusions are given in Section 5.

2. MapReduce Programming Model.

MapReduce is a programming model designed for processing large volumes of data in parallel by dividing the work into a set of independent tasks. From a user's perspective, there are two basic operations in MapReduce: Map and $Reduce$. The Map function reads a stream of data and parses it into intermediate $< Key, Value >$ pairs. When that is complete, the $Reduce$ function is called once for each unique key that was generated by Map, and is given the key and a list of all values that were generated for that key as a parameter. The execution model for programs written in the MapReduce style can be roughly characterized by three steps as follows:

1. Split inputting images
   a. inputting images→ split→ $\{split\}$
2. Mapper task: process splitting images
   a. $\{split\}$→read→$\{< k_1, v_1 >\}$
   b. $\{< k_1, v_1 >\}$→map & partition→$\{< k_2, v_2 >\}$
3. Reducer task: process mapper output images
   a. $\{< k_2, v_2 >\}$→shuffle & sort→$\{< k_2, \{v_2\} >\}$
   b. $< k_2, \{v_2\} >$→reduce→$< k_3, v_3 >$
   c. $\{< k_3, v_3 >\}$→write→output images

MapReduce is implemented in a master/worker configuration, with one master serving as the coordinator of many workers. A worker may be assigned a role of either a map worker or a reduce worker. MapReduce execution model can be graphically represented in Figure 1. There may be M_N mapper tasks (step 2) and R_N reducer tasks (step 3) executed in parallel.

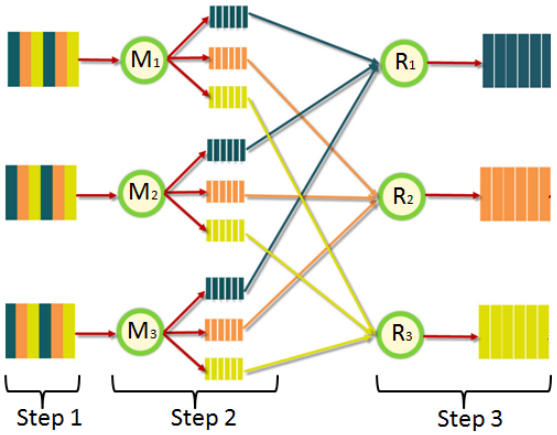

FIGURE 1. MapReduce execution model

As shown in Figure 1, three steps are described in more detail as follows.

Step 1: A usually very large input file is subdivided into a number of logical "splits". Each split starts and ends at record boundaries specific to a given file format. In the case of image files, a record may represent a single line in one 2D matrix and thus, splitting occurs at line endings.

Step 2: Each split is passed as input to a mapper task. Up to M_N mapper tasks may run in parallel. The mapper tasks read the split and convert it into a vector of key-value pairs $< k_1, v_1 >$ (step 2.a). If the input is an image file, the key k_1 could be the line number and the value v_1 is the pixel value. Each input pair $< k_1, v_1 >$ is then passed to the user supplied Map function which transforms it into zero, one or more intermediate key-value pairs $< k_2, v_2 >$ (step 2.b).

Step 3: R_N reducer tasks run in parallel. Each reducer gets one or more specific partitions of the output of a mapper. The partition number ranging from 1 to R_N is computed from each intermediate key-value pair $< k_2, v_2 >$ by using a partitioning function (usually a hash function of k_2). This step is already performed by mapper tasks. Each reducer task reads all the intermediate key-value pairs $< k_2, v_2 >$ of its partitions of all mappers, merges and sorts them by key k_2 (step 3.a). All values v_2 that have same keys k_2 are aggregated in a list and passed as $< k_2, v_2 >$ to the reducer function. The reducer function will reduce all the intermediate values v_2 for a given key k_2 and output a new key-value pair $< k_3, v_3 >$ (step 3.b). Finally the new key-value pairs are collected, formatted and written to the output file (step 3.c).

3. **Design of Parallel K-Means Algorithm Based on MapReduce Model on Hadoop System.** In the parallelization of K-Means algorithm, each iteration corresponding to a Job, is a MapReduce operation: calculating the distance of each data object to all cluster centers and sending the data object corresponding to a Map task, updating the cluster center corresponding to a Reduce task. The data records of dataset are stored in rows, in order to expediently start Map task. Therefore, each Map task automatically gets a record for executing and is finished automatically by MapReduce on Hadoop system.

3.1. **K-Means algorithm.** The K-Means algorithm is an efficient iterative method to partition a given dataset into a user specified number of clusters, K. Its objective is to minimize the average squared Euclidean distance of documents from their cluster centers. Let μ_c^k denote the mean for cluster center c, and the K-Means objective function can be written as

$$J(c,\mu) = \sum_{i=1}^{k} \sum_{j=1}^{n} \parallel x^j - \mu_c^i \parallel^2 \tag{1}$$

where J measures the sum of squared distances between each training example x^j and the cluster centroid μ_c^i to which it has been assigned. It can be shown that K-Means is exactly coordinate descent on J. Specifically, the inner-loop of K-Means repeatedly minimizes J with respect to c while holding μ fixed, and then minimizes J with respect to μ while holding c fixed.

With this function well defined, we can split the process in several steps, in order to achieve the wanted result. Our starting point is a large set of data entries and a K defining the number of centers. The K-Means clustering algorithm is as follows.

1) The first step is to choose K of our points as partition centers randomly.

2) Next, we compute the distance between every data point on the set and those centers, and store that information.

3) Supported by the last step calculations, we assign each point to the nearest cluster center. This is, we get the minimum distance calculated for each point, and we add that point to the specific partition set.

4) Update cluster center positions by using the following formula:

$$\mu_c^i = \frac{1}{\mid k_i \mid} \sum_{j \in k}^{k} x^j \tag{2}$$

5) If the cluster centers change, we repeat the process from Step 2. Otherwise we have successfully computed the K-Means clustering algorithm, and got the partition's members and centroids.

The achieved result is the minimum configuration for the selected start points. It is possible that this output is not the optimal minimum of the selected set of data, but instead a local minimum of the function. To mitigate this problem, we can run process more than one time in order to get the optimal solution.

3.2. **Design of Map function.** The mapping phase of the Map/Reduce approach applies a function to each input value, producing a list of Key/Value pairs for each input. All these lists (each containing several Key/Value pairs) are gathered into another list to constitute the final output of the mapping phase as shown in Figure 2.

The task of Map function is calculating the distance from data object to cluster centers and sending the data object to the nearest cluster. The input data format is $< Key, Value >$, where key is the number of row, and value is the data record. In

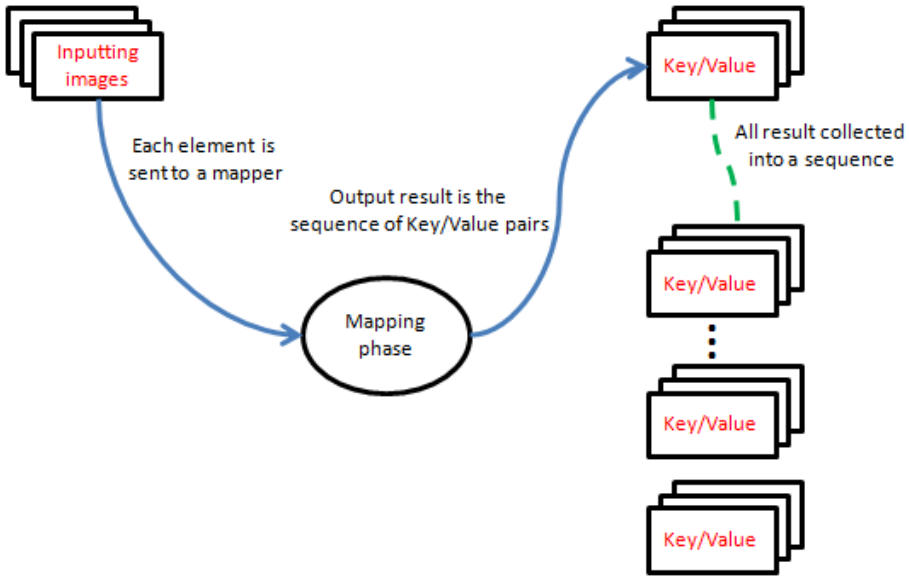

FIGURE 2. Process of mapping phase

addition, each Map function also reads K cluster centers, which are initial cluster centers or the cluster centers after several iterations. The format of Map function output is $< Key, Value >$, where key is clustering ID, and value is data record. The pseudo code of Map function is shown as follows:

```
1  void Map (Writable key, Image point) {
2      min_distance = MAXDISTANCE;
3      for(i = 0;i < k;i++){
4          if(distance(point,cluster[i]<min_distance)){
5              min_distance = distance(point,cluster[i]);
6              currentCluster_ID = cluster_number;
7             }
8      }
9      Emit_Intermediate(currentCluster_ID,point);
10 }
```

In this pseudo code, the 'for' circulation is traversing K points. If there is a distance less than *min_distance*, then we re-assign the *min_distance* and get serial number of the cluster.

3.3. **Design of combination phase.** The combination phase takes the output of the mapping phase, and collects each key and associated values from the collection of lists of Key/Value pairs. The combined output is then essentially a map with unique keys created during the mapping process, and each associated value is a list of values from the mapping phase. Process of combination phase is shown in Figure 3.

In this phase, the partial sums are calculated and an associated counter for each group is updated. The input is $< currentCluster\_ID, point >$ pairs from the Map phase. The output is pairs of $< currentCluster\_ID, R >$ where R is a data structure containing a partial sum and the count of summed up patterns. The phase proceeds as below.

1) Add up pairs according to their corresponding centroid identifiers.

2) Update the counters and partial sums in the R data structure for each unique centroid identifier.

3) The output is $< currentCluster\_ID, R >$ pairs.

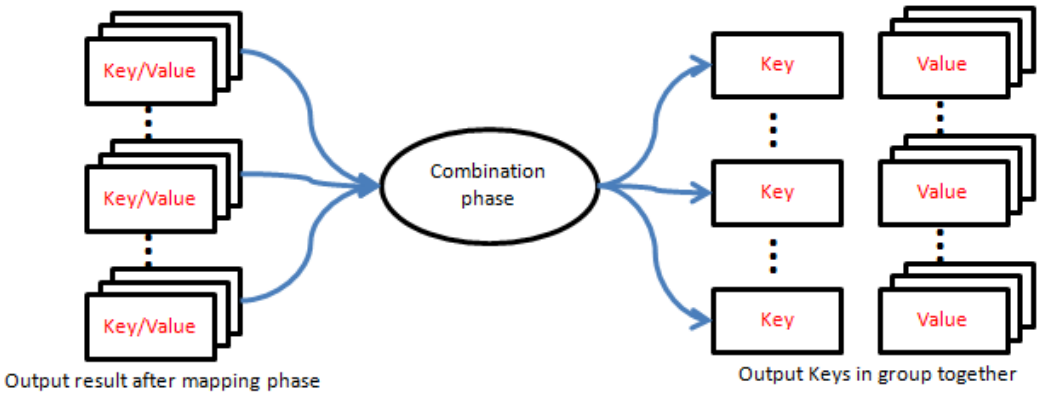

FIGURE 3. Process of combination phase

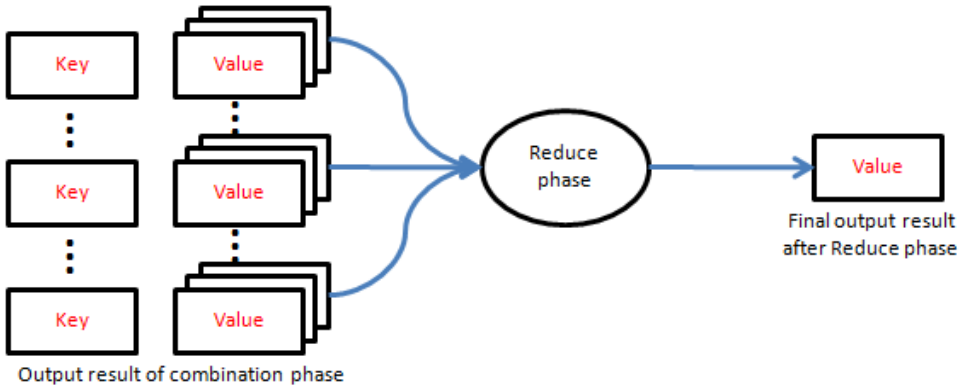

FIGURE 4. Process of Reduce phase

3.4. **Design of Reduce function.** The input to the Reduce phase is the output of combination phase, which is a map, with keys being all the unique keys found in the mapping operation and the values being the collected values for each key from the combination phase. The output of the Reduce phase can be any arbitrary value. Process of Reduce phase is shown in Figure 4.

After obtaining the intermediate results from combination phase, Reduce function will update the corresponding center of the cluster ID and output the new center of each cluster. These data records sets are composed of all the intermediate results from the combination task. The pseudo code of Reduce function is shown as follows:

```
void Reduce(Writable key, Iterator<PointWritable>points){
    num = 0;
    while(points.hasNext()){
        PointWritableCurrentPoint = points.next();
        num += currentPoint.getNum();
        for(i = 0; i < dimension; i++){
            sum[i]+= currentPoint.point[i];
        }
    }
    for(i = 0; i < dimension; i++){
    mean[i] = sum[i]/num;
    }
    Emit(key, mean);
}
```

In this pseudo code, the 'while' circulation is traversing each point, and then calculating the summation of each dimension of each point in clusters. Finally, it will respectively calculate the mean of each dimension and determine the new cluster center that is the mean.

When the Reduce tasks are completed, the change of the center will be calculated to see whether it is within a threshold range. If it is in the threshold range, the final result will be output; otherwise, it will start a new MapReduce task following the K new centers to iterate again. Before getting the final clustering results, it needs to start another Job which only includes Map task to allocate all data points.

4. **Experiment and Results.** We have implemented the effective detection of satellite images via K-Means clustering on Hadoop system. The servers are configured in sets of 3, 5, 7 and 9 on the cluster on which the K-Means algorithm is run. The single processor speed is 3.2GHz and of type AMD Opteron 6-Core 4180. Therefore, each server has six processors and each processor has six cores. The servers are running Centos OS and each has 32GB memory. The program used for the experiments is implemented in the Scala programming language. In addition, Apache Hadoop 2.6 is used as the framework for distributing the computations on the cluster. In the experiments, the programming framework is used to provide the needed connectivity. Its component packages allow access to the MapReduce and the HDFS. Our data set includes satellite images varying sizes from 475KB to 15MB with number of pixels in the range of 2 million to 14 million. Multiple sequence files are created with the data set varying from 1GB to 32GB.

The time taken by the K-Means algorithm is recorded in minutes and each value recorded is an average of three repeated trials for a given data set on a particular cluster configuration. The max number of iterations for all the experiments is 1000, with four centers or clusters, and none of the data sets used converge within those iterations. To the different size image files, Table 1 shows comparison of execution time between sequential codes and our method.

Table 1. Comparison of execution time between sequential codes and our method

Different Methods	**Small Size**	**Middle Size**	**Large Size**
Sequential codes	1089m	4623m	5867m
Our method	32m	92m	103m

In order to give a clearer comparison, Figure 5 shows the detection speedup ratio of the three different size images comparing with sequential execution. Apparently, the image size determines the speedup due to the I/O and CPU processing time. In our Apache Hadoop 2.6 system, it is designed to achieve better performance and shows a maximum 56.96 times speedup ratio for large size images.

Also we compare different algorithms to highlight effectiveness of our method. Using our method we have obtained about 38m when implementing detection on 1G size images; however, the C-Means [14] on Hadoop system acquires 68m. The difference of detection time between our method and C-Means [14] on Hadoop is more obvious when increasing size of images greatly. The difference of detection on different size image files is between two methods as shown in Figure 6.

From Figure 6, our method is about 1.79-1.87 times faster than C-Means [14] while keeping the same detection accuracy both in training and testing. This is because K-Means on Hadoop is less computationally expensive over C-Means on Hadoop in the membership update computation. Also the resulting membership values of C-Means do

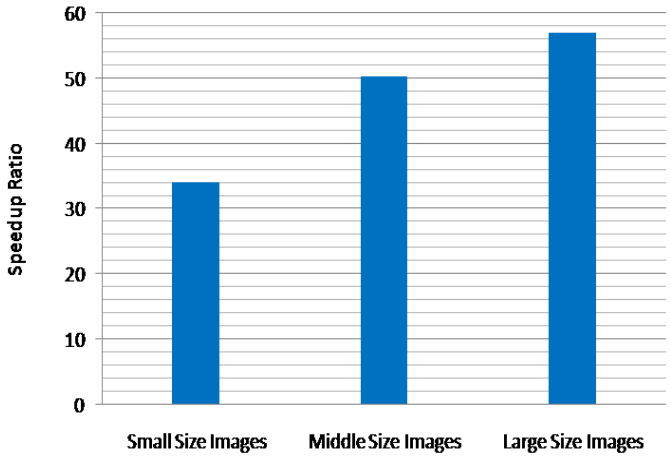

FIGURE 5. Speedup ratio of the three different size images

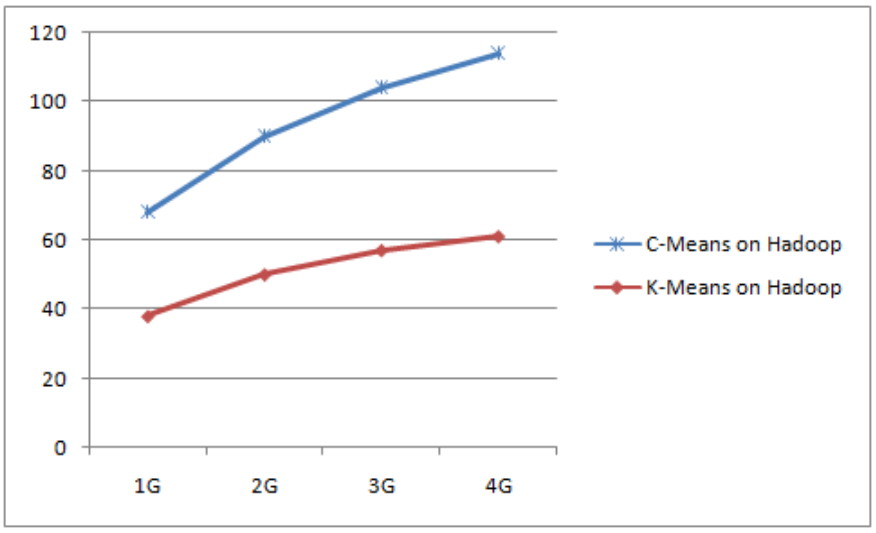

FIGURE 6. Detection time on different size image files

not always correspond well to the degree of belonging of the data, and it may be inaccurate in noisy environment.

In our experiment, the Hadoop system has good robustness and scalability. Compared with the traditional parallel program [8], MapReduce programs are able to complete jobs even one or more computing nodes in a cluster are down. New nodes could be added into the Hadoop system at runtime to meet dynamic requirements, thus get better performance in most cases and provide elastic computing as needed. Scaleup is demonstrated in this framework as well as speedup, and we use three group datasets to implement and test it. The numbers of data points in the datasets used are as follows: 1G, 2G, 4G, 8G in the first group, 2G, 4G, 8G, 16G in the second group, and 4G, 8G, 16G, 32G in the third group. Data in each group is implemented on 1 node, 2 nodes, 4 nodes, 6 nodes and 8 nodes respectively. Figure 7 shows the test result of scaleup performance.

From Figure 7, for the same dataset, when the number of nodes and the size of the test dataset increase in proportion, the scaleup has linearly and reasonably reduced using Hadoop with the proper configuration of HDFS and MapReduce. This is because when the number of nodes increases, the communication cost between the nodes will also linearly

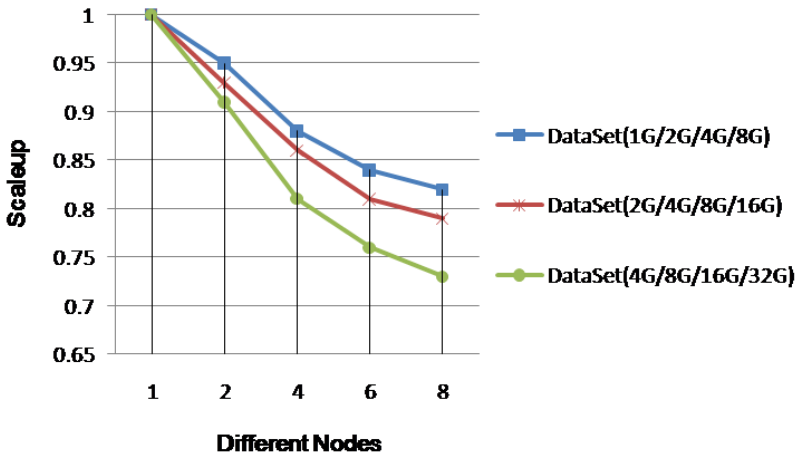

FIGURE 7. Testing scaleup on three group datasets

increases, which is in agreement with true computation and shows our method scales very well.

5. **Conclusions.** We have implemented an effective detection of satellite images via K-Means clustering on Hadoop system. From experiment and results, we can see comparison of execution time between sequential codes and Hadoop system. A clear speedup ratio is acquired on different size images and it is shown a maximum 56.96 times speedup ratio for large size images. Also our method is about 1.79-1.87 times faster than C-Means on Hadoop [14] while keeping the same detection accuracy. Furthermore, scaleup is demonstrated in this framework as well as speedup and reasonably reduced with increase of nodes number and test dataset size in proportion.

To our knowledge, Hadoop is not good at frequent interactive operation. Apache Spark is a new engine for big data processing with in-memory computing nature. So Spark programs can better utilize the cluster resources to solute the low latency problem. In next step, we will try to move to Spark platform, and evaluate the performance of detection.

Acknowledgment. This work was supported by the National Natural Science Foundation of China (No. 41671431), the Ability Construction Project in Local University of Shanghai Science and Technology Commission (No. 15590501900), the Youth Science and Technology Project of Shanghai Ocean University (No. A2-0203-00-100216), and the Doctoral Start-up Fund for Scientific Research of Shanghai Ocean University (No. A2-0203-00-100346).

REFERENCES

[1] K. Michael and K. W. Miller, Big data: New opportunities and new challenges, *Faculty of Engineering and Information Sciences*, pp.22-24, 2013.

[2] C. Alzate and J. A. K. Suykens, Image segmentation using a weighted kernel PCA approach to spectral clustering, *IEEE Symposium on Computational Intelligence in Image & Signal Processing*, pp.208-213, 2007.

[3] K. S. Chuang, H. L. Tzeng, S. Chen and T. J. Chen, Fuzzy C-Means clustering with spatial information for image segmentation, *Computerized Medical Imaging & Graphics*, vol.30, no.1, pp.9-15, 2006.

[4] W. Cai, S. Chen and D. Zhang, Fast and robust fuzzy C-Means clustering algorithms incorporating local information for image segmentation, *Pattern Recognition*, vol.40, no.3, pp.825-838, 2007.

[5] J. Zheng, Z. Cui, A. Liu and Y. Jia, A K-Means remote sensing image classification method based on AdaBoost, *Proc. of International Conference on Natural Computation*, pp.27-32, 2008.

[6] Y. F. Zhong and L. P. Zhang, Initialization methods for remote sensing image clustering using K-Means algorithm, *Systems Engineering & Electronics*, vol.32, no.9, pp.2009-2014, 2010.
[7] Z. Lv, Y. Hu, H. Zhong and B. Li, Parallel K-Means clustering of remote sensing images based on MapReduce, *Web Information Systems and Mining*, pp.162-170, 2010.
[8] J. Zhang, G. Wu, X. Hu, S. Li and S. Hao, A parallel K-Means clustering algorithm with MPI, *The 4th International Symposium on Parallel Architectures, Algorithms and Programming*, pp.60-64, 2011.
[9] R. Mmel, Google's MapReduce programming model-revisited, *Science of Computer Programming*, vol.70, no.1, pp.1-30, 2008.
[10] J. Dean and S. Ghemawat, MapReduce: Simplified data processing on large clusters, *Communications of the ACM*, vol.51, no.1, pp.107-114, 2008.
[11] X. Pan and S. Zhang, A remote sensing image cloud processing system based on Hadoop, *IEEE International Conference on Cloud Computing and Intelligence Systems*, pp.492-494, 2012.
[12] M. H. Almeer, Hadoop MapReduce for remote sensing image analysis, *International Journal of Emerging Technology and Advanced Engineering*, vol.2, no.4, pp.637-644, 2012.
[13] S. Vemula and C. Crick, Hadoop image processing framework, *IEEE International Congress on Big Data*, pp.506-513, 2015.
[14] S. A. Ludwig, MapReduce-based fuzzy C-Means clustering algorithm: Implementation and scalability, *International Journal of Machine Learning & Cybernetics*, vol.6, no.6, pp.923-934, 2015.