Implementing K-means Clustering in Hadoop MapReduce

S14-18645 Semester Project

Chen Wang
Department of Electrical and
Computer Engineering
Carnegie Mellon University
Pittsburgh, Pennsylvania 15217
Email: chenw@cmu.edu

Sung Jae Lee
Department of Electrical and
Computer Engineering
Carnegie Mellon University
Pittsburgh, Pennsylvania 15217
Email: sungjael@andrew.cmu.edu

***Abstract*—k-means clustering algorithm is a popular and one of the simplest technique used in numerous applications that computes from various datasets. However, when it comes to handling large datasets, time it takes to cluster does increase significantly to the size of datasets. To handle large scale of datasets more efficiently, In this paper, we distributed the work of k-means with MapReduce technology that processes large data sets with parallel algorithm. By testing with various data sets with different number of nodes, we proved that MapReduce can speed-up k-means effectively as datasets gets larger.**

***Keywords*—k-means, Hadoop, MapReduce, Clustering.**

I. Introduction

As there emerges applications handling large datasets, there needs a way to speed up the computation time. We have an existing project that computes and search from a huge image database. Instead of matching searched imaged to images from huge datasets, our application extracts visual features from training images and clusters those features into visual words. Before we implemented the parallel way, naive single machine k-means was used for clustering. However, the problem we want to fix and improve is that it was taking too much time for single node to cluster the data. What we are proposing with this paper is that we can speed up the k-mean computation time by implementing map and reduce that can be run in multiple nodes. By paralleling using N workers with Amazon EMR instances, we expect a speed up above $(N-1)x$.

This paper is organized as following, we introduce related works done by others that approaches the way to implement parallel k-means. Then in section 3, we describe what we did to make the k-means parallel. Then we show test results to prove how our parallel k-means speeds up the computation time with multiple datasets.

II. Related Works

A. *Pk-means [1]*

Their Pk-means (Parallel k-means) algorithm performs the procedure of assigning each sample to the closest center while the reduce function performs the procedure of updating the new centers. They also developed a function called combiner to decrease the cost of network communication, it deals with partial combination of the intermediate values with the same ky within the same map task.They tried 1gb to 8 gb of datasets and 1 to 4 numbers of nodes to check the speeds up of their parallel K-Means. As a result, They did meet 2x speed up for 2 nodes and 3.3x speed up for 4 nodes with 8gb of data.

B. *MapReduce Design of k-means [2]*

This project used local LAN network to accomplish using multiple nodes to do the clustering. The Data set is fed to the master node. The data set is along with the Job is distributed among the nodes in the network. The Map function is called for the input in form of $< key, value >$ pair. Once the assignment is complete, the Reduce function recalculates the centroid and makes the data set ready for the subsequent iterations.

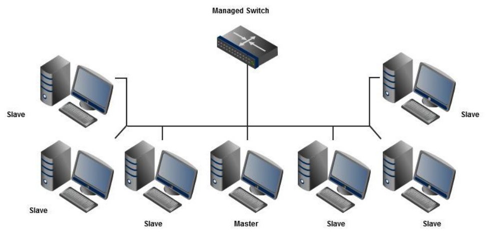
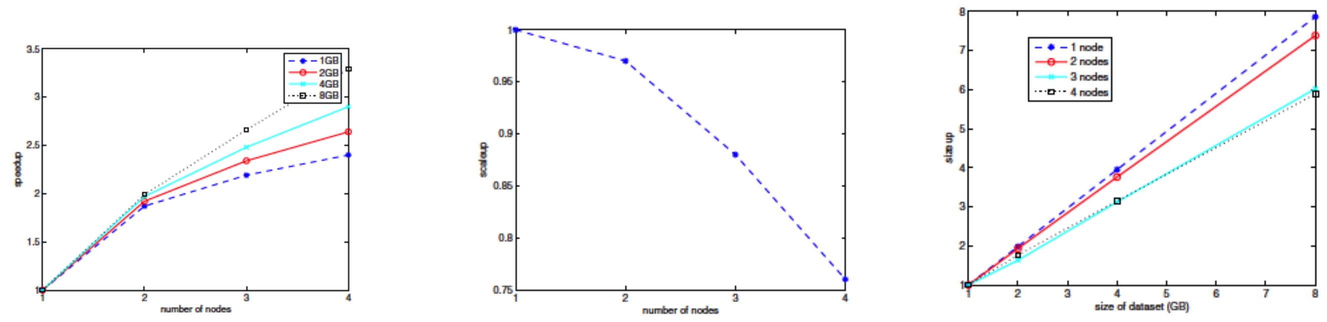


Fig. 2: Experimental Topology for [2]

C. *Implementation Example of k-means on MapReduce [?]*

Each map task read in a shared file containing all the cluster centroid. The cluster centroids file is stored in Hadoop distributed cache. Then each map task read in portion of input data poitns. For each data point, it assign it to cluster where centroid is closest to the current data. Optimization they did was to create a set of k centroids that store the new coordinates. so data points in the same map task are accumulated in the new cluster centroids locally. In result, it reduces the total number of output $< key, val >$ pairs of the map task.



(a) Speedup (b) Scaleup (c) Sizeup

Fig. 1: Evaluation Results from [1]

III. Detailed description of techniques

The mission of K-Means clustering is to partition input N observation data into K clusters, so each observation data belongs to the cluster with the nearest center. The widely known *k-means algorithm* adopts an iterative refinement technique, which is also refered to as *Lloyds algorithm* [4].

A. Brief Description of K-means Algorithm

In the following, we briefly introduce main steps in *k-means algorithm*. Given a data set and the number of clusters k, the algorithm should first give some initial guess about k cluster centers. This step is named as an *initialization step*. Randomly choosing k observations and using them as cluster centers is one of the popular way [5]. After the initialization step, the initialized k centers $m_1^{(1)}, m_2^{(1)}, \cdots, m_k^{(1)}$, where (1) denotes the initialized centers to feed into the algorithm for the first time. Then all observation data are also fed into the algorithm, which proceeds by alternating between two steps:
Assignment Step: Assign each observation data to the cluster with the closest mean. (t) denotes that the set of classified results is generated in the tth iteration.

$$S_i^{(t)} = \{x_j : \left\| x_j - m_i^{(t)} \right\| \leq \left\| x_j - m_n^{(t)} \right\| \text{ for all } n = 1, \cdots, k, \forall j\} \tag{1}$$

Update Step: Calculate the new means to be the centroid of the observations in the cluster.

$$m_i^{(t+1)} = \frac{1}{\left| S_i^{(t)} \right|} \sum_{x_j \in S_i^{(t)}} x_j \tag{2}$$

Then, those two steps will iterate as t increase until convergence.

B. MapReduce Implementation of K-means Clustering Algorithm

Initialization Step: a driver class is written to initialize k cluster centers and is run on master node on HDFS. Instead of reading through the data files and randomly choosing clusters, we simply let the master node pick up the first k observation data as initialized cluster centers. The selected cluster centers are passed as global "String" variable through Hadoop distributed cache. According to the algorithm described above about k-means, we know that the most time consuming computations are within the assignment step and the update step where observations are classified to centers and then classified data vectors are merged into new centers.

Assignment Step: Considering the assignment step, it reads each data from observations, computes distance of the data to k cluster centers generated from last iteration, and then remember the cluster id for the data. It can be adapted to map function as follows Algorithm 1.

Algorithm 1: Mapper of Assignment Step

```
Data: Observation data,
      $\mathbf{x}_i = [\ x_{i,1} \quad x_{i,2} \quad \cdots \quad x_{i,d}\ ]$
Result: Intermediate $< key, value >$ pair, where the
        key is the index of the closest cluster center
        and the value is a string form of observation
        data, written in the form of
        "$x_{i,1} \quad x_{i,2} \quad x_{i,3} \quad \cdots x_{i,d}$"
Load data observation $x_{i,1} \quad x_{i,2} \quad x_{i,3} \quad \cdots x_{i,d}$;
Load cluster centers $\{m_i^{(t)}, 1 \leq i \leq k\}$;
Initialize:
  $dist \leftarrow Double.MAX\_VALUE$ ;
  $id \leftarrow 0$ ;
forall the $i = 1 : k$ do
    $d = euclidean(\mathbf{x}_i, m_i^{(t)})$;
    if $d < dist$ then
        $id \leftarrow i$ ;
        $dist \leftarrow d$ ;
    end
end
Write key value pairs out as $< id, \mathbf{x}_i >$ ;
```

Update Step: After map task, the MapReduce libraries will sort all intermediate pairs based on id that represents the index of the cluster centers. All the data belonging to the same cluster will have the same key id. Then MapReduce are merging those data values to a list that is iterative, which simplifies the updating steps. The reducer just needs to get the mean of all data in a list of values with key id and the mean would be the new center for that id. The reduce algorithm can be shown as Algorithm 2.

```
Algorithm 2: Reducer of Update Step
Data: Classified data
$< i, \{\mathbf{x}_j : \|\mathbf{x}_j - m_i^{(t)}\| \leq \|\mathbf{x}_j - m_n^{(t)}\|\} >$
Result: $< i, m_i^{(t+1)} >$ pair, where the key is the index
        of the cluster center, and the value is the string
        contain of new cluster center.
Load list of classified data
$S_i^{(t)} = \{x_j : \left\|x_j - m_i^{(t)}\right\| \leq \left\|x_j - m_n^{(t)}\right\|\}$;
Initialize:
    $count \leftarrow 0$ ;
    $sum \leftarrow [\ 0 \quad 0 \quad \cdots \quad 0\ ]$ ;
while $S_i$ has next item $\boldsymbol{x}_j$ do
    $sum \leftarrow sum + \mathbf{x}_j$;
    $count++$;
end
Get the mean: $mean \leftarrow \frac{mean}{count}$ ;
Write key value pairs out as $< i, mean >$;
```

Terminating Criteria Checking: Above Map and Reduce functions will run assignment and update step only once, however, k-means algorithm needs to run above steps iterately until the algorithm converge. In our implementation, we regard one iteration of assignment and update steps as one job with above mapper and reducer. After running the job, the driver class on master node reads updated centers from reducer output and compares it with the original centers fed into the job. If there is no difference between the old centers and updated centers, the program finishes. Otherwise, the program would create a new MapReduce job and read updated centers as input for the new job.

IV. Performance Metrics

In previous works, we compare the "fast" cost according to the **Speedup** that accounts how fast a piece of code can speed up after some changes. However, in the MapReduce k-means, though the $(N-1)x$ speedup was expected due to the parallelism of computations, the MapReduce k-means running on multiple instances actually takes longer time than the same code running on only 1 instance. That reminds us the costs of splitting data into multiple nodes and collecting results from multiple nodes. Therefore, we are wondering to what scale, the parallelism of MapReduce k-means is able to take advantage of parallelism to make up these costs.

A. Scaleup

Since we are asking a question of whether the system performance gets improved faster than the speed that system gets bigger, we scale up the system as well as scaling up the testing dataset. We define a dataset running on a single node as a unit dataset. Then, we test M times bigger dataset on M times bigger system (with M nodes). The run time of unit dataset on a single node system is denoted as unit job time. Then the unit job time over the real run time of M size dataset on M node system is defined as **Scaleup** according to [1]. **Scaleup** describes the percentage of M sized data being processed on M scaled system within the unit job time. Namely, it tells us whether the performance would grow faster than system grows up if both of them grow up. If **Scaleup** value should increase as the system scales up, it shows that more speed up can be obtained via increasing the system scale. It can be computed as unit job run time divided by the run time of M times larger dataset running on M times bigger system.

B. Sizeup

For describing the performance of parallelism, [1] also used a metric called **Sizeup** to describe how much more time a M times larger dataset will take given a certain size of system. In our later experiments in Section V-C, we used above three metrics to characterize the performance of our k-means implementation on MapReduce.

V. Analysis of Results

A. Implementation Overhead

In order to parallelize the *k-means algorithm*, we adapt one iteration of assignment and update steps to one MapReduce job with above Mapper and Reducer in Algorithm 1 and Algorithm 2. Each job creation introduce overhead on communications. In addition, in initialization step and terminating criteria checking, the master node needs to read center files from Hadoop file system, so there are more overhead on communications and file operations. The theoretical speedup of parallelization of k-means algorithm should be $(N-1)x$ where N is the total number of nodes. However, due to the overhead, the speedup of real testing should be lower than that.

B. Initial Testing

We conduct initial testing of the implementation on some small datasets with following sizes in Table I (Data are obtained from previous mini-projects). These testing are just for validating the correctness of our implementation. There would be no speedup to run these datasets on multiple nodes because MapReduce split data into multiple nodes with 64MB size and all following datasets are smaller than 64MB. Larger dataset will be generated and speedup testing will be conducted in the future. We then create 4 medium instances on Amazon EMR and run above MapReduce k-means implementation on those 4 datasets. The total running time and total number of jobs to converge are counted as Table II. The total running time on 4 instances are also compared with the running time on 1 instance for these datasets.

TABLE I: Initial Testing Dataset

Dataset	Number of Data	Dimension	Size	k
kmeans01.dat	351	34	112KB	3
kmeans02.dat	7089	4	320KB	3
kmeans03.dat	191681	22	40.2MB	3
kmeans04.dat	488565	8	40.6MB	3

TABLE II: Initial Testing Results

	kmeans01.dat	kmeans02.dat	kmeans03.dat	kmeans04.dat
Iterations	7	8	5	19
Time on 1 ins.	16.744 sec	20.276 sec	52.053 sec	3.22 minutes
Time on 4 ins.	4.19 minutes	4.92 minutes	4 minutes	15 minutes

It can be seen that due to the overhead introduced in creating jobs, reading and writing data into Amazon S3 storage, the running time of multiple instances is unexpectedly larger than the running time on one instance. One reason as we discussed above, might be the data size is too small to run on clusters. In the following, we will focus on the speedup of the code and answer the question about up to what scale of dataset, the mapreduce version of k-means is able to parallelize the computation well enough to make a speedup to compensate the overhead of mapreduce.

C. Testing for Varied Scales

In order to test how MapReduce k-means can reduce computation time by taking advantage of parallelism using the same algorithm and same implementation, we generate various sizes of datasets with dimension $D = 10$ and number of clusters $k = 3$ and test MapReduce k-means clustering with different number of nodes. Three cluster centers are generated by random vectors. Testing data sets are generated via Random Gaussian distribution with three cluster centers. The sizes of datasets are shown in following Table III.

TABLE III: Scaleup Testing Dataset

Dataset	Number of Data	Dataset Size	Dimension	k
kmeans100k.dat	100K	9.1 MB	10	3
kmeans1M.dat	1 million	92.4 MB	10	3
kmeans10M.dat	10 millions	933.5 MB	10	3
kmeans20M.dat	20 millions	1.8 GB	10	3
kmeans40M.dat	40 millions	3.6 GB	10	3
kmeans50M.dat	50 millions	3.7 GB	10	3
kmeans80M.dat	80 millions	5.9 GB	10	3

In order to test how system performance gets improved as system scales up, we picked up 1, 2, 4, 5, 8 AWS c1.medium instances as varied scales of system to test our implementation. Similarly as [1], we draw performance metrics of **Speedup**, **Scaleup** and **Sizeup** as follows.

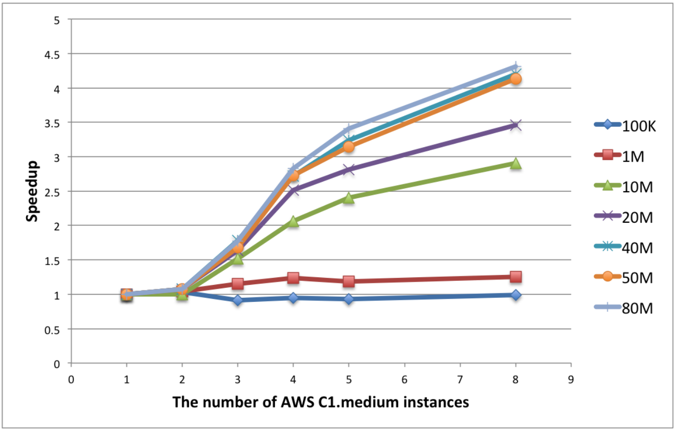


Fig. 3: Speedup

In the **Speedup** Figure 3, it can be seen that there are only speedup for dataset with sizes over 10 millions data. The file size of 10 million data is nearly 1GB. We have also shown that in initial testing, there is no speedup for dataset with lower than 100 MB size. In addition, Figure 3 also shows that the speedup increases faster with the system scale for larger dataset. However, the slopes are all below 1 which means there are some costs introduced to parallel the *k-means algorithm* by MapReduce.

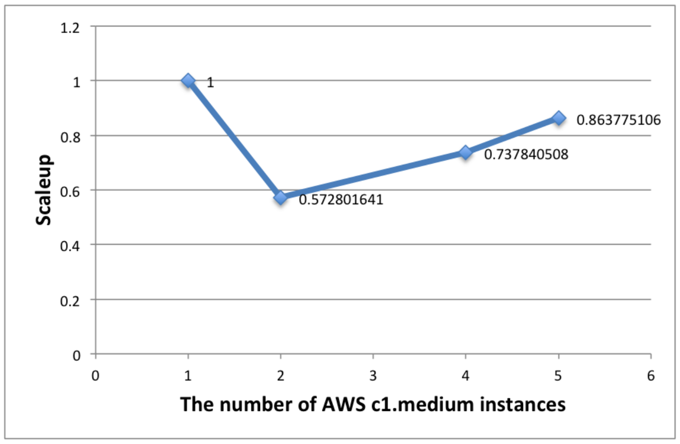


Fig. 4: Scaleup

Scaleup in Figure 4 describes the ability of our code to process M larger dataset on M scaled system within the unit job time. The figure shows an increasing trend on the curve as the system scales up, which shows that as the system scales the performance gets better. It also indicates that the impact of MapReduce cost on system performance decreases as system scales up.

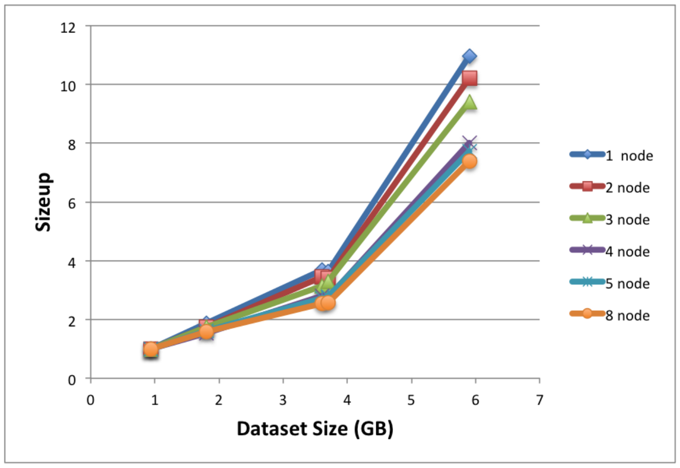


Fig. 5: Sizeup

Figure 5 shows as the size of dataset increases, the time to process larger datasets increases accordingly. However, larger scaled system can lower down increasing speed of running time. Namely, it shows that for large dataset, large scaled system can save time over sequentially processing data on small scaled system. For example, for 8 node system, the processing time for 80 million data is lower than 8 times of the processing time for 10 million data on single node system.

VI. Conclusion

In this project, we made a practice on using MapReduce through implementing k-means clustering algorithm. By testing our code on various datasets on varied scales of system, we proved that MapReduce can speedup k-means algorithm via parallelization. We also showed that the parallelization of MapReduce is only effective for large datasets. Our experiments only had apparent speedups for datasets over 1GB size. Additionally, we also found out that larger scaled system took better advantage of parallelization by MapReduce and the impact of MapReduce overhead on performance becomes smaller as system scales up. Results also demonstrated that the reduction of running time performs better for larger scaled system and larger dataset.

Acknowledgment

The authors would like to thank Professor Jike Chong and Professor Ian Lane for their lectures and guidances.

References

[1] W. Zhao, H. Ma, and Q. He, "Parallel k-means clustering based on mapreduce," in *Cloud Computing*. Springer, 2009, pp. 674–679.

[2] P. P. Anchalia, A. K. Koundinya, and S. NK, "Mapreduce design of k-means clustering algorithm," in *Information Science and Applications (ICISA), 2013 International Conference on*. IEEE, 2013, pp. 1–5.

[3] Y. Zhang. (2014) Mapreduce kmeans algorithm implementation. [Online]. Available: http://yunmingzhang.wordpress.com/2014/01/13/mapreduce-kmeans-algorithm-implementation/

[4] S. Lloyd, "Least squares quantization in pcm," *Information Theory, IEEE Transactions on*, vol. 28, no. 2, pp. 129–137, 1982.

[5] G. Hamerly and C. Elkan, "Alternatives to the k-means algorithm that find better clusterings," in *Proceedings of the eleventh international conference on Information and knowledge management*. ACM, 2002, pp. 600–607.