Big Data Analysis project report

Parallel K -Means Clustering Based on MapReduce

Weizhong Zhao, Huifang Ma, and Qing He

Group Members

MBOUOPDA Michael Franklin	ICS
Jiariu XIE	INT
NTONGTONG TCHANI Larissa	GLIA
DOMKUE NGAKO Franck Arnold	SIAD

Introduction

Describe the Problem

With the development of information technology, volume of data processed by many applications will routinely cross the petascale threshold, which would, in turn, increase the computational requirements. Efficient parallel clustering algorithms and implementation techniques are the keys to meet the scalability and performance requirements entailed in such scientific data analyses.

Our Contributions

In this report, we did contributions like the following:

- Review the conceptions of K-means and MapReduce
- Give our codes[3] according to the pseudo-code provided by the PK-Means algorithm[1]
- Execute the codes over 1 master and 4 slaves distributing environment with a cars dataset of 16,185 images.
- Verify the results got from experiments meanwhile draw the conclusions

Outlines

The rest of the paper is organized as follows:

- Section 2 demonstrates the basic conceptions of K-means and MapReduce
- Section 3 gives specific experiments, contains:
    - Dataset
    - Preprocessing
    - Hadoop setup
    - Execution and results
- Finally, we offer our conclusions in Section 4.

Parallel K-means with MapReduce

Basic K-means

K-means is an unsupervised algorithm for non-hierarchical clustering. It makes it possible to group a dataset in K distinct clusters such that data in the same cluster are most similar to each other while data in different clusters are very distinct. In the basic form of k-means k random points are selected as the centers of the cluster; then the algorithm iteratively assign each point to its closest centroid (a cluster's center) and then each centroid is recomputed as the mean of all data assign to it. The process continues until the centroids do not change anymore.

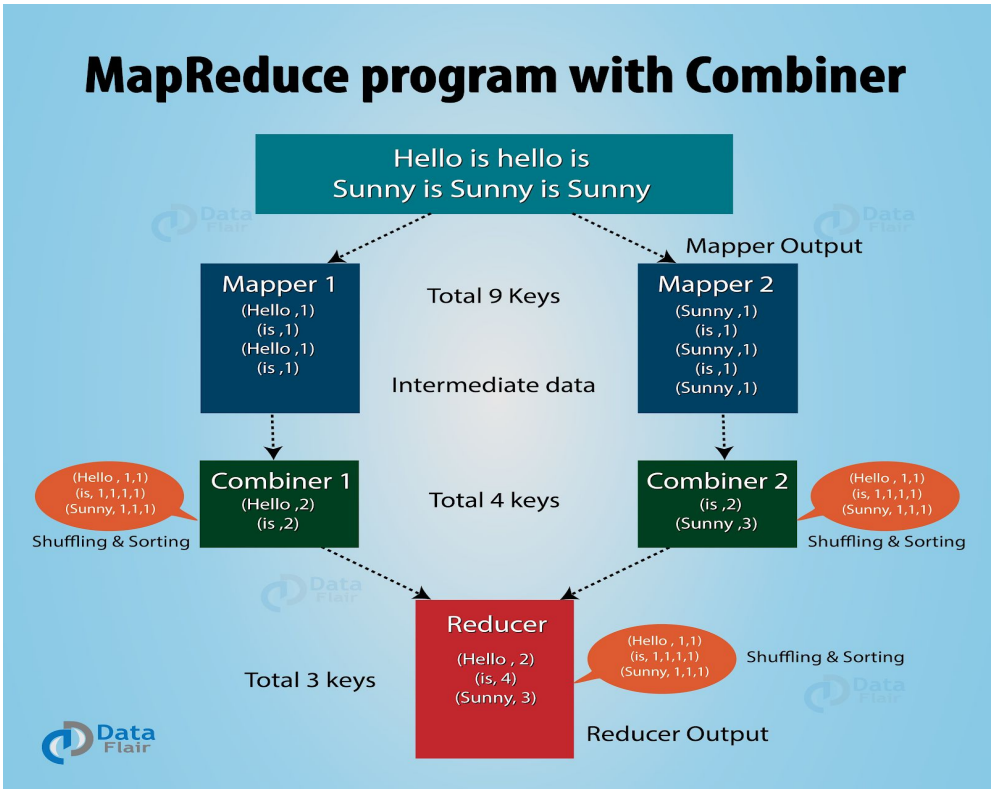
```
Algorithm: KMeans
Input
  K the number of clusters to form
  The set of data
Begin
   Randomly choose K centroids.
   REPEAT
      For K cluster by assigning each point to its closest centroid
      Recompute the centroid for each cluster
   UNTIL CONVERGENCE
End
```

In k-means algorithm, the most intensive computation that occurs is the calculation of similarity between data and centroids. In each iteration, it would require a total of $(n \times k)$ similarity computations where n is the number of objects and k is the number of clusters being created. Doing this computation on a big dataset will require a lot of time before the convergence of the algorithm.

Since the computation of the similarity between a data point and a centroid do not required data from any other computation, parallelization can be used to reduce the computation time.

Parallel K-means

Hadoop is a framework that allows for the distributed processing of large data sets across clusters of commodity computers using a simple programming models called MapReduce. A MapReduce program is designed by defining the Mapper function and the Reducer function. A Combiner (also called a Mini Reducer) function can be used to reduce the volume of data going from the Mapper to the Reducer and by the way reduce the communication cost on the network. Hadoop will partition the dataset and each partition is going to be processed by an instance of the mapper function. The output of each mapper is going to be processed by the combiner. The reducer will receive the output of all combiner to do the last computation. The following is the flow of a MapReduce program that counts the number of occurrences of each word in the input data.



The paper we had to read gives a design of a parallel KMeans using MapReduce: it is called PKMeans. Each mapper have access to a global variable with is an array that contains the k centroids. At the beginning, PKMeans randomly initialized this global variable.

Mapper function

The job of the map function is to assign each point to its closest centroid.

> **Function Map**
> Input: Global centroids, <key, datapoint>
> Output: <key', datapoint> where *key'* is the index of the closest centroid

Combiner function

The combiner will output the sum and the number of data points assigned to same cluster by the mapper.

> **Function Combine**
> Input: <key, D>, where *key* is the cluster index and *D* the list of data assigned to that cluster
> Output: <key, <SUM, N>>, where *SUM* is the sum of data in *D* and *N* the length of *D*

Reducer function

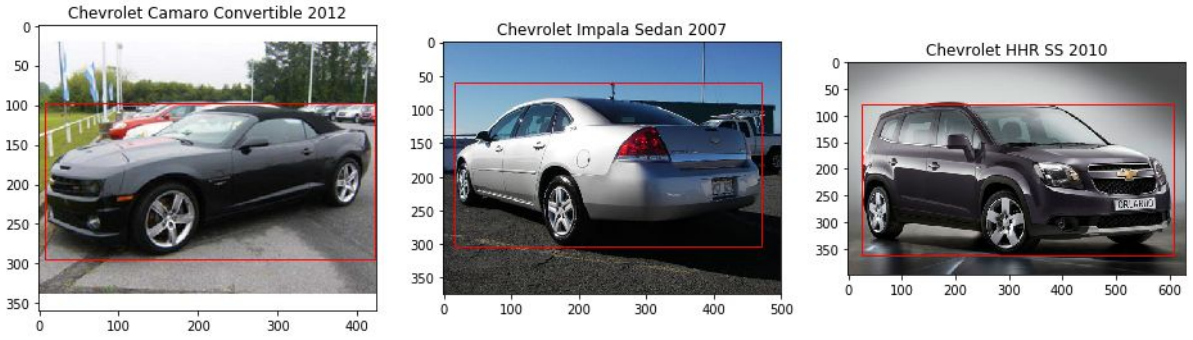
The role of the reducer is to compute the new centroid using the output from the combiner.

> **Function Reduce**
> Input: <key, L>, where key is the cluster index and L is the list of <SUM, N> produced by the combiner
> output: <key, centroid>, where centroid is the new coordinate of the k-th centroid.

Experiments

Dataset

The dataset we used for the experiments is the Stanford Car Dataset by classes available at the link https://www.kaggle.com/jutrera/stanford-car-dataset-by-classes-folder. The Cars dataset contains 16,185 images of 196 classes of cars. The data is split into 8,144 training images and 8,041 testing images, where each class has been split roughly in a 50-50 split. The whole data set is contained in a 2Gb zip file. Each file is an image that contains a car and the dataset provides the bounds of the car in the each image. Below are three samples from the dataset. We added red rectangle on each image to show the bound of the car in the image.

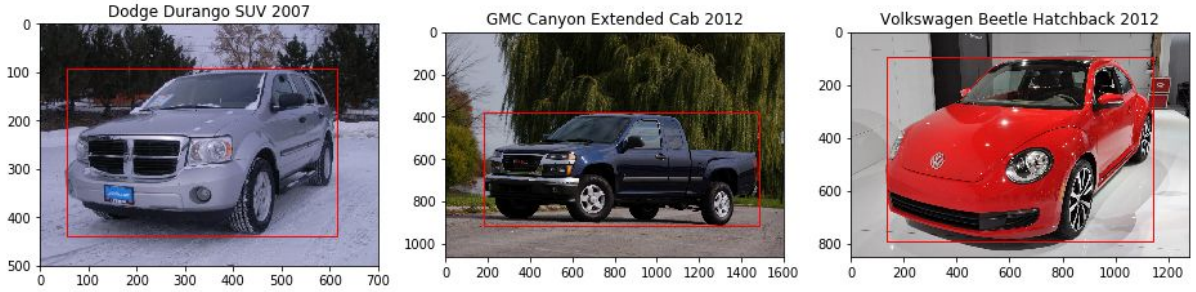


Preprocessing

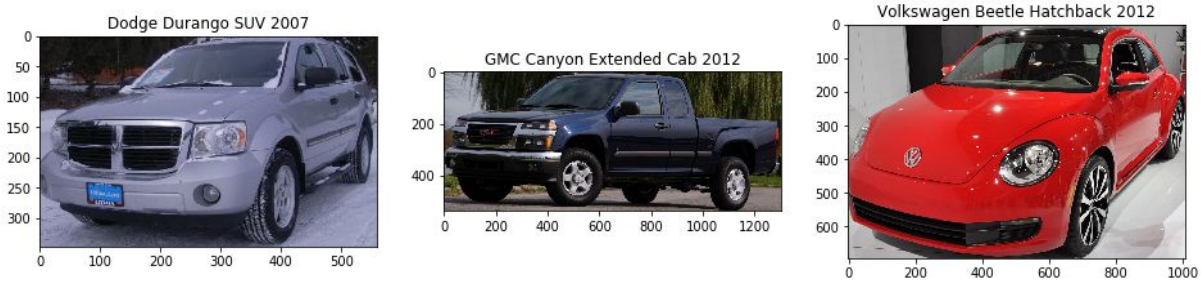
To use an image as input to PKMeans, there are some preprocessing to do:

- firstly the bounds of the the car are use the extract only the car from the image
- the extracted image is then resize to 100x100 to keep all image in the same scale. We have tried many size and choose this one because the quality of the obtained image is not too bad.
- then the image is converted in a grayscale vector in which each component is divided by 255 for normalization.

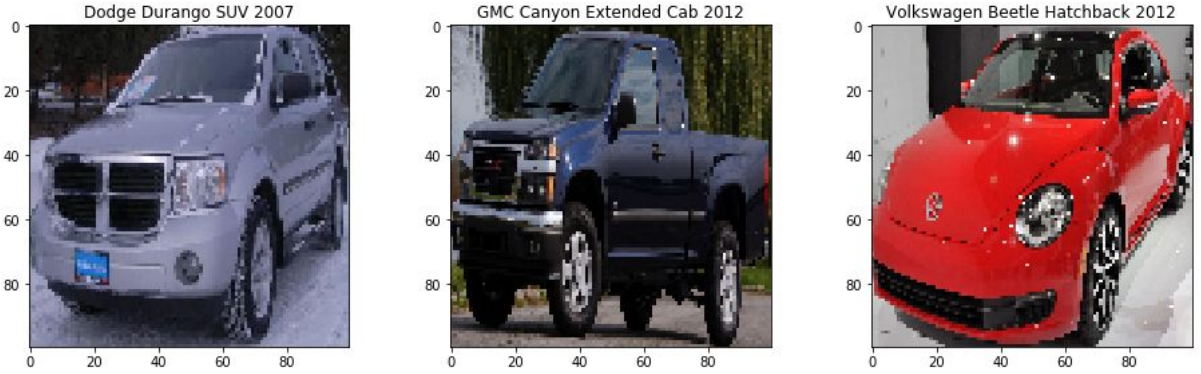
Below are some samples and their corresponding output after the preprocessing



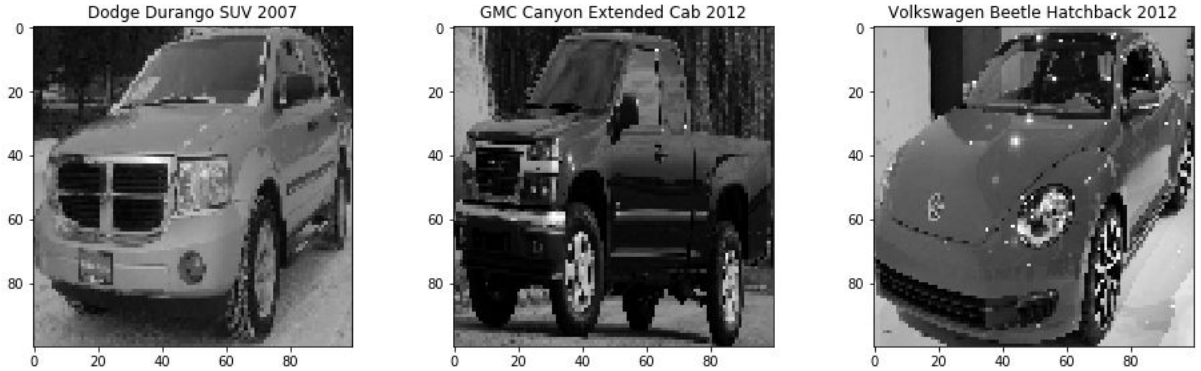
Original images



Extracted cars



Resized images



Grayscale images

After the preprocessing of the whole dataset we get a training dataset of 645 Mb and a testing dataset of 630 Mb. Each line in the training/testing dataset is the grayscale vector of a car image.

We did the preprocessing in Python and the source is available here : https://github.com/frankl1/PKMeans/blob/master/Proprocess.ipynb

Hadoop setup

We set up Apache hadoop 2.9.2 on a local computer and we used java 1.8. We run hadoop in pseudo-distributed mode and in fully distributed mode. The fully distributed mode have one master and 4 slaves. Because of the lack of computers, the fully distributed mode have been set using Docker containers.

Execution and results

We have run the code in pseudo-distributed mode on a dataset of 24K to make sure it's working. In average, PKMeans converge after eighteen iterations on this dataset. We have then run the code in fully distributed mode using the Stanford Car Dataset but got an Out Of Memory error before the convergence of the algorithm.

Our implementation is available at https://github.com/frankl1/PKMeans

Conclusion

The PK-Means algorithm is a nice alternative to deal with challenging task risen by clustering of the very large scale of data. In this paper, we have implemented the parallel k-means clustering algorithm[1] based on MapReduce, which has been widely embraced by both academia and industry.

We set a distributed Hadoop on one computer using Docker. But due to the limited storage of RAM, the result is not so reasonable as expected. Roughly, the speedup, scaleup and size up respected to the performances proposed by the reference paper[1].

In the future, we will plan to use an authentic cluster to verify our codes in real distributed environment.

References

[1] Zhao W., Ma H., He Q. (2009) Parallel *K*-Means Clustering Based on MapReduce. In: Jaatun M.G., Zhao G., Rong C. (eds) Cloud Computing. CloudCom 2009. Lecture Notes in Computer Science, vol 5931. Springer, Berlin, Heidelberg

[2] Dataset: https://www.kaggle.com/jutrera/stanford-car-dataset-by-classes-folder

[3] Source code: https://github.com/frankl1/PKMeans