Mining of Massive Dataset

Hao Jiteng, Zhou Lizhi, Yang Fangzhou

June 19, 2012

Abstract

K-means is a simple yet useful clustering algorithm. It's underline natural implies that it could be parallelized or distributized. Some implementation of kmeans employs platforms such as Hadoop and CUDA to boost the process of mining of massive dataset. In our project we implement k-means algorithm on Apache Hadoop Project. We ran our algorithm on our tiny cluster. Evaluation has been done to measure our algorithm.

1 Introduction

1.1 Hadoop

Following is the introduction of Hadoop on its project homepage [?].

The Apache Hadoop project develops open-source software for reliable, scalable, distributed computing.

The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using a simple programming model. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. Rather than rely on hardware to deliver high-avaiability, the library itself is designed to detect and handle failures at the application layer, so delivering a highly-available service on top of a cluster of computers, each of which may be prone to failures.

We mainly use the Hadoop MapReduce framework rather than HDFS.

1.2 k-means

There are many materials that introduce k-means algorithms. MacQueen, J. et al. proposed k-means algorithm in [?]. In [?] the author introduced a simple k-means MapReduce algorithm.

The dataset is relatively large comparing to the memory size. Thus we need a mechanism to deal with the incompatibility of memory. MapReduce is a solution to this problem. The detail of our algorithm is documented in the following section.

1.3 Dataset

2 Clustering Algorithm

In this section we discuss the k-means clustering algorithm used in our implementation. Note that to find proper initial clusters, which may lead to fewer iterations and good result, we also implemented a Canopy clustering MapReduce algorithm.

2.1 Canopy Clustering

Canopy method is a relative easy but efficient clustering method. The basic idea of Canopy is to put the similar items into the subset, which are called Canopies. Different Canopies can overlap each other, but there is no node, which doesn't belongs to any of Canopy. Thus, here we use this method to pretreat the dataset before Kmeans method.

The algorithm is built up by the following MapReduce phases,

1. Choose two distance threshold $t1$ and $t2$, where $t1 > t2$.
2. Choose a node P from the list, and quickly calculate the distance between node P and Canopy, if this distance is less than t1, put P into this Canopy, if no such kind Canopy exits, take P as a new Canopy.
3. if P was near enough to some Canopy, actually it means the distance between P and the Canopy is less than t2, then delete node P from the list.
4. repeat 2 and 3 until the list is empty.

The parallel strategy described as map-reduce is to produce some Canopy in the mapper work and get the final Canopy set in the reducer work. The more details are described as follows

1. Mapper: `<LongWritable, Text>`$\to$ `<1,CanopyCluster>`. It reads the input file content as value. The key is a fixed string "centroid", and the value is the center node of the Canopy.
2. Reducer: there is only one canopy reducer in the canopy clustering procedure and it should gather the output of all the mappers and do the second round of canopy clustering for the result of mappers (and it's possible to use a different $t1, t2$ combination). Then the reducer write `<identifier,KmeansCluster>` into context. It's handy to use the KmeansCluster as the result format and kmeans clustering pass could load the result quickly.

2.2 k-means Clustering

Mahout Project [?] is a data mining framework under Apache Fundation. It contains a k-means implementation. The blog [?] gives a very detailed view of the algorithm. Our algorithm mainly based on the idea of Mahout Project. Actually, this algorithm is very similar to the BFR algorithm in our lectures.

The algorithm is built up by the following MapReduce phases,

1. The k-means iteration, output every cluster centroid if converged.
2. Assign every point to a known cluster and output the result.

The detail of the first phase is described as follows

2.2.1 Iterations

1. Mapper
   `<LongWritable, KmeansCluster>`→ `<LongWritable,KmeansCluster>`. It reads the input file content as value. The key is the value offset in the file. Then it convert the value to a VectorDoubleWritable, which is used to represent the feature vector. It finds the nearest cluster to the vector, and output cluster id as key, a new cluster containing only point as value.

2. Combiner
   `<LongWritable, KmeansCluster>`→ `<LongWritable, KmeansCluster>`. It reads the output from Mapper and combine those tuples who have the same cluster id(meaning that they are assigned to the same cluster) using KmeansCluster.omitCluster(). This function reduced the network transmission flow because the actual meaningful information need to be communicated between different nodes are only the N, SUM and SUMSQ of clusters, which is described in the lecture slides of BFR algorithm.

3. Reducer
   `<LongWritable, KmeansCluster>`→ `<LongWritable, KmeansCluster>`. It reads the cluster id as key and the KmeansCluster as value. It adds the N, SUM and SUMSQ. The result leads to the combination of clusters. Finally the reducer outputs the result clusters of this single iteration. This clusters are input of next iteration. During reducer, if the movement of one cluster if less than a threshold, it's said to be "Converged". If a cluster is converged, a counter in context will increase by one.

4. If, in the driver, the counter equals the number of clusters, meaning that all clusters are converged, this phase is finished.

2.2.2 Assign Point to Clusters

After the iterations, the clusters are stable. Next we are going to assign every point to its nearest cluster. This phase only require mapper to do all the works, since this procedure is highly parallelized. Each two point are independent of each other.

3 Evaluation

In our experiment, we calculate the purity to evaluate the results. A higher purity means a more accurate clustering result.

3.1 Evaluation Algorithm

Because we don't know the map relationship between our clusters and the GrandTruth classes, so we must find a maximum match from the clusters to origin classes, so that we can get a maximum purity. The basic idea to find such a maximum match is to find a highest purity of the clusters for each class, and map the cluster to the origin class. The detail of the Algorithm is described as follows

1. For each origin class, find the cluster-ID, which has the maximum purity.
2. Find the class which has the maximum purity among the maximum purity of different classes. And assign the maximum cluster-ID to this class.
3. Set all current purity in this class to zero, and set all the cluster-ID in other classes to zero.
4. Repeat 1,2,3 until all classes have an assigned cluster-ID.
5. Output the purity for each classes.

3.2 Evaluation Results