1. **The Difference between K-means and K-means++**

Actually, the process of assigning data points to K centroids and updating centroids based on new data points in the cluster is same. Firstly, the distance between all the data points and all the K-centroids will be calculated and all the data points will be assigned to the centroid will smallest distance. Then the centroid will be updated based on the average value on all dimensions of the data points in its cluster. And the process will be executed again and again until the centroid do not change any more.

However, the method of selecting K different initial centroid is different. For K-means, the K initial centroids are selected randomly from all the data points in the dataset. But as for K-means++, its philosophy of selecting next centroid is to select the one from those data points that are relatively far from the existed centroids.

Specifically, it will randomly select the first centroid, then calculate the distance of all the data points from the centroid. After that, next centroid will be selected based on probability and the one with high distance will have higher possibility to be selected. Finally, K centroids will be selected.

1. **Programming Design**

In order to implement the experiment, Firstly I should implement the parallel k-means algorithm.

Specifically, I designed the map function which take the k-centroids list and the RDD read from file as input. Then another function will be called to calculate distance between every centroid in the centroid list and the instance, then the index of the centroid with closest distance will be selected and returned as the key of the instance.

Then a combiner function is designed to combine the RDD with same key (same centroid index). Specifically, the combiner will create a list for each centroid index and keep summing each dimension into the list for all the points assigned to the centroid. Additionally, the combiner function will put another value after the RDD instance to log the number of points assigned to each centroid.

Then the reduce method will update each centroid in the centroid list by firstly summing up all the value together for each key, then divide each dimension by the number of points which is the second element in the value of each RDD.

Then this process will be put into a while loop until the converge or the change between the iteration and last iteration reach to level which is acceptable. The objective of this experiment is to see how much efficiency the parallel k-means bring. So firstly, I created google Cloud clusters with 1, 2, 3, 4 worker nodes separately and split the data into 1/8, 1/4, 1/2 and whole data set. Then run the Parallel k-means for datasets in different size and compare the time used for running different datasets in different number of nodes to check the speedup, sizeup and scaleup.

Here is the screenshots for the clusters with different number nodes ‘running results:

Figure1. Result for data running on 1 node

图片包含 屏幕截图

描述已自动生成

Figure2. Result for data running on 2 nodes

图片包含 屏幕截图

描述已自动生成

Figure3. Result for data running on 3 nodes

图片包含 屏幕截图

描述已自动生成

Figure4. Result for data running on 4 nodes

图片包含 屏幕截图

描述已自动生成

Figure5



Figure6



Figure7



According to the general trend of 3 line charts, we could see that although the efficiency increase from 3 nodes to 4 nodes is not that obvious, parallel k-means do increase the efficiency especially when the number of nodes got increased and data size got increased.