HARP KMEANS

Project Report – Harp Kmeans

Indiana University

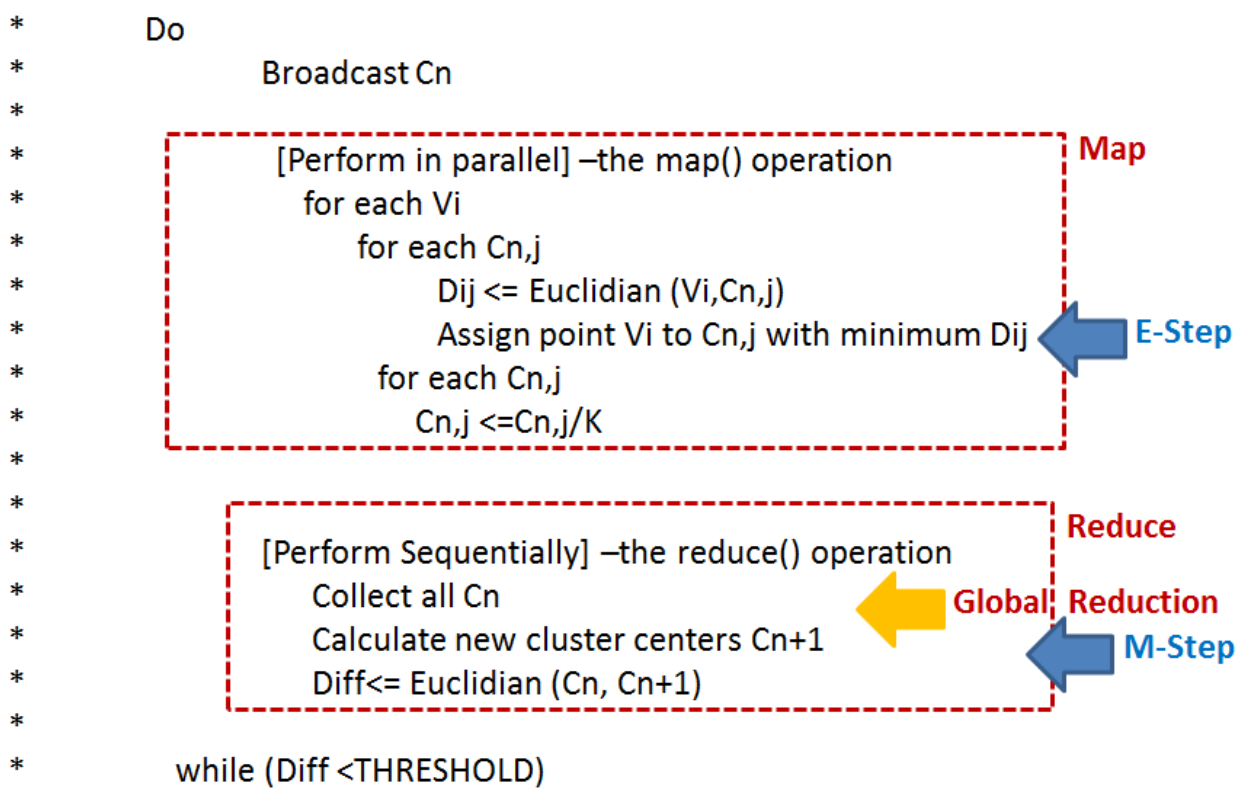
Revision	Date	Description	Author
1.0	20-Apr-2017	Coding	Karthik Vegi
1.1	21-Apr-2017	Testing and report	Melita Dsouza

HARP KMEANS

PROJECT DESCRIPTION

In this project, we will implement Harp Kmeans algorithm

The Kmeans algorithm works as follows:



Vi refers to the ith vector
Cn,j refers to the jth cluster center in nth * iteration
Dij refers to the Euclidian distance between ith vector and jth * cluster center
K is the number of cluster centers

Data Structure and Logic

- The main data structure used for this assignment is ArrTable<DoubleArray> which is a collection of double array objects. Each double array represents a center.
- For example, if the centers for your program are of n dimensions, it will have (n+1) elements.
- In a 3 dimensional space he first 3 will be the x, y, z coordinates. The last element is used to store the number of points assigned to this center.
- To retrieve all the centers you can invoke the **getPartitions()** method on ArrTable object, which will return a collection of ArrPartition objects
- To retrieve the underlying double array from these ArrPartition objects, you can invoke the **getArray()** method on ArrPartition object
- To figure out the index (ID) of this center you can invoke **getPartitionID()** method on the same ArrPartition object.

HARP KMEANS

Flow of the program: main components

- **KMeansMapCollective**
  - launch the kmeans launch code which will set the configuration, check for the arguments and set the input and output directories
  - **runmbkMeans** with the required parameters. Configure the job and report error if the job has failed
  - **configureMBKMeansJob:** this will configure the job by setting the file system, setting the job configuration constants, setting the input format, set the jar by class
- **KMeansConstants**
  - here we set all the constants we use in the program
- **KMeansMapper**
  - We get the same data points by generating them randomly
  - We loop over the defined iterations and get points
  - We find the nearest centroid
  - We modify the centroid based on the points

nearestCentroid:

- We find the nearest centroid based on the centroids and data points
- For each partition from previous centroid table, we calculate the Euclidean distance between the data points and the centroid
- If this is the minimum distance, we assign the point to centroid

outputCentroids:

We finally write all the centroids to HDFS

Bonus Credits - Perform experiments on various (small, medium, large, etc) datasets

Input 1: 1000 data points

Partial Output1 looks as below…

HARP KMEANS

```
| Job#0 Finished in 95039 miliseconds |
Total MBKmeans Execution Time: 95040
MBKmeans Completed

cc@cc-VirtualBox:~/Documents/hadoop-2.6.0$ hdfs dfs -cat /kmeans/centroids/*

1.3616343714570434 7.617956152158955
0.6108507761810744 0.654379607443788
4.931713504412931 5.289448176765996
9.460129462181978 4.908147924939926a
6.314116674614426 9.572645514173567
0.07790975448274784 6.8073207171773475
7.829742747331228 3.5683498067427744
8.860896762532949 2.572035308245516
9.632093238019584 6.82168925203494
6.983849860725604 6.55263555648113
9.595030714147214 5.88318994057186
2.8437188758898713 3.218301153592289
2.4380911218016807 4.460214935821241
0.07790975448274784 6.8073207171773475
7.829742747331228 3.5683498067427744
6.879750592703203 6.197687552199086
6.8653972662431535 3.061939709656325
1.9678862899151228 9.827391494565788
4.725167834878457 9.225193261193404
5.275542234022234 7.977698354625145
6.1895205911353655 2.447189209984132
7.221633849733368 7.350844613742675
8.860896762532949 2.572035308245516
9.632093238019584 6.82168925203494
6.983849860725604 6.55263555648113
```

HARP KMEANS

Input 2: 10000 data points

Partial Output 2

```
| Job#0 Finished in 137312 miliseconds |
Total MBKmeans Execution Time: 137312
MBKmeans Completed

cc@cc-VirtualBox:~/Documents/hadoop-2.6.0$ hdfs dfs -cat /kmeans/centroids/*

5.060377256762672 2.8971351485650443
3.489690191682524 9.221799290521975
9.712682115785057 7.936607820090822
3.9472050672156564 5.107514316678432
1.9678862899151228 9.827391494565788
4.725167834878457 9.225193261193404
5.275542234022234 7.977698354625145
6.1895205911353655 2.447189209984132
7.221633849733368 7.350844613742675
8.860896762532949 2.572035308245516
9.632093238019584 6.82168925203494
5.882752958901914 8.75515252262722
3.238742969604187 8.190392737171974
8.923426105714828 5.3797692951329275
4.1494969586932475 8.879153702235493
6.9664632131431725 6.966269776975434
5.917378938640034 4.538145398817864
9.460129462181978 4.908147924939926a
6.314116674614426 9.572645514173567
0.07790975448274784 6.8073207171773475
2.9057328859744636 0.48355568488620104
9.80979819036770 2.045971551330438
```

HARP KMEANS

Input 3: 500000 data points

Partial output 3:

```
end Jod#0 19:21:12.732
| Job#0 Finished in 215431 miliseconds |
Total MBKmeans Execution Time: 215431
MBKmeans Completed

cc@cc-VirtualBox:~/Documents/hadoop-2.6.0$ hdfs dfs -cat /kmeans/centroids/*

5.742572690415459 9.375255166459521
8.425493670702254 6.59532736738909
8.858724125777444 5.4739173250414055
4.151908487782382 2.223954260889597
4.632257919130172 8.89460537360328
4.084463497295529 1.8312720158874052
0.9096777643472875 3.3429097778229733
5.788568641916013 7.366847562755524
6.314116674614426 9.572645514173567
0.07790975448274784 6.8073207171773475
8.452223187387876 3.0315099543223214
3.7924647559540214 0.898826743961818
3.7264094276874724 5.186613235481438
0.18830069373720115 8.251169948652585
9.460129462181978 4.908147924939926a
8.113924454334555 3.0180836371816677
5.373466592036618 9.49491505344388
1.4500610236561329 4.595790932389261
.809798190367703 2.045971551330438
```