Mini Batch K-means using Harp:

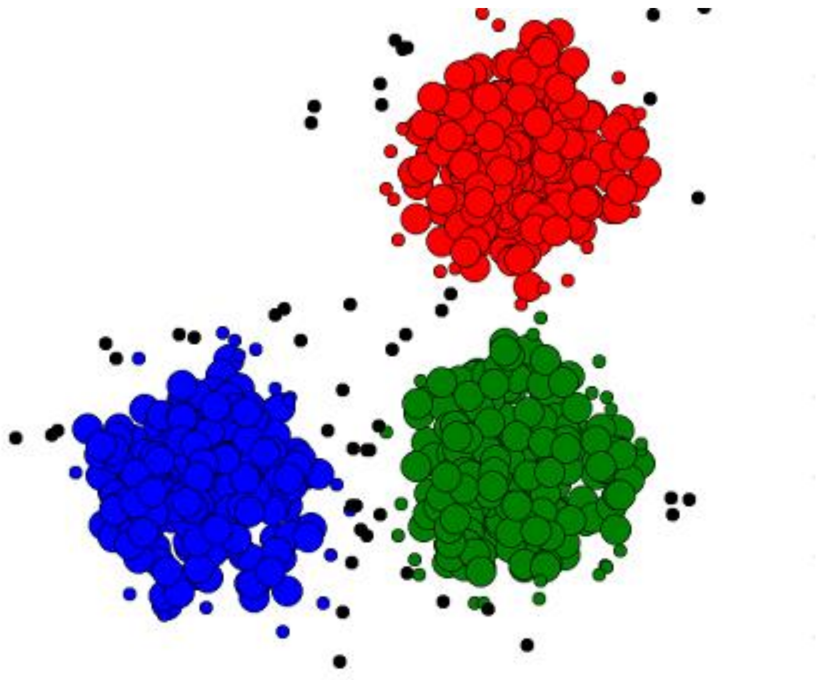

Diagram showing Clusters of data [1]

K-Means: It is one of the most popular clustering algorithm in the world.

Harp: This Hadoop framework tries to implement MPI like features like allreduce and allgather. Harp provides more granular control over network communication for optimizing run-time.

Why are we implementing K-Means in Harp?

Currently, K-means is not optimized for running in parallel, as it requires reduction after every iteration. We have implemented Mini-Batch K-means that reduces this overhead.

Mini-Batch K-Means:

Step 1: Select *c* centroids randomly from the dataset

Step 2: For *t* iterations

Step 3: Sample *m* values from the dataset

Step 4: For each data point, find the nearest centroid and store it in a cache

Step 5: For each centroid, calculate gradient step and update it

Step 6: Call all reduce to synchronize the centroids across all the mappers

Implementation description:

We have implemented Mini Batch K-Means using all-reduce technique. Following is the code description:

1. We randomly create data points and store to HDFS
2. We pick random centroids
3. We randomly pick *batchSize* points from the data points file and pass it to a mapper
4. Mappers find the nearest centroids for all the data points in their batches
5. We cache these centroids along with the per-center counts in the mapper
6. Now, we synchronize the centroids and per-center counts across all the mappers using all reduce
7. Finally after running all the iterations, we store the centroids in the HDFS

How to execute?

Step 1: cd $HARP_ROOT_DIR

Navigate to the Harp root directory

Step 2: mvn clean package

Building the project

Step 3: cd $HARP_ROOT_DIR/harp-tutorial-app

Navigate to our project's directory

Step 4: cp target/harp-tutorial-app-1.0-SNAPSHOT.jar $HADOOP_HOME

Copy the jar to Hadoop's home directory

Step 5: cd $HADOOP_HOME

Navigate to the Hadoop's home directory

Step 6: hadoop jar harp-tutorial-app-1.0-SNAPSHOT.jar edu.iu.kmeans.common.MiniBatchKmeansMapCollective 1000 10 10 100 2 10 /minibatch /tmp/minibatch

Run the jar

Class arguments:

1. Number of data points
2. Number of centroids
3. Dimensions of each data point
4. Number of points in a batch
5. Number of map tasks
6. Number of iterations
7. The root directory: Where the output files will be kept
8. The initial data points will be kept here

Step 7: hadoop fs -copyToLocal /minibatch/centroids/* /home/cc/Documents

Copies the output file to Documents directory

Step 8: vi /home/cc/Documents/centroids_0

Displays the contents of the file

Output:

```
5.901060983984809 5.558106599751342 2.4645361379131394 0.8734011696061406
3.0429180532752187 8.504280885571193 5.437551784275355 1.2095980486960611
5.690455000330448 9.506812325928912

1.8305963054491292 7.6701250583 6.544444404707324 2.603143579499081
7.647223966333016 6.274654452214051 6.430835288873537 9.157685924765804
6.963862658836128 5.668440578037117

7.23605314464474 4.808600319969765 4.472947333421904 8.948882725151622
9.076049204062128 9.36178347406638 2.331368950932152 9.156612736502135
3.416937500415976 7.6609298240590356
```

8.251799539652318 7.031261378151728 1.5701279687817804 3.495742911855506
0.26078191871277023 5.696519318054639 9.752451462399689 9.206071883707343
1.0963022315863147 4.56605478138005

7.7318396664991305 0.7979506973312711 1.8288907483369654 3.9513055580060996
7.971712539104554 2.108224331384976 9.964123975596852 2.6041214819647873
6.142294587564537 1.8376453321014508

2.0207294060317125 2.7416742190548513 1.8846637197306482 4.118030601761058
4.3773227972659425 0.9780181459763382 0.7934164966202428 6.687267979171926
9.042138333003333 9.02612921427598

4.726404951529716 0.7166700826219929 2.923932763493784 5.036207155180527
7.755087911883713 2.8683703053965637 0.16858276867266486 5.000478441340498
9.336607158743671 3.2561643796075614

7.174714572451367 3.7816857608184327 4.618786763048708 2.641398055001256
9.013037553203429 7.011914493901349 4.181823804304521 8.41621280559739
4.147879871960773 8.360344601129073

1.9755240831956788 1.7096554880026116 1.5957530659175945 1.482982118570233
0.7285081781976499 7.297867313589541 0.9546004052447354 2.344371622222857
8.685895705611472 3.907769265639799

1.098711778038095 5.540744601417704 0.17455223323552427 4.974361599514289
7.758116249483859 7.756011965736879 5.200060973671232 8.944961610505826
3.6875910449281815 2.5464651794837225

References:

[1] : https://cssanalytics.wordpress.com/2013/11/26/fast-threshold-clustering-algorithm-ftca/