B534/E599 Project2B: Harp Kmeans

Goal

You will implement a parallel version of K-means using the programming interfaces of Hadoop and Harp.

Deliverables

You are required to turn in the following items in a zip file (username_HarpKmeans.zip) in this assignment:

1. The source code of Harp K-means you implemented.
2. Technical report (username_HarpKmeans_report.docx) that contains:
   a. The description of the main steps and data flow in your program.
   b. The data files you generated and the output centroid file.

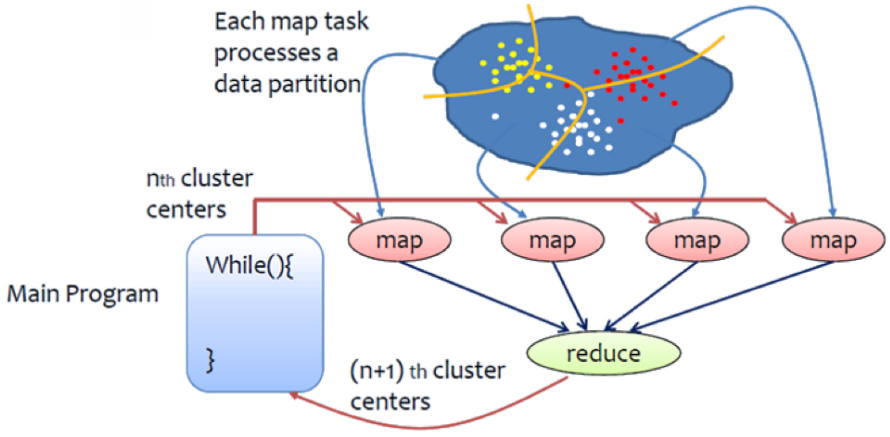
Evaluation

The point total for this project is 10, where the distribution is as follows:

1. Completeness of your code and output (7 points)
2. Correctness of written report (3 points)

K-means

This section describes how to implement the K-means algorithm using Harp.



K-Means is a clustering algorithm that divides a given set of points into "K" partitions. "K" needs to be specified by the user. In order to understand K-Means, first you need to understand a few terminologies.

Centroids

Centroids can be defined as the center of each cluster. If we are performing clustering with k=3, we will have 3 centroids. To perform K-Means clustering, the users need to provide the initial set of centroids.

Distance

In order to group data points as close together or as far-apart we need to define a distance between two given data points. In K-Means clustering, distance is normally calculated as the Euclidean Distance between two data points.

The K-Means algorithm simply repeats the following set of steps until there is no change in the partition

assignments, in that it has clarified which data point is assigned to which partition.

1) Choose K points as the initial set of centroids.
2) Assign each data point in the data set to the closest centroid (this is done by calculating the distance between the data point and each centroid).
3) Calculate the new centroids based on the clusters that were generated in step 2. Normally this is done by calculating the mean of each cluster.
4) Repeat steps 2 and 3 until data points do not change cluster assignments, meaning their centroids are set.

Implementation

1 Download Harp3-Project from https://github.iu.edu/IU-Big-Data-Lab/Harp3-Project

2 Set up hadoop and harp environment following the installation instruction we provided.

3 Implement your harp K-means under the directory:

```
$HARP3_PROJECT_HOME/harp3-app/src/edu/iu/km
```

You need to add jar files in **lib** directory to the build path when you do implementation using eclipse or other IDEs. Under this directory, you will find a simple code template to use.

For Hadoop APIs, please refer to the official documentation at https://hadoop.apache.org/docs/r2.6.0/
For Harp APIs, please refer to the slides on the canvas.

Pseudo Code and Java Interfaces

Definitions:

N is the number of data points
M is the number of centroids
D is the dimension of centroids
Vi refers to the ith data point(vector)
Cj refers to the jth centroid

The Main Method

The tasks of the main class are to configure and run the job iteratively.

```
generate N data points (D dimensions), write to HDFS
generate M centroids, write to HDFS
for iterations{
    configure a job
    launch the job
}
```

Let's look at the actual java implementation. The first step of the sample program is to generate a set of points. This is done using the following line in the main launch method. The actual method that is in the Utils.java class is also shown for reference. This method generates a set of data points and writes them into the specified folder. The method also takes in parameters to specify the number of files and number of data points that need to be generated. The data points are divided among the data files. You will generate "numMapTaks" data files, with each data file containing a part of the data points. Then you can generate data to localDirStr and use fs to copy the data from localDirStr to dataDir.

```
Utils.generateData(numOfDataPoints, vectorSize, numMapTasks, fs, localDirStr, dataDir);

//generateData method from Utils.java
public static void generateData( int numOfDataPoints, int vectorSize, int numMapTasks,
                                 FileSystem fs, String localDirStr, Path dataDir) throws IOException,
InterruptedException, ExecutionException{
        //TO DO

}
```

The next step is to generate a set of centroids. As mentioned in the description, K-Means needs a set of initial centroids. The "generateInitialCentroids" method in the Utils class will generate a set of random centroids. You can generate one file contains centroids and use fs to write is to cDir. JobID indicates the current job. It is optional.

```
Utils.generateInitialCentroids((int numCentroids, int vectorSize, Configuration configuration, Path cDir,
FileSystem fs, int JobID);
```

After the completion of initialization steps, the main class will run a set of map reduce jobs iteratively. The number of iterations are specified by the user. The following code block at each iteration configure a job and run it. When configuring a job, you can use MultiFileInputFormat class in edu.iu.fileformat to set inputFormatClass: job.setInputFormatClass(MultiFileInputFormat.class). In this way, every map task will load data file paths. Then you can read data from the file paths in map task.

```
for(int iter = 0; iter < numIteration; iter++){
    //delete output directory if existed
    if( fs.exists(outDir)){
        fs.delete(outDir, true);
    }

    //configure a job
    //waitForCompletion(job)
}
```

The MapperCollective

```
1 Read data from point files.
2 load centroids
3 find the nearest centroid Cj for all data points V
4 allreduce centroids, write back to HDFS
5 repeat 2~4
```

For the java implementation, a setup method will be called to initialize the mapper class. In the setup method, all the needed configurations will be loaded. The main map task is handled in the mapCollective function. In the mapCollective function, it first gets data files from KeyValReader and then reads data from files. It then loads initialization centroids from HDFS. For each data point, it calculates distances between the current data point and each centroid to determine the closest centroid to the data point.

Now you can use allreduce function to generate the new global centroids. And then write the new centroid values to the centroid file. In the next iteration these newly created centroids will be read in by the map task.

```
protected void mapCollective( KeyValReader reader, Context context) throws IOException,
InterruptedException {


}
```

Compilation & Run

1. compile harp3-app

Enter "harp3-app" home directory and execute "ant". Then copy build/harp-app-hadoop-2.6.0.jar to $HADOOP_HOME

```
cd $HARP3_PROJECT_HOME/harp3-app
ant
cp build/harp-app-hadoop-2.6.0.jar $HADOOP_HOME
```

2. Start Hadoop

```
$HADOOP_HOME/sbin/start-dfs.sh
$HADOOP_HOME/sbin/start-yarn.sh
```

3. Run Kmeans job

Use the following command to run harp kmeans:

```
cd $HADOOP_HOME

hadoop jar harp3-app-hadoop-2.6.0.jar edu.iu.km.KmeansMapCollective <numOfDataPoints> <num of
Centroids> <size of vector> <number of map tasks> <number of iteration> <workDir> <localDir> );
```

<numOfDataPoints>: the number of data points you want to generate randomly
<num of centriods>: the number of centroids you want to clustering the data to
<size of vector>: the number of dimension of the data
<number of map tasks>: number of map tasks
<number of iteration>: the number of iterations to run
<work dir>: the root directory for this running in HDFS
<local dir>: the harp kmeans will firstly generate files which contain data points to local directory. Set this argument to determine the local directory.

For example:

```
hadoop jar harp3-app-hadoop-2.6.0.jar edu.iu.km.KmeansMapCollective 1000 10 10 2 10 /kmeans
/tmp/kmeans
```

To fetch the results, use the following command:

```
hdfs dfs –ls /
```