```java
import java.io.*;

/*-------------------------------------------------------------
 * Tony Hyun Kim
 * CS 246, PS 2, Problem 4
 * k-means on MapReduce
 *-------------------------------------------------------------
 */
public class kmeans
{
    // Map assigns data to the k-centers
    public static class Map extends Mapper<LongWritable, Text, IntWritable, VectorWritable>
    {
        // Pulls data from DistributedCache into local memory
        ArrayList<VectorWritable> centers;
        void loadInitCenters(Path cachePath) throws IOException
        {
            BufferedReader cReader = new BufferedReader(new FileReader(cachePath.toString()));
            try
            {
                String line;
                this.centers = new ArrayList<VectorWritable>();
                while((line = cReader.readLine()) != null)
                    this.centers.add(new VectorWritable(line));
            }
            finally
            {
                cReader.close();
            }
        }
        void configure(Configuration conf)
        {
            try
            {
                Path[] cacheFiles = DistributedCache.getLocalCacheFiles(conf);
                if(cacheFiles != null && cacheFiles.length > 0)
                    for(Path cachePath : cacheFiles)
                        loadInitCenters(cachePath);
            }
            catch (IOException ioe)
            {
                System.err.println("IOException reading from distributed cache");
                System.err.println(ioe.toString());
            }
        }

        public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException
        {
            // Load the updated list of centers from the DistributedCache
            configure(context.getConfiguration());
```

```
63
64              // This is the input data point
65              VectorWritable x = new VectorWritable(value.toString());
66
67              // Let's find the center nearest to x
68              int Ncenters = centers.size();
69              int minIdx = 0;
70              double minDist = x.minus(centers.get(minIdx)).magnitude();
71              for(int i=1; i<Ncenters; i++)
72              {
73                  double dist = x.minus(centers.get(i)).magnitude();
74                  if(dist<minDist)
75                  {
76                      minIdx  = i;
77                      minDist = dist;
78                  }
79              }
80
81              // Send datum to the new cluster
82              context.write(new IntWritable(minIdx),x);
83
84              // This is a special flag to compute the cost function
85              context.write(new IntWritable(-1),new VectorWritable(minDist*minDist));
86          }
87      }
88
89      // Reduce recomputes the center locations to be the respective center of mass
90      public static class Reduce extends Reducer<IntWritable, VectorWritable,
    IntWritable, VectorWritable>
91      {
92          public void reduce(IntWritable key, Iterable<VectorWritable> values, Context
    context)
93                  throws IOException, InterruptedException
94          {
95              // Compute new center of mass
96              Iterator<VectorWritable> iter = values.iterator();
97              VectorWritable sum = new VectorWritable(iter.next());
98              int count = 1;
99              while(iter.hasNext())
100             {
101                 sum.accumulate(iter.next());
102                 count++;
103             }
104
105             if(key.get() != -1) // Just a regular cluster
106                 context.write(null, sum.times(1.0/count));
107             else // Compute the cost function
108             {
109                 // We're going to write a 'cost.txt' directly to HDFS...
110                 String outDir  = FileOutputFormat.getOutputPath(context).toString
    ();
111                 Path scorePath = new Path(outDir+"/cost.txt");
112                 FileSystem fs = FileSystem.get(context.getConfiguration());
```

```
113                 BufferedWriter writer = new BufferedWriter(new OutputStreamWriter
    (fs.create(scorePath,true)));
114                 writer.write(sum.toString()+"\n");
115                 writer.close();
116             }
117         }
118     }
119 
120     // THK: Trying out DistributedCache
121     //  Despite the excess initialization efforts, DC should be useful in the future
122     //  Adapted from: http://developer.yahoo.com/hadoop/tutorial/module5.html
123 
124     public static final String baseHdfsPath = "/user/cs246/";
125     public static String LOCAL_INITCENTERS_LIST = "/home/cs246/Desktop/dataset/
    c2.txt";
126 
127     public static void main(String[] args) throws Exception
128     {
129         Path inputPath = new Path(args[0]);
130         String outDir  = baseHdfsPath + args[1];
131         int numIter = Integer.valueOf(args[2]);
132 
133         for(int iter = 0; iter < numIter; iter++) // K-means iterations
134         {
135             Path centerPath = new Path(outDir+"/iter"+String.format("%03d",iter)+"/
    part-r-00000"); // Previous output
136             Path outputPath = new Path(outDir+"/iter"+String.format("%03d",iter+1));
137 
138             Configuration conf = new Configuration();
139 
140             if(iter==0) // On the first iteration, we'll copy the initial list of
    centroids
141             {
142                 FileSystem fs = FileSystem.get(conf);
143                 fs.copyFromLocalFile(false, true, new Path
    (LOCAL_INITCENTERS_LIST),centerPath);
144             }
145             DistributedCache.addCacheFile(centerPath.toUri(), conf);
146 
147             Job job = new Job(conf, "kmeans");
148             job.setJarByClass(kmeans.class);
149             job.setOutputKeyClass(IntWritable.class);
150             job.setOutputValueClass(VectorWritable.class);
151 
152             job.setMapperClass(Map.class);
153             job.setReducerClass(Reduce.class);
154 
155             job.setInputFormatClass(TextInputFormat.class);
156             job.setOutputFormatClass(TextOutputFormat.class);
157 
158             FileInputFormat.addInputPath(job, inputPath); // Input is always the
    dataset
159             FileOutputFormat.setOutputPath(job, outputPath);
```

```
160
161             job.waitForCompletion(true);
162         }
163     }
164 }
165
```