DEPARTMENT OF COMPUTER SCIENCE, UNIVERSITY OF GEORGIA

CSCI6900 Assignment 2: K-Means on Hadoop

DUE: Monday, February 23 by 11:59:59pm

Out Wednesday, February 4, 2015

1 OVERVIEW

K-Means iteratively improves the partition of the data into k sets:

- Predefine the number of clusters, k.
- Initialize k cluster centroids.
- Iterate until the centroids no longer change.
  - Associate each data instance with the nearest centroid (we consider them in a Euclidean space for this assignment).
  - Recalculate centroids as an average of the associated data instances.

2 K-MEANS CLUSTERING ON MAPREDUCE

To parallelize K-Means on MapReduce, we are going to share some small information, i.e. the cluster centroids, across the iterations. This will result in a duplication, but very minimal comparing with the large amount of data.

Therefore, before starting, a file is created accessible to all processors (through `FileSystem` in `Configuration`) that contains the initial k cluster centroids. This file will be updated after each iteration to contain the latest cluster centroids calculated by Reducer. Then

1. The **Mapper** reads this file to get the centroids from last iteration. It then reads the input data and calculates the Euclidean distance to each centroid. It associates each instance with the closest centroid, and outputs (data instance id, cluster id).

2. Since this is a lot of data, we use a **Combiner** to reduce the size before sending it to Reducer. The Combiner calculates the average of the data instances for each cluster id, along with the number of the instances. It outputs (cluster id, (intermediate cluster centroid, number of instances)).

3. The **Reducer** calculates the weighted average of the intermediate centroids, and outputs (cluster id, cluster centroid).

The main function runs multiple iteration jobs using the above Mapper + Combiner + Reducer. You can use the following sample codes to implement the multiple iterations in main:

```
int iteration = 0;

// counter from the previous running job
long counter = job.getCounters().findCounter(Reducer.Counter.CONVERGED).getValue();

iteration++;
while (counter > 0) {
    conf = new Configuration();
    conf.set("loops.iter", iteration + "");
    job = new Job("KMeans " + iteration, conf);

    // ...
    // job.set Mapper, Combiner, Reducer
    // ...

    // Take the output from the last iteration as the input to the next iteration.
    in = new Path("files/kmeans/iter_" + (iteration - 1) + "/");
    out = new Path("files/kmeans/iter_" + iteration);

    // ...
    // job.set Input, Output
    // ...

    // Run the job and update the counter.
    job.waitForCompletion(true);
    iteration++;
    counter = job.getCounters().findCounter(Reducer.Counter.CONVERGED).getValue();
}
```

You can define an enum counter (as you did in the previous homework), and update the counter in the Reducer if a centroid is updated:

```
context.getCounter(Counter.CONVERGED).increment(1);
```

3 DATA

For this assignment, we are using a truly "big" dataset: 20 million small images [1], procured from the web and used in several recent publications [2] [3]. Each image is RGB 32×32 pixels.

As the analysis of images is its own area of research, this assignment simplifies the process of quantitative image description by providing you with the feature vectors that will effectively represent each of the 80 million images in your algorithm. We use the "gist" vectors, 384-dimensional codes that you can consider locality-sensitive hashes of each image. Put simply, if the images are "similar," then their gist vectors will also be similar. This relationship will enable us to cluster the gist vectors and to discover groups of images depicting similar events.

The data appears at `s3://uga-mmd/tinyimages/`. Each row is a single data instance, written as a key-value pair: the key is an integer representing the image ID, and the value is a comma-separated list of the 384 dimensions of the gist vector describing that image. The key and value are separated by the tab `"\t"` character. Like your previous assignment, a much smaller toy dataset is provided for debugging.

```
gist.small.txt
gist.full.txt
```

NOTE: These files are BIG. Even the `small` dataset, which contains 0.01% of the original data, is still 2,000 images. $2,000 \times 384$ is still nearly 1 million floating-point values. The full dataset, $20,000,000 \times 384$, is over 7.6 *billion* floating point values. Furthermore, the 20 million images used in this homework is only a quarter of the full, original dataset of *80 million images!*

For convenience of grading, the initial cluster centroids for $k = 10$ and $k = 50$ were already randomly generated. Please use the following files as your starting points. Each row indicates a single cluster centroid as a key-value pair. The key is the cluster ID, and the value is a comma-separated list of the 384 dimensions. The key and value are separated by the tab character.

```
centroids10.small.txt
centroids50.small.txt
centroids10.full.txt
centroids50.full.txt
```

Rather than upload all 200+GB of images to S3, I've made the image dataset available on my local webserver, `ridcully`. You can view them at the following link from any machine connected to the UGA campus network: `http://ridcully.cs.uga.edu/assignment2/` . You will need this link to answer the questions in the next section.

[1] The Tiny Images Dataset: `http://horatio.cs.nyu.edu/mit/tiny/data/index.html`

[2] Small Codes and Large Image Databases for Recognition: `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.229.3256&rep=rep1&type=pdf`

[3] Spectral Hashing: `http://papers.nips.cc/paper/3383-spectral-hashing.pdf`

4 DELIVERABLES

When grading your assignments, I will use the **most recent commit in your repository *prior* to the submission deadline**. I should be able to clone your repository, change to the directory containing this assignment, and run your code. If my BitBucket username is `magsol`, my assignment should be in `magsol/assignment2`.

Please include a `README` file with 1) the command you issued to run your code, and 2) any known bugs you were unable to fix before the deadline. In addition, please provide answers to the following questions.

1. Run $k = 10$ and $k = 50$ clusters on the small data. Report the number of iterations for convergence and the wall time respectively. Within each cluster, identify the image ID that is **closest** to the centroid of the cluster.
2. Run $k = 10$ and $k = 50$ clusters on the full data. Report the number of iterations for convergence and the wall time respectively. Within each cluster, identify the image ID is **closest** to the centroid of the cluster.

   For each iteration, we compared each instance to each possible centroid, which may result in a large computation cost. We can reduce the number of distance comparison by applying the Canopy Selection, which we touched on in lecture and is described in `http://www.kamalnigam.com/papers/canopy-kdd00.pdf`. Please read the paper, and answer:

3. What distance metric would you choose for the canopy clustering? Why?
4. Can you implement the Canopy Selection on MapReduce? If yes, please describe the workflow.
5. Describe the workflow to combine the Canopy Selection with K-Means on MapReduce.

Bonus Question: Implement the Canopy Selection with K-Means on MapReduce. Run $k = 10$ on both small and full data. Report the number of iterations for convergence and the wall time respectively. Within each cluster, identify the image ID that is **closest** to the centroid of the cluster.

Finally, also include your controller and syslog files from running your AWS jobs.

5 MARKING BREAKDOWN

- Code correctness (commit messages, program output, and the code itself) **[45 points]**
- Question 1, 2 **[10 + 10 points]**

- Question 3, 4, 5 **[5 + 15 + 15 points]**
- Bonus Question **[25 pointes]**

6 OTHER STUFF

You may consider using the `GenericOptionsParser` for passing command line arguments. It parses all arguments that have the form `-D name=value` and turns them into name/value pairs in your `Configuration` object.

Now that Assignment 1 is behind you and you're all experts in Hadoop hacking, this assignment will likely feel much easier. Nevertheless, do try to start early again, if only to get a feel for where the "hard parts" of the programming assignment are; for instance, you will almost certainly have to make use of the `DistributedCache` in this assignment to pass around the intermediate cluster centroids. Furthermore, creating a robust loop in the main driver class that reads from the output of the previous iteration can also be very tricky. So do try to start early.