PROJECT 2 - MAHOUT
CS525: BIG DATA ANALYTICS

Hao Zhou, James Forkey, Pengfei Geng

TASK 1: SCALABLE K-MEANS THE MAHOUT WAY

The Objective

Your main task is then to apply clustering on at least 2 different twitter data sets to produce "interesting" clusters (of size 100MB or later). This will require you to prepare the data for loading into Mahout. More importantly, please experiment with different parameter settings and then report on your findings. Your results should explain what results you found (displaying the actual cluster results and their description in terms of top 10 key words). Discuss how meaningful you could make the results by varying the different parameters. Consider extreme cases of K=2 to K=10,000, and so on. Similarly, also report on the execution times for different parameter settings.

The Results

As expected, increasing the number of clusters exponentially effects required processing time. However, we found that the time differential between 2 and 20 clusters is insignificant. However, when scaling up to 5000 clusters, the system is greatly taxed and processing time is quite lengthy. Thus, in conclusion, it is obvious that determining the number of clusters is essential. The methods for determining cluster size vary greatly and largely depend on the intent of use from the resulting data.

Top 10 words from cluster 1 of 20

VL-119759{n=8712 c=[00am:0.000, 00pm:0.000, 05:0.000, 08:0.000, 0asts:0.000, 1.0:0.000, 101:0.000, 1

Word	Frequency
Game	0.12846281521675249
Wtf	0.07720465000954321
Out	0.02904207892151469
Rt	0.02542220151399761
I	0.018674732437807724
He	0.01670711950100747
Good	0.014910927052691998

Go	0.0147710907882388 19
My	0.012668613586772032
Bears	0.012573708087687801

Top 10 words from cluster 2 of 20 :VL-119813{n=2402 c=[00am:0.000, 00pm:0.001, 08:0.000, 11:0.003, 12pm:0.000, 131:0.000, 13th:0.001, 1	
Word	**Frequency**
Friday	0.27979987901263126
Rt	0.03911122498508451
I	0.028397967481543235
Night	0.026368898273964866
Black	0.023744063939000536
Thursday	0.019878447182105928
Saturday	0.018701835086563694
Monday	0.017868544745403166
Wait	0.01764897047586519
Go	0.016578556182028595

Elapsed Time

Time (in minutes)	Process
21.00	Hadoop fs-put [Note: Inaccurate!]
19.78	Mahout seqdirectory
02.01	Majout seq2sparse
00.90	Mahout kmeans [Cluster-20]
00.04	Mahout clusterdump
76.56	Mahout kmeans [Cluster-5000]

TASK 2: ROLL YOUR OWN K-MEANS

The Objective

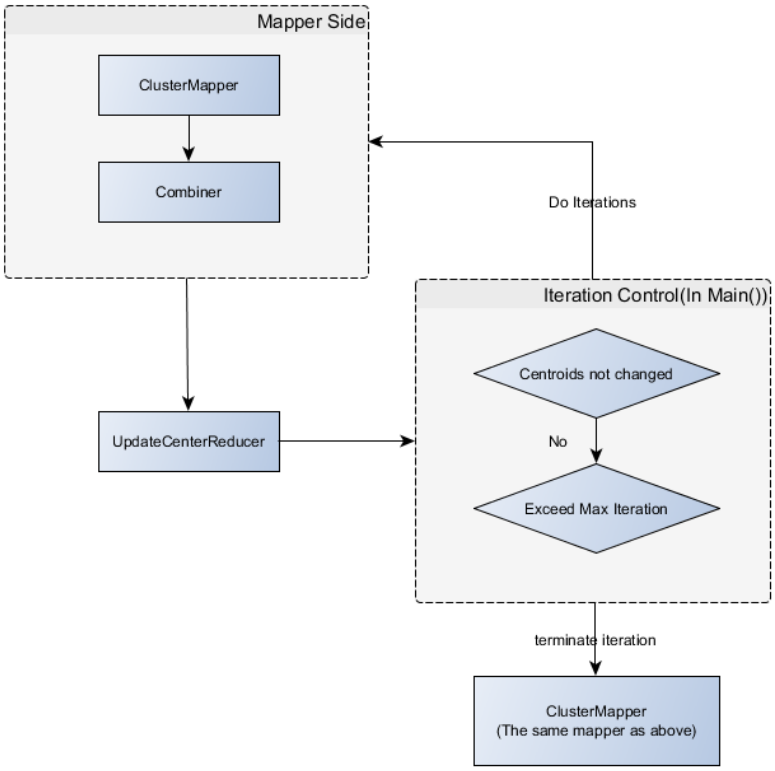
Write map-reduce job(s) to implement your own version of the K-Means clustering algorithm directly on top of Hadoop. Your system should terminate if either of these two conditions become true:

a) The K centers did not change over two consecutive iterations

b) The maximum number of iterations has been reached (parameter R for rounds)

Consider in your design that since the algorithm is iterative, you need your program that generates the map-reduce jobs to also control whether it should start another iteration or not. You are welcome to refine the above termination conditions by considering some delta by how much the center points are allowed to migrate in a given iteration without requiring another iteration to be kicked off. But this is optional.

Design



Architecture Figure

How it works

1. Compile source and make it as kmeansCluster.jar
2. Run it with command*:*
   hadoop jar kmeansCluster.jar iteration-numbers Threshold [Enable-detail-Cluster-info-dump]

   For example *hadoop jar kmeansCluster.jar 6 10 0* means max iteration = 6, threshold = 10 (if distance between two points, it means they are the same), no instructions for cluster, as the last parameter is 0. (If max iteration = 0, it means that program terminates after it converges)

Input & Output

1. **ClusterMapper**:
   a. input is a `point`
   b. Compare its distance to all the centroids, then find the nearest one
   c. output: **key**: centroid.x, centroid.y --- **value**: point.x, point.y
2. **Combiner**:
   a.just do aggregation for mapper result here
   b. output: **key(no change from mapper)**: centroid.x, centroid.y --- **value**: sum(point.x), sum(point.y),*countOfPointsForThisKey*
3. **UpdateCenterReducer**:
   Two steps here
   Step 1. Do the same work as combiner for global points
   Step 2. According to the result from step 1, generate new centroid, and compare new centroid with previous one
   output: **key**: newCentroid.x, newCentroid.y --- **value**: **1** if centroid not changed, otherwise **0**

TASK 3: COMPARING THE TWO

The Objective

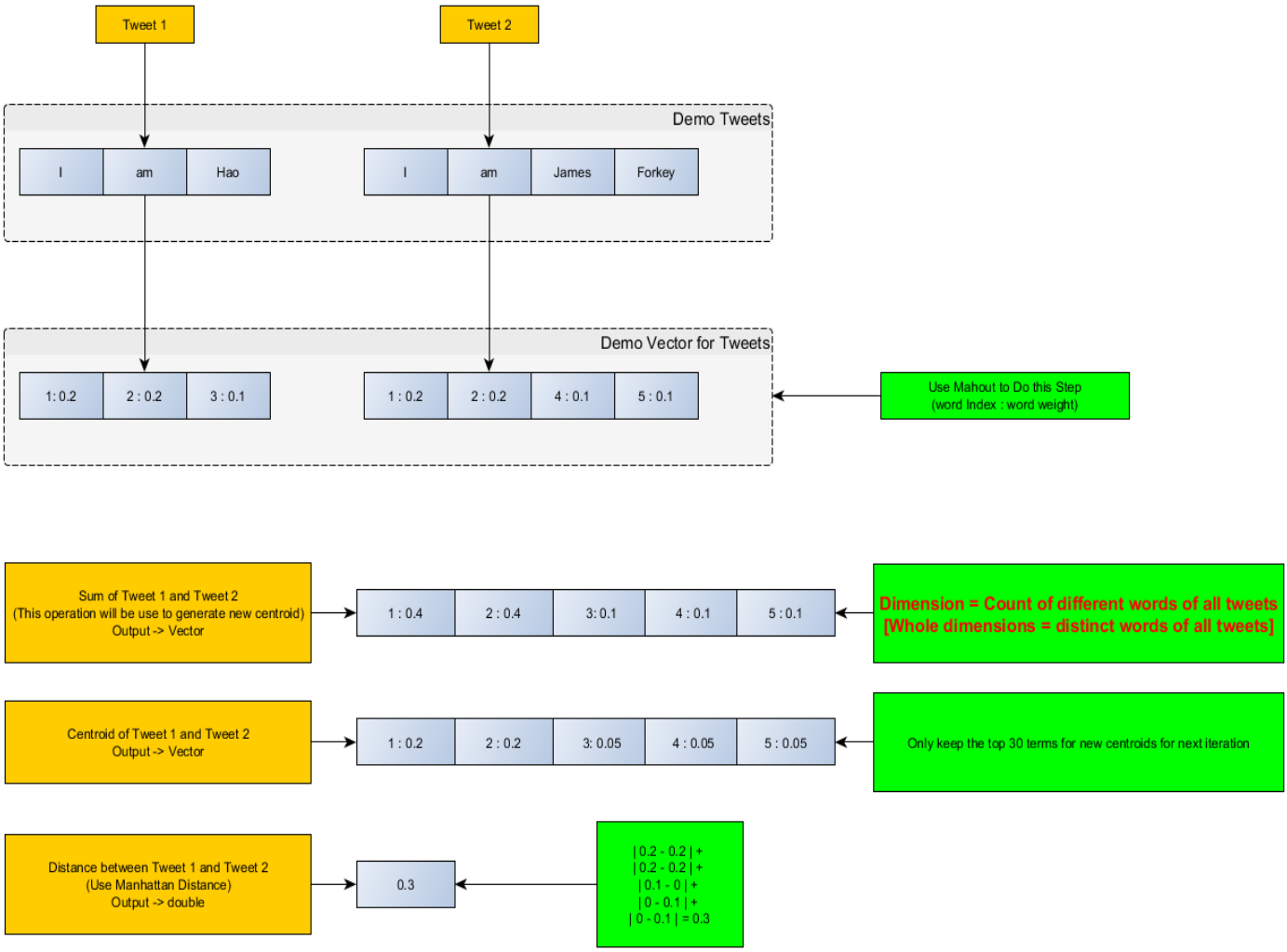
a. This on the one hand means that you need to consider how to convert your data into numeric feature vectors before you start up your clustering for the distance function to be an easier task.
b. Second, you may want to reconsider the distance function that you use, if you wish.

c. Third, you ideally would link the numeric output vectors with their corresponding textual tweet objects; so that at the very end you can produce the content of your clusters in a meaningful manner.
d. And, better yet, the final more advanced task would be to produce descriptors for each of your clusters by providing summaries of them in the form of their frequent keywords or any other descriptors that make sense.

Then compare the performance numbers of mahout solution versus your own technologies for the identical twitter data set and with each of the parameter settings being as similar as possible.

Third very briefly compare the capabilities of your solution with those of mahout, in terms of ease of use, your effort, etc.

Design



The Results

1). MapReduce:

1. **9** iterations –Iterations outputs the **new centroids**.
   [start at 13:38:57 ~ end at 13:52:34] = 13 minutes 37 seconds

```
CPU Time  1: 69670
CPU Time  2: 75660
CPU Time  3: 77180
CPU Time  4: 76380
CPU Time  5: 77490
CPU Time  6: 77170
CPU Time  7: 78300
CPU Time  8: 77340
CPU Time  9: 77940
```

2. **Final output of tweet Clusters** -- map only
   [start at 13:52:34 ~ end at 13:52:42] = 8 seconds

```
CPU Time:  3670
```

3. **Descriptors**: The output of top 10 terms for each cluster is here

2). Mahout for the same task

job_201311140804_0031	Thu Nov 14 15:13:33 EST 2013	NORMAL	ubuntu	PartialVectorMerger::M
job_201311140804_0032	15:24:06 EST 2013	NORMAL	ubuntu	Cluster Iterator runni /kmeans-clusters/clus
job_201311140804_0033	Thu Nov 14 15:24:29 EST 2013	NORMAL	ubuntu	Cluster Iterator runni /kmeans-clusters/clus
job_201311140804_0034	Thu Nov 14 15:24:51 EST 2013	NORMAL	ubuntu	Cluster Classification /vectors/tfidf-vectors

```
ed.JobClient:     Physical memory (bytes) snapshot=44994560
ed.JobClient:     Spilled Records=0
ed.JobClient:     CPU time spent (ms)=470
ed.JobClient:     Total committed heap usage (bytes)=16252928
ed.JobClient:     Virtual memory (bytes) snapshot=384237568
ed.JobClient:     Map output records=14437
ed.JobClient:     SPLIT_RAW_BYTES=127
er.MahoutDriver: Program took 65258 ms (Minutes: 1.0876333333333332)
```

```
mahout kmeans \
```

```
-i /task4/vectors/tfidf-vectors/  \
-c /task4/kmeans-centroids  \
-cl \
-o /task4/kmeans-clusters  \
-k 10 \
-ow \
-x 9 \
-dm org.apache.mahout.common.distance.CosineDistanceMeasure
```

It appears that Mahout's K-means beats our method by a landslide. This might be due to the large support community Mahout has backing it up compared to a few students in a graduate course. Even if we set the iteration count to 9, Mahout finishes the cluster job in 3 iterations. This becomes much more evident as we scale the dataset as well. We see that increasing the number of records to 110,000 drastically decreases performance; our Map-Reduce job takes 40 minutes, compared to Mahout's 1 minute. If this was the entire focus of the course and we were given the entire semester to implement what we believe to be the most efficient and K-means solution, our results may rival Mahout's outstanding performance.

Descriptors: The output of top 10 terms for each cluster is here