Name: K Means Data Clustering using Sequential, OpenMp and MapReduce Methods

Group Member: Peijun Wu, Yingping Zhao

Background:

For graph of large size, the analysis of it using global statistics is often not enough. One interesting property of network is the arise of community, or cluster, within which the nodes are more similar and between which nodes are less similar. The definition of clusters varies for different applications. Everitt[1] gives three definitions:1) A cluster is a set of similar objects and the objects in different clusters should not be similar; 2) A cluster is the set of points which are aggregated in the testing environment such that the distance between two points in a cluster is less than the distance between the points of other clusters; 3) Clusters can be densely connected regions in a multi-dimensional space separated by loosely connected points. This project adopts the second definition with the use of K-means clustering algorithm.

In practice, clustering requires many experimentations with different algorithms or with different features of the same data set. Hence, they are computationally expensive and saving computational complexity is a significant issue for the clustering algorithms. Therefore, the parallelization of clustering algorithms is of practical significance.

MapReduce[2] is a programming model and an associated implementation for processing and generating large datasets. This technique is inspired by the *map* and *reduce* primitives present in many functional programming languages like Lisp. In practice, many computations can be realized using this programming model. It has an advantage of dealing with millions of objects because it doesn't store all objects in the main memory.

Project Implementation:

The project includes two parts. Part I generates graph with different sizes of points and implement sequential K-means algorithm and a simple parallel algorithm with OpenMP. Part 2 implements parallel K-means algorithm with OpenMP based on MapReduce.

Part I

Sequential K-means Clustering and simple Parallel K-means Clustering

The algorithm of the Basic K-mean Algorithm is expressed as the following

1. Select K points randomly from the dataset as initial cluster centers.
2. Assignment Step: Form K clusters by assigning each point to its closest cluster center (based on Euclidean distance).
3. Update Step: Update the cluster center for each cluster as the mean position of all the points assigned to that cluster.

4. **Repeat step 2 and step 3 until** the cluster center does not change. (Converge!)

Part I has the following function implemented:

1) Generate a graph file composed of N two dimensional data points, which is used for the analysis part, users can specify the number of points included in the graph;
2) With the generated graph as the input, the K-means Clustering algorithm implemented clustering, with user input the dimension of the dataset, number of points of the dataset, and an output file with each point assigned to the final clusters;
3) With the same functionality of the Kmeans Clustering program as described above, the distance calculation part is paralyzed by using the OpenMP.

The following 3 sections will show the 3 functions in the first part in detail.

1.1 Input File generation

gen_dataset.c is used to generate the input file for the K-means Cluster Analysis
Users can define the dimension of the data point and number of data points.
For example, if the user specifies 2, 2000000 as the input parameter, then a dataset with 2000000 two-dimensional data points will be generated for future data analysis.

1.2 Sequential K-means Clustering Method

Kmeans.c is used for the sequential K-means clustering implementation, and the user has to specify the dataset dimensions, number of points included in the dataset, the input file name, the output file name.
For example, if the user specifies "2, 2000000, inputfile.txt out" the input parameter, then the dataset with 2000000 two-dimensional data points (inputfile.txt) will be used for the data analysis, and then a file called out.csv will be generated. The output file includes the X coordinates, Y coordinates and assigned cluster number for each data point in the dataset.
The procedure for this sequential K-means clustering is divided into the following steps with Pseduo codes provided for the key functions.

- Random choose K data points from the input dataset to function as the initial cluster centroids
- Calculate the distance of each data point to all the preciously assigned cluster centroids
  The input data array is initialized with each datapoint's distance to each cluster equals to zero first.

Calculate Distance(input_data, centroids array, # of points)

Input: input data array, centroids array,
Output: data array with distances of each data point to each cluster center calculated and attached

```
1. For each data point i, the distance of the data point to a cluster centroid j is
   $Dist_{ij}$ = sqrt(square(diff(X_coordinates)) + square(diff(Y_coordinates)))
2. for i =1 to # of points
     for j =1 to # of clusters
       compute $Dist_{ij}$;
3. Update the input data matrix
```

Assign Cluster(input_data)

Input: input data array with newly calculated distance
Output: data array with newly assigned centroids number

```
1. for i =1 to # of points
     for j =1 to # of clusters
       compare $Dist_{ij}$;
       remember the k, which $Dist_{ik} = \min(Dist_{ij}), \forall j$;
2. Update the input data matrix with data points's assigned centroid number updated
```

Calculate Centroids(input_data, centroids array)

Input: input data array, centroids array,
Output: newly updated centroids array, for each cluster's new centroid j, the coordinates are calculated as the average distance of all the data points who is assigned with j,

```
1. for j =1 to # of clusters
       for i =1 to # of points
           if point is assigned to cluster j, $num_j$++
           $x_j$ += X_coordinate;
           $y_j$ += Y_coordinate;
           $x_j = x_j / num_j$;
           $y_j = y_j / num_j$;
2. New Centriods updated and stored in the centroids array
```

1.3 OpenMp K-means Clustering

Kmeans_mp.c is used for the OpenMP Parallel K-means clustering implementation. Same as the sequential Kmeans method, the user also has to specify the dataset dimensions, number of points included in the dataset, the input file name, the output file name.
For example, if the user specifies "2, 2000000, inputfile.txt out" the input parameter, then the dataset with 2000000 two-dimensional data points (inputfile.txt) will be used for the data analysis, and then a file called out.csv will be generated. The output file includes the X coordinates, Y coordinates and assigned cluster number for each data point in the dataset.

The only difference between this Parallel K-means and the sequential K-means is that the parallel computing is expected to be much faster than the one without parallelization.

For the distance calculation, parallel computing using the OpenMp is used. And for the two layers of loops, collapse function is used, also the inner layer variable is set as the private variable in order to make the code function correctly.

1.4 Results Comparison

We set the cluster to be 6, the maximum iteration umber to be 100, for different number of data points input and different number of threads, the runtime for sequential and simple openmp is shown below:

Runtime * Comparison				
Datapoints	**Sequential K-Means**	**OpenMP Thread #2**	**OpenMP thread #3**	**OpenMP Thread #4**
20,000	0.66	0.59	0.55	0.53
200,000	7.13	5.85	5.534	5.24
2,000,000	71.23	58.9	56.1	52.48
5,000,000	166.68	146.6	139.2	132

Scalability				
Datapoints	**Sequential K-Means**	**OpenMP Thread #2**	**OpenMP thread #3**	**OpenMP Thread #4**
20,000	-	11%	17%	20%
200,000	-	18%	22%	27%
2,000,000	-	17%	21%	26%
5,000,000	-	12%	16%	21%

NOTE: All the runtime is tested 10 times and an average is used to represent the real run time.

The Scalability is calculated as the representative of the runtime improvement based from the sequential K-means method.

The following can be seen from this two tables:

1. As the number of points increase, the runtime for both methods increase
2. As the number of usable threads increases, the running time decrease more, which means the scalability of the OpenMp is better
3. The scalability of 2 usable threads is around 15%, and the scalability of 3 usable threads is around 20%, and the scalability of 4 usable threads is around 25% for this particular program. The reason why the scalability of more than 4 usable threads are not included are not included here is that the author's computer only has 4 maximum usable threads, so the testing results shows that the scalability of more than 4 number of threads are almost the same as the 4 usable threads.

Part II

MapReduce is the programming mode adopted by Google to process huge datasets. In practice, people find out that many computations can be transformed or divided into the mapping computation and reduction computation. The followings are basic concepts of the MapReduce model.

- Map
  - Compute each record into a set of key/value pairs (intermediate values)
- Reduce
  - Perform aggregation for derived key/value pairs with the same key
- Combine
  - Combine key/value pairs with the same key. (Often used as an optimization method to decrease the data traffic)

Basically we can see that the Mapping has a intuitive nature of parallel processing because each record is independent of all others and can thus be parallel processed without worrying about the race conditions. Whereas reduce is aggregating the results with the same key. Depending on the number of keys, this process can also be parallel computed.

2.1 Basic Structure:

map(key, String)	**combine(key, values from different segments)**	**reduce(key, values with the same key):**

for each value:
    dispatch threads to compute:
     IntermediateMap<key, value>;
	for each key:
	  Combine the values with that same key across all the segments;
		aggregate the values of the same key
		(Aggregate means compute one single value from a list of values. For example: max(list), sum(list), avg(list))

2.2 Implementation algorithm of K-means clustering using MapReduce model:

The following is the algorithm to implement K-means clustering based on MapReduce model.

Map(key, value)

Input: Global variable centers, segments of sample points, which is a struct containing attribute “cltr_assigned”, “xcoord, ycoord”
Output: a set of <key’, value > pairs for each segment, where the key’ is the “cltr_assigned” for each sample point, and value’ is the “xcoord, ycoord” of sample points..

```
1. Construct sample points from the segments;
2. For each sample point in the segment, repeat 3 to 5 steps:
3. minDist = ComputeDist(sample point, center[0]
4. for  I =1 to len(center), do:
       dis = ComputeDist(sample point, center[i]
       if dis< minDist{
              minDis = dis;
              index = i+1;
       }
5. Take index as “cltr_assigned”;
```

Combine(key, V)

Input: key the index of the clusters, V is segments of samples points.
Output: a set of <key’, value > pairs for each cluster, where the key’ is the index of the cluster, and the value’ is all the sample points assigned to the same cluster.

```
For each segment, repeat:
        For each sample point in the segment, repeat:
        Assigned the sample point to the list of the cluster .
```

Reduce(key, V)

Input: key is the index of the cluster, V is the list of sample points in the cluster of that index.
Output: a set of <key’, value > pairs for each cluster, where the key’ is the index of the cluster, and the value’ is coordinates of the new cluster center.

```
1.  Initialize num_points as 0;
2. For all the sample points in the list, do:
        sum_x += sample_point->xcoord;
        sum_y += sample_point->ycoord;
        num_points++;
3. X_new = sum_x/num_points;
4. Y_new = sum_y/num_points;
5. Take X_new and Y_new as the new coordinates for the new center.
```

2.3 Performance analysis of K-means based on MapReduce:

When the dataset has a great size, extra consideration should be taken. We found that there is a lot of work done behind the scene for distributing the tasks for the OpenMP task construct. By doing some comparison by tuning parameters, we construct table below.

Running time (wall time) of the K-means Clustering (unit is in seconds)					
Num_segs	Sequential	Thread = 1	Thread = 2	Thread = 4	Thread = 8
NUM_THREADS/4	0.3623	--	--	0.3062	0.1853
NUM_THREADS/2	0.3521	--	0.3348	0.1956	0.1280
NUM_THREADS	0.3556	0.3542	0.2201	0.1263	0.1025
NUM_THREADS*5	0.3509	0.3569	0.2223	0.1372	0.1038
NUM_THREADS*10	0.3535	0.3468	0.2188	0.1326	0.1071
NUM_THREADS*20	0.3546	0.3413	0.2196	0.1323	0.1564
NUM_THREADS*40	0.3515	0.3576	0.2229	0.1254	0.2880
NUM_THREADS*80	0.3977	0.3600	0.2275	0.2404	0.6907
NUM_THREADS*160	0.3579	0.3795	0.2259	0.5925	1.4474
NUM_THREADS*320	0.3513	0.3489	0.2413	1.4362	2.9270
NUM_THREADS*640	0.3632	0.3607	0.2501	2.9439	5.6778
NUM_THREADS*1280	0.3542	0.3870	1.4799	5.8450	11.6169
>num_points	--	>0.5	--	--	--

Table - K-means Clustering with OpenMP with dataset of size 20,000

(Note: NUM_OF_ITERATIONS = 100; NUM_CLUSTERS = 8; NUM_DIM = 2)

From the table above, we can draw the following conclusions:

1. The smaller the number of segments is, the faster. But the prerequisite is that the number of segments is N*NUM_THREADS, where N is an integer.
2. Only when the number of segments is set properly can we achieve scaling. Otherwise, the trend will become: the more threads, the slower.

Further discussion:

There are multiple data structure implementations for the same problem. Different data structures can affect the performance of sequential program, and more so for the parallel program. For this project, different data structures and OpenMP constructs were tried and then the performance is tested. To sum up, for part I, we used a whole data matrix to store the data and conduct the cal_distance() of K-means algorithm using OpenMP collapse construct. For part II we take the idea of Mapreduce, and then used linked-list to store the data and conduct K-means algorithm using OpenMP task construct. After we learnt that "the less the number of tasks, the better", we further adopted linked-lists of linked-lists to hold the whole dataset in smaller segments of datasets.

Reference:

[1] B. S. Everitt., Cluster analysis, Heinemann Educational [for] the Social Science Research Council, London :1974.

[2] Jeffrey Dean and Sanjay Ghemawat, MapReduce: simplified data processing on large clusters, OSDI: 2004.

[3] Mahwish Arif and Hans Vandierendonck, A Case Study of OpenMP applied to Map/Reduce-style Computations.