COMPARE TWO K-MEANS PARALLEL ALGORITHMS

Instructor: Dr. Russ Miller
Nannan Zhai
nannanzh@buffalo.edu
Introduction

K-Means Algorithm

K-Means Algorithm parallel using MapReduce

K-Means Algorithm parallel using MPI

Implementation

Reference
K-means algorithm

K-means algorithm is the most well-known and commonly used clustering method. It takes the input parameter, k and partitions a set of n objects into k clusters so that the resulting intra-cluster similarity is high whereas the intercluster similarity is low. Cluster similarity is measured according to the mean value of the objects in the cluster, which can be regarded as the cluster’s "center of gravity".
K-means algorithm

Firstly, it randomly selects k objects from the whole objects which represent initial cluster centers.
K-means algorithm

Each remaining object is assigned to the cluster to which it is the most similar, based on the distance between the object and the cluster center.
The new mean for each cluster is then calculated. This process iterates until the criterion function converges.

**K-means algorithm**
K-Means Algorithm parallel using MapReduce

MapReduce is a processing technique and a program model for distributed computing based on java. The MapReduce algorithm contains two important tasks, namely Map and Reduce. Map takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs).
Example of MapReduce

<table>
<thead>
<tr>
<th>data</th>
<th>target</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,1</td>
<td>1,1</td>
</tr>
<tr>
<td>2,2</td>
<td>2,2</td>
</tr>
<tr>
<td>3,3</td>
<td>3,3</td>
</tr>
<tr>
<td>11,11</td>
<td>11,11</td>
</tr>
<tr>
<td>12,12</td>
<td>12,12</td>
</tr>
<tr>
<td>13,13</td>
<td>13,13</td>
</tr>
</tbody>
</table>

1 class

2 class
Example of MapReduce

random two centroid

c1:(1,1)
c2:(11,11)
Example of MapReduce

1, 1
2, 2
3, 3
11, 11
12, 12
13, 13

cl : (1, 1)
c2 : (11, 11)

store two nodes
Example of MapReduce

c1:(1,1)
c2:(11,11)

node1

1,1
12,12
3,3

node2

11,11
2,2
13,13
Example of MapReduce

c1:(1,1)
c2:(11,11)

node1

1,1
12,12
3,3

map
combine
reduce

node2

11,11
12,12
13,13

map
combine

1,1
2,2
3,3
11,11
12,12
13,13
Step1. Map

Algorithm 1. map (key, value)

**Input:** Global variable centers, the offset key, the sample value

**Output:** <key', value'> pair, where the key' is the index of the closest center point and value' is a string comprise of sample information

1. Construct the sample *instance* from value;
2. minDis = Double.MAX_VALUE;
3. index = -1;
4. For i=0 to centers.length do
   
   dis = ComputeDist(instance, centers[i]);
   
   If dis < minDis {
     
     minDis = dis;
     
     index = i;
   }

5. End For
6. Take index as key';
7. Construct value' as a string comprise of the values of different dimensions;
8. output <key', value'> pair;
9. End
Step2. Combine

**Algorithm 2.** combine \((key, V)\)

*Input:* \(key\) is the index of the cluster, \(V\) is the list of the samples assigned to the same cluster
*Output:* \(<key', value'>\) pair, where the \(key'\) is the index of the cluster, \(value'\) is a string comprised of sum of the samples in the same cluster and the sample number

1. Initialize one array to record the sum of value of each dimensions of the samples contained in the same cluster, i.e. the samples in the list \(V\);
2. Initialize a counter \(num\) as 0 to record the sum of sample number in the same cluster;
3. while\((V\).hasNext())\{
   Construct the sample \(instance\) from \(V\).next();
   Add the values of different dimensions of \(instance\) to the array \(num++\);
   
4. \}
5. Take \(key\) as \(key'\);
6. Construct \(value'\) as a string comprised of the sum values of different dimensions and \(num\);
7. output \(<key', value'>\) pair;
8. End
Step3. Reduce

Algorithm 3. reduce (key, V)

Input: key is the index of the cluster, V is the list of the partial sums from different host
Output: <key', value'> pair, where the key’ is the index of the cluster, value’ is a string representing the new center

1. Initialize one array record the sum of value of each dimensions of the samples contained in the same cluster, e.g. the samples in the list V;
2. Initialize a counter NUM as 0 to record the sum of sample number in the same cluster;
3. while(V.hasNext()){
   Construct the sample instance from V.next();
   Add the values of different dimensions of instance to the array NUM += num;
}
4. }
5. Divide the entries of the array by NUM to get the new center’s coordinates;
6. Take key as key’;
7. Construct value’ as a string comprise of the center’s coordinates;
8. output <key’, value’> pair;
9. End
Demo
K-Means Algorithm parallel using MapReduce

We have performed the speedup evaluation on datasets with different sizes and systems. The number of computers varied from 1 to 4. The size of the dataset increases from 1GB to 8GB.

(a) Speedup  
(b) Scaleup  
(c) Sizeup
K-Means Algorithm parallel using MPI

Message Passing Interface (MPI) is a standard developed by the Message Passing Interface Forum (MPIF). MPI is a standard library specification designed to support parallel computing in a distributed memory environment.
K-Means Algorithm parallel using MPI
K-Means Algorithm parallel using MPI

**Input:** number of clusters $K$, number of data objects $N$

**Output:** $K$ centroids

1. MPI_INIT∥ start the procedure;
2. Read $N$ objects from the file;
3. Partition $N$ data objects evenly among all processes, and assume that each process has $N'$ data objects;
4. For each process, install steps 5-11;
5. Randomly select $K$ points as the initial cluster centroids, denoted as $\mu_k$ ($1 \leq k \leq K$);
6. Calculate $J$ in Formula (1), denoted as $J'$;
7. Assign each object $n$ ($1 \leq n \leq N$) to the closest cluster;
8. Calculate the new centroid of each cluster $\mu_k$ in Formula (2);
9. Recalculate $J$ in Formula (1);
10. Repeat steps 6-9 until $J' - J < \text{threshold}$;
11. Generate the cluster id for each data object;
12. Generate new cluster centroids according to the clustering results of all processes at the end of each iteration;
13. Generate a final centroid set $\text{Centroid}$ by Function Merge and output the clustering results: $K$ centroids;
14. MPI_FINALIZE∥ finish the procedure;
Two different K means parallel algorithms

1. Same Dataset
2. Same size
3. Same number of instance

Compare the performance (time cost is the key performance)

K-Means Algorithm parallel using MPI should be relatively stable and portable, and efficient in the clustering on large data sets.
# K-Means Algorithm parallel using MPI  

## Performance

<table>
<thead>
<tr>
<th>Code Number of Data Sets</th>
<th>MKmeans(ms)</th>
<th>SKmeans(ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>I/O</td>
<td>Clustering</td>
</tr>
<tr>
<td>1</td>
<td>12.5</td>
<td>0.2</td>
</tr>
<tr>
<td>2</td>
<td>22.2</td>
<td>0.2</td>
</tr>
<tr>
<td>3</td>
<td>48.6</td>
<td>7.6</td>
</tr>
<tr>
<td>4</td>
<td>72.0</td>
<td>8.8</td>
</tr>
<tr>
<td>5</td>
<td>179.3</td>
<td>45.4</td>
</tr>
<tr>
<td>6</td>
<td>251.3</td>
<td>57.9</td>
</tr>
<tr>
<td>7</td>
<td>1268.2</td>
<td>268.8</td>
</tr>
</tbody>
</table>

## TABLE III  

Comparison Results between MKmeans and SKmeans.
Reference

[1] Parallel K-Means Clustering Based on MapReduce (Weizhong Zhao1,2, Huifang Ma1,2, and Qing He1 2009)

[2] A Parallel K-Means Clustering Algorithm with MPI (Jing Zhang, Gongqing Wu, Xuegang Hu, Shiyeng Li, Shuilong Hao)