Improved MapReduce k-Means Clustering Algorithm with Combiner

Prajesh P Anchalia
Department Of Computer Science and Engineering
R V College Of Engineering
Bangalore, India.

Abstract— Data Mining and High Performance Computing are two broad fields in Computer Science. The k-Means Clustering is a very simple and popular data mining algorithm that has its application spread over a very broad spectrum. MapReduce is a programming style that is used for handling high volume data over a distributed computing environment. This paper proposes an improved and efficient method to implement the k-Means Clustering Technique using the MapReduce paradigm. The main idea is to introduce a combiner in the mapper function to decrease the amount of data to be written by the mapper and the amount of data to be read by the reducer which has considerably reduced the redundant MapReduce calls that have resulted in a significant reduction in the time required for clustering as it has decreased the read/write operations to a large extent. The implementation of Improved MapReduce k-Means Clustering has been clearly discussed and its effectiveness is compared to the regular implementation in an experimental analysis. The results consolidate this research by concluding that the Improved MapReduce Implementation of k-Means Clustering Algorithm out performs the regular implementation by over 300 seconds.

Keywords—MapReduce; k-Means; Data Mining; Hadoop; Distributed Computing

I. INTRODUCTION

Cluster analysis is a study of algorithms and methods of classifying objects. Cluster analysis does not label or tag and assign an object into a pre-existent structure; instead, the objective is to find a valid organisation of the existing data and thereby to identify a structure in the data. It is an important tool to explore pattern recognition and artificial learning.

A cluster is described as a set of similar objects or entities collected or grouped together. All entities within a cluster are alike and the entities in different clusters are not alike. Each entity may have multiple attributes, or features and the likeness of entities is measured based on the closeness of their features. Therefore, the crucial point is to define proximity and a method to measure it.

There are many clustering techniques and algorithms in use. K-means is the most common and often used algorithm. K-means algorithm takes an input parameter k, and partitions a set of n objects into k clusters according to a similarity measure. The mean values, or centroids, are a summary measure of the similarity of data objects within the same cluster. First, the algorithm randomly chooses k initial centroids and then iterates over the dataset. In each iteration k-means uses the similarity metric to associate each data object with its nearest centroid. Then, the algorithm computes a set of new centroids by taking the mean of all the data objects in each cluster respectively. The final step is to calculate the change in the centroid positions between the latest iteration and the previous one. The iteration terminates when the change in the centroid position is less than some pre-defined threshold.

With datasets in petabytes, this iteration of data point assignment and centroid calculation could be tedious tasks. One common method used to cluster huge datasets is to filter and reduce the dataset into a sample set and use this small representative set to find the probable cluster centers. The challenge here is in identifying the representative sample set. The choice of sample set directly impacts the final cluster centers. The other method is to distribute the dataset on multiple nodes and perform the calculation of centroids in a parallel mode on chunks of data at a time. This is similar to the Single Program Multiple Data (SPMD) algorithms that can be implemented using threads, MPI or MapReduce. The choice of an appropriate implementation strategy is based on the size of the dataset, complexity of the computational requirements, algorithmic synchronization, ease of programming and the available hardware profile. The type of applications we target have loosely coupled, easily distributable high volume data that require simple computation.

This research paper contains ten sections in all. Section I gives an introduction to clustering analysis. Section II throws light on MapReduce[7][11] programming. Section III explains the k-Means clustering algorithm. Section IV gives a pictorial representation of the MapReduce implementation of the k-Means Clustering algorithm. Section V gives the system deployment and experimental setup used. Section VI presents the algorithms used for regular k-Means Clustering. Section VII describes the implementation of regular MapReduce k-Means clustering. Section VIII presents the algorithms used for the improved MapReduce k-Means Clustering implementation. Section IX describes the implementation of the improved MapReduce k-Means clustering. Section X concludes the paper.
II. MAPREDUCE PROGRAMMING

There are separate Map and Reduce steps. Each step is done in parallel on sets of (key, value) pairs. Thus, program execution is divided into a Map and a Reduce stage, separated by data transfer between nodes in the cluster. The Map stage takes in a function and a section of data values as input, applies the function to each value in the input set and generates an output set. The Map output is a set of records in the form of (key, value) pairs stored on that node. The records for any given key could be spread across many nodes. The framework, then, sorts the outputs from the Map functions and inputs them into a Reducer. This involves data transfer between the Mappers and the Reducer. The values are aggregated at the node running the Reducer for that key. The Reduce stage produces another set of (key, value) pairs as final output. The Reduce stage can only start when all the data from the Map stage is transferred to the appropriate machine. MapReduce\textsuperscript{7}[11] requires the input as a (key, value) pair that can be serialised and therefore, restricted to tasks and algorithms that use (key, value) pairs.

The MapReduce\textsuperscript{7}[11] framework has a single master Job Tracker and multiple Task Trackers. Potentially, each node in the cluster can be a slave Task Tracker. The master manages the partitioning of input data, scheduling of tasks, machine failures, reassignment of failed tasks, inter-machine communications and monitoring the task status. The slaves execute the tasks assigned by the master. Both input and output are stored in the file-system. The single Job Tracker can be a single point failure in this framework.

MapReduce\textsuperscript{7}[11] is best suited to deal with large datasets and therefore ideal for mining large datasets of petabytes size that do not fit into a physical memory. Most common use of MapReduce\textsuperscript{7}[11] is for tasks of additive nature. However, we can tweak it to suit other tasks.

III. K-MEANS CLUSTERING ALGORITHM

K-means is a common and well-known clustering algorithm. It partitions a set of ‘n’ objects into ‘k’ clusters based on a similarity measure of the objects in the dataset. The clusters have a high intra-cluster and a low inter-cluster similarity. As the number of objects in the cluster varies, the center of gravity of the cluster shifts. This algorithm starts with the selection of the ‘k’ initial random cluster centers from the ‘n’ objects. Each remaining object is assigned to one of the initial chosen centers based on similarity measure. When all the ‘n’ objects are assigned, the new mean is calculated for each cluster. These two steps of assigning objects and calculating new cluster centers are repeated iteratively until the convergence criterion is met. Comparing the similarity measure is the most intensive calculation in k-means clustering. For ‘n’ objects to be assigned into ‘k’ clusters, the algorithm will have to perform a total of ‘nk’ distance computations. While the distance calculation between any object and a cluster center can be performed in parallel, each iteration will have to be performed serially as the center changes will have to be computed each time.

IV. FLOW CHART FOR THE MAPREDUCE IMPLEMENTATION OF THE K-MEANS CLUSTERING ALGORITHM

![Flow Chart](image_url)

Fig. 2. Structure of MapReduce Implementation of k-Means Clustering Algorithm
V. SYSTEM DEPLOYMENT AND EXPERIMENTAL SETUP

The k-Nearest Neighbor technique was implemented on a setup consisting of four nodes connected over a private LAN. One node was used as a Namenode and Job Tracker while the other three nodes were used as Datanodes and Task Trackers. All the four nodes had Intel i3 processors with 2.40 GHz and 2 GB memory. The Operating System running on all the 4 nodes was Ubuntu version 12.04. The programming language used to code the k-Means Algorithm in both regular and improved MapReduce implementation was JAVA. Apache Hadoop version 1.0.4 was installed on all the four nodes and the single node and consequent multimode configurations were done according to the guide found at [8][9].

VI. ALGORITHMS FOR REGULAR K-MEANS MAPPER AND REDUCER

K-means is a clustering algorithm used to cluster a set of data objects into a given number of clusters based on the distance measure between the points. Our first step is to generate a random dataset that would represent real-life data objects and create a seed centroid file.

A. Startup

Algorithm 1 Algorithm for Startup

Require:
- A set of d-dimensional objects \( X = \{x_1, x_2, \ldots, x_n\} \)
- k-number of clusters where \( k < n \)
- initial set of centroids \( C = \{c_1, c_2, \ldots, c_k\} \)
- \( \delta \) convergence delta

Output: a new set of centroids, number of iterations, final clusters, time taken to converge

1. load \((X, C)\)
2. current_centroids = \( C \)
3. initialise numIter, startTime, finalClusters
4. startTime = currentTime()
5. \( C' = \) perform MapReduce
6. new_centroids = \( C' \)
7. numIter = 1
8. while( change(new_centroids, current_centroids) > \( \delta \), do
9. current_centroids = new_centroids
10. \( C' = \) perform MapReduce
11. new_centroids = \( C' \)
12. numIter = numIter + 1
13. end while
14. endTime = currentTime()
15. timeToConverge = \( \) endTime - startTime
16. perform outlierRemoval
17. finalClusters = perform finalClustering
18. writeStatistics
19. return current_centroids, numIter, finalClusters, timeToConverge

We use a start-up program to initiate the process by generating random objects and inputting the initial set of centroids. Then, we call the mapper algorithm to assign the objects to clusters by calculating the distance between the centroids and the objects. The mapper algorithm calls the reducer to recalculate new centroids based on the current clustering. The start-up program then, evaluates the change in the centroid positions and compares it with the convergence delta that we have defined. If the change is more than \( \delta \), the start-up program iterates through the mapper and reducer functions again. When the change in centroid position is less than \( \delta \), we assume convergence. At this point, the algorithm generates the output files and creates the final clusters.

B. Mapper

Algorithm 2 Algorithm for Mapper

Require:
- A subset of d-dimensional objects of \( \{x_1, x_2, \ldots, x_n\} \) in each mapper
- initial set of centroids \( C = \{c_1, c_2, \ldots, c_k\} \)

Output: a list of centroids and objects assigned to each centroid separated by tab. This list is written hourly one line per data point and read by the Reducer program.

1. \( M_i = \{x_1, x_2, \ldots, x_n\} \)
2. current_centroids = \( C \)
3. distance \( (p, q) = \sum_{i=1}^{d} (p_i - q_i)^2 \) where \( p_i \) (or \( q_i \)) is the coordinate of \( p \) (or \( q \)) in dimension \( i \)
4. for all \( x_i \in M_i \) such that \( 1 \leq i \leq m \) do
5. current_centroid = null
6. minDist = \( \infty \)
7. for all \( c \in current_centroids \) do
8. dist = distance \((x_i, c)\)
9. if \( c = bestCentroid \) or || dist < minDist \) then
10. minDist = dist
11. bestCentroid = \( c \)
12. end if
13. end for
14. outputFileLine = <\( (bestCentroid, x_i)\)
15. i += 1
16. end for
17. return outputFileLine

The input dataset of objects is distributed across the mappers. The initial set of centroids is either placed in a common location and accessed by all the mappers or distributed to each mapper. The centroid list has identification for each centroid as key and the centroid itself as the value. Each input object in the subset \( (x_1, x_2, \ldots, x_n) \) is assigned to its closest centroid by the mapper. We have used Euclidean distance to measure proximity of points. The distance between each point and each centroid is measured and the shortest distance is calculated. Then, the object is assigned to its closest centroid. When all the objects are assigned to the centroids, the mapper sends all the input objects and the centroids they are assigned to, to the reducer.
C. Reducer

Algorithm 3 Algorithm for Reducer

Require:
Input: (key, value) where key = best centroid and value = objects assigned to the centroid by the mapper.
Output: (key, value) where key = old centroid and value = new best centroid which is the new centroid value calculated for that best centroid.

1. outputlist = outputlists from mappers
2. v = {}
3. newCentroidList = null
4. for all y : outputlist do
5. centroid = y.key
6. object = y.value
7. v[centroid] += object
8. end for
9. for all centroid ∈ v do
10. newCentroid, sumOfObjects, numOfObjects = null
11. for all object ∈ v[centroid] do
12. sumOfObjects += object
13. numOfObjects += 1
14. end for
15. newCentroid = (sumOfObjects / numOfObjects)
16. newCentroidList = (newCentroidList + newCentroid)
17. end for
18. return newCentroidList

The reducer accepts the key, value pair output from the mappers, loops through the sorted output of the mappers and calculates new centroid values. For each centroid, the reducer calculates a new value based on the objects assigned to it in that iteration. This new centroid list is emitted as the reducer output and sent back to the start-up program.

VII. IMPLEMENTATION OF K-MEANS ALGORITHM USING REGULAR MAPPER AND REDUCER

Once we had generated the data points and the initial set of centroids, we initialized the required statistical parameters, such as, iteration counters and timers. Next, we copied the required files into Hadoop Distributed File System (HDFS). Then, we called the Hadoop[1][2][3] MapReduce iteratively until the set of centroids converged. We started with 500 data points of about 5KB in size and gradually increased the number of points up to 13 million about 125MB in size. We clustered each dataset five times using the five input centroid seed sets. Repeating the experiment five times helped us to reliably monitor the time taken to cluster, ensure we could replicate the result with stability and identify any unexplainable circumstance.

A. Experimental Observations

We analyzed the time taken for various jobs in the Hadoop framework to tune the performance. The table below shows the time taken for various tasks in all iterations of MapReduce[7][11]. Each dataset took 5 iterations to converge.

<table>
<thead>
<tr>
<th>Data Points (in Million)</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average Map Time (in seconds)</td>
<td>14</td>
<td>28</td>
<td>43</td>
<td>56</td>
<td>69</td>
</tr>
<tr>
<td>Shortest Map Time</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>Shuffle Task Time</td>
<td>54</td>
<td>84</td>
<td>105</td>
<td>157</td>
<td>207</td>
</tr>
<tr>
<td>Average Reduce Time</td>
<td>14</td>
<td>27</td>
<td>30</td>
<td>50</td>
<td>61</td>
</tr>
<tr>
<td>Total Time for a MapReduce Cycle</td>
<td>84</td>
<td>148</td>
<td>213</td>
<td>284</td>
<td>340</td>
</tr>
<tr>
<td>Number of Spilled Records (in Millions)</td>
<td>2</td>
<td>4</td>
<td>9</td>
<td>12</td>
<td>15</td>
</tr>
<tr>
<td>Number of Killed Tasks</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Time Taken to Converge (in seconds)</td>
<td>431</td>
<td>742</td>
<td>1060</td>
<td>1414</td>
<td>1728</td>
</tr>
</tbody>
</table>

B. Experimental Inference

We noticed a large number of spilled records and an increase in the time taken to shuffle. There were more number of redundant map and reduce tasks that were started and killed when the data points increased to over 2 million. To reduce the number of duplicated tasks and the shuffling time, and to improve the performance of MapReduce, we added a combiner.

VIII. ALGORITHMS FOR K-MEANS MAPPER AND REDUCER WITH A COMBINER

A. Mapper with Combiner

In the MapReduce[7][11] framework, the output of the map function is written locally to the disk and the reduce function reads the output from the mappers. When the mappers are ready to write, the output is sorted and shuffled. Our initial experiments showed that the time taken for data shuffling and sorting increased as we increased the number of data points from 50000 to 5 million. The time taken to shuffle 50000 points was about 4 seconds which rose up to 30 seconds for 500000 points and 207 seconds for 5 million points. Sorting and shuffling the map output was time intensive. Therefore, we modified the mapping algorithm slightly to combine the map outputs in order to reduce the amount of data that the mappers write and the reducer reads. So we created a combiner that reads the mapper outputs and calculates a local centroid in each mapper. The reducer, then, reads the output from the mappers and calculates global centroid. This improved the performance of Hadoop clustering. Reducing the amount of write in the mapper and the read in the reducer, improved the clustering performance.
We use the same deployment as used for the regular implementation. The difference here is the modified mapper and reducer algorithms.

### A. Experimental Observation

We analyzed the time taken for various jobs in the Hadoop framework to tune the performance. The table below shows the time taken for various tasks in all iterations of MapReduce. Each dataset took 5 iterations to converge.

<table>
<thead>
<tr>
<th>Data Points (in Million)</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average Map Time (in seconds)</td>
<td>14</td>
<td>26</td>
<td>38</td>
<td>50</td>
<td>63</td>
</tr>
<tr>
<td>Shortest Map Time</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>Shuffle Task Time</td>
<td>49</td>
<td>98</td>
<td>129</td>
<td>164</td>
<td>182</td>
</tr>
<tr>
<td>Average Reducer Time</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Total Time for a MapReduce Cycle</td>
<td>63</td>
<td>114</td>
<td>166</td>
<td>209</td>
<td>257</td>
</tr>
<tr>
<td>Number of Spilled Records</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>Number of Killed Tasks</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Time Taken to Converge (in seconds)</td>
<td>345</td>
<td>588</td>
<td>831</td>
<td>1062</td>
<td>1270</td>
</tr>
</tbody>
</table>

### B. Experimental Inference

After recoding the time for each iteration for both regular and improved k-Means. The Improved implementation outperforms the regular implementation by over 350 seconds for 5 million data points. It is evident that the time taken for the improved k-Means implementation is significantly lesser than the regular k-Means Implementation.
X. CONCLUSION AND FUTURE WORK

In this study, we have applied MapReduce technique to k-means clustering algorithm and clustered over 10 million data points. We have shown that k-means algorithm can be successfully parallelised and clustered on commodity hardware. MapReduce can be used for k-means clustering. Our experience also shows that the overheads required for MapReduce algorithm and the intermediate read and write between the mapper and reducer jobs makes it unsuitable for smaller datasets. However, adding a combiner between Map and Reduce jobs improves performance by decreasing the amount of intermediate read/write.

Future research work includes working on reducing redundant map-reduce calls for lower volume data sets. Another area of research is to find a solution to a single point failure in the HDFS cluster, which is the namenode. Applying data mining techniques like expectation maximization and likewise.

ACKNOWLEDGEMENT

I would like to thank every member of the faculty at R V College Of Engineering for their guidance and support, which has helped me, complete this research project successfully.

REFERENCES