Vidyavardhini's College of Engineering & Technology

Department of Artificial Intelligence and Data Science

AY: 2025-26

Class:		**Semester:**	
Course Code:		**Course Name:**	

Name of Student:	BARI ANKIT VINOD
Roll No. :	61
Experiment No.:	**9**
Title of the Experiment:	**To implement a Clustering Algorithm (K-Means or CURE) using Hadoop MapReduce.**
Date of Performance:	
Date of Submission:	

Evaluation

Performance Indicator	Max. Marks	Marks Obtained
Performance	5	
Understanding	5	
Journal work and timely submission	10	
Total	20	

Performance Indicator	Exceed Expectations (EE)	Meet Expectations (ME)	Below Expectations(BE)
Performance	4-5	2-3	1
Understanding	4-5	2-3	1
Journal work and timely submission	8-10	5-8	1-4

Checked by

Name of Faculty :

Signature :

Date :

Experiment No.: 9
AIM:

To implement a Clustering Algorithm (K-Means or CURE) using Hadoop MapReduce.

THEORY:

Clustering is an unsupervised learning technique used to group similar data points together based on their features.
In Big Data applications, traditional clustering algorithms may not work efficiently due to large dataset sizes.
Hence, distributed computing using Hadoop MapReduce helps perform clustering efficiently across multiple nodes.

Two commonly used clustering algorithms are:

K-Means Clustering:

Partitions the dataset into K clusters.

Each cluster has a centroid representing the mean of its points.

Steps:

Initialize K random centroids.

Assign each point to the nearest centroid.

Recalculate centroids.

Repeat until convergence.

CURE (Clustering Using Representatives):

Uses a set of well-scattered points in each cluster as representatives.

Reduces sensitivity to outliers and performs well with large datasets.

When implemented using MapReduce, these algorithms can handle massive datasets efficiently by dividing computation across nodes.

STEPS TO EXECUTE (K-Means with Hadoop MapReduce):
1. Prepare Dataset:

Example data.txt:

1,2
2,3
3,4

8,9
9,10
10,11

2. Mapper Code (KMeansMapper.java):

```
import java.io.IOException;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.Mapper;

public class KMeansMapper extends Mapper<LongWritable, Text, IntWritable, Text> {
   private double[][] centroids = {{2,3}, {9,10}}; // Initial centroids

   public void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException {
      String[] points = value.toString().split(",");
      double x = Double.parseDouble(points[0]);
      double y = Double.parseDouble(points[1]);

      int cluster = 0;
      double minDistance = Double.MAX_VALUE;

      for (int i = 0; i < centroids.length; i++) {
         double distance = Math.sqrt(Math.pow(x - centroids[i][0], 2) + Math.pow(y -
centroids[i][1], 2));
         if (distance < minDistance) {
            minDistance = distance;
            cluster = i;
         }
      }
      context.write(new IntWritable(cluster), new Text(x + "," + y));
   }
}
```

3. Reducer Code (KMeansReducer.java):

```
import java.io.IOException;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.Reducer;

public class KMeansReducer extends Reducer<IntWritable, Text, IntWritable, Text> {
   public void reduce(IntWritable key, Iterable<Text> values, Context context)
      throws IOException, InterruptedException {

      double sumX = 0, sumY = 0;
      int count = 0;
      for (Text val : values) {
         String[] points = val.toString().split(",");
         sumX += Double.parseDouble(points[0]);
         sumY += Double.parseDouble(points[1]);
         count++;
      }
      double newX = sumX / count;
```

```
        double newY = sumY / count;
        context.write(key, new Text(newX + "," + newY));
    }
}
```

4. Run the Program:
hadoop jar KMeans.jar input/data.txt output/

5. Output:
Cluster 0: 2.0,3.0
Cluster 1: 9.0,10.0

OBSERVATION:

The dataset was divided into two distinct clusters:

Cluster 1: Points closer to (2,3)

Cluster 2: Points closer to (9,10)

CONCLUSION:

The K-Means Clustering Algorithm was successfully implemented using the Hadoop MapReduce framework.
This experiment demonstrates how Hadoop can efficiently process and group large datasets by distributing computations across multiple nodes, enabling scalable data clustering.