K. J. Somaiya College of Engineering, Mumbai-77

(Autonomous College Affiliated to University of Mumbai)

Batch: BDA2 Roll No.: 1211061

Experiment / assignment / tutorial No.___5___

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of the Staff In-charge with date

TITLE : Implementation of any one Clustering algorithm using MapReduce.

AIM : To Implementation of any one Clustering algorithm using MapReduce.

Expected Outcome of Experiment :

CO :

CO3 Interpret business modes and scientific computing paradigm and apply software tools for Big data analytics.

Books/ Journals/ Websites referred :

1. Anand Rajaraman and Jeff Ullman "Mining of Massive Datasets", Cambridge University Press,
2. Alex Holmes "Hadoop in Practice", Manning Press, Dreamtech Press.
3. Big data analytics by Radha Shankarmani, M. Vijayalakshmi,Wiley publication

Pre Lab/ Prior Concepts :

K-means clustering, or Lloyd's algorithm, is an iterative, data-partitioning algorithm that assigns n observations to exactly one of k clusters defined by centroids, where k is chosen before the algorithm starts.

The algorithm proceeds as follows:

1. Choose k initial cluster centers (centroid). For example, choose k observations at random (by using 'Start','sample') or use the k-means ++ algorithm for cluster center initialization (the default).
2. Compute point-to-cluster-centroid distances of all observations to each centroid.

3. There are two ways to proceed (specified by OnlinePhase):
- Batch update — Assign each observation to the cluster with the closest centroid.
- Online update — Individually assign observations to a different centroid if the reassignment decreases the sum of the within-cluster, sum-of-squares point-to-cluster-centroid distances.

4. Compute the average of the observations in each cluster to obtain k new centroid locations.
5. Repeat steps 2 through 4 until cluster assignments do not change, or the maximum number of iterations is reached

Conclusion :

Thus we implemented K-mean Clustering algorithm using MapReduce task followed by mapper, reducer and driver classes

Post Lab Questions:

1. Explain BDMO algorithm with an example.

A:

BDMO Approach BDMO = Babcock, Datar, Motwani, O'Callaghan.

The true version of the algorithm involves much more complex structures, which are designed to provide performance guarantees in the worst case. The BDMO Algorithm builds on the methodology for counting ones in a stream that was described in Section 4.6. Here are the key similarities and differences:

• Like that algorithm, the points of the stream are partitioned into, and summarized by, buckets whose sizes are a power of two. Here, the size of a bucket is the number of points it represents, rather than the number of stream elements that are 1.

• As before, the sizes of buckets obey the restriction that there are one or two of each size, up to some limit. However, we do not assume that the sequence of allowable bucket sizes starts with 1. Rather, they are required only to form a sequence where each size is twice the previous size, e.g., 3, 6, 12, 24, ...

• Bucket sizes are again restrained to be nondecreasing as we go back in time. As in Section 4.6, we can conclude that there will be O(log N) buckets.

• The contents of a bucket consists of: 1. The size of the bucket. 2. The timestamp of the bucket, that is, the most recent point that contributes to the bucket. As in Section 4.6, timestamps can be recorded modulo N. 3.

• A collection of records that represent the clusters into which the points of that bucket have been partitioned. These records contain:

(a) The number of points in the cluster.

(b) The centroid or clustroid of the cluster.

(c) Any other parameters necessary to enable us to merge clusters and maintain approximations to the full set of parameters for the merged cluster.

Example : Perhaps the simplest case is where we are using a k-means approach in a Euclidean space. We represent clusters by the count of their points and their centroids. Each bucket has exactly k clusters, so we can pick p = k, or we can pick p larger than k and cluster the p points into k clusters when we create a bucket initially .The best matching between the k clusters of the first bucket and the k clusters of the second. Here, "best" means the matching that minimizes the sum of the distances between the centroids of the matched clusters.

Note that we do not consider merging two clusters from the same bucket, because our assumption is that clusters do not evolve too much between consecutive buckets. Thus, we would expect to find in each of two adjacent buckets a representation of each of the k "true" clusters that exist in the stream. When we decide to merge two clusters, one from each bucket, the number of points in the merged cluster is surely the sum of the numbers of points in the two clusters. The centroid of the merged cluster is the weighted average of the centroids of the two clusters, where the weighting is by the numbers of points in the clusters. That is, if the two clusters have n1 and n2 points, respectively, and have centroids c1 and c2 (the latter are d-dimensional vectors for some d), then the combined cluster has n = n1 + n2 points and has centroid

$$c = \frac{n1c1 + n2c2}{n1 + n2}$$

IMPLEMENTATION:

MAPPER CLASS :

```
import java.io.IOException;

import org.apache.commons.lang.StringUtils;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import java.io.*;

public class K_Means_Mapper extends Mapper<LongWritable,Text, Text,IntWritable>
{
	IntWritable value=new IntWritable();
	Text key=new Text();

	int mean[]=new int[2];
	@Override
	protected void setup(org.apache.hadoop.mapreduce.Mapper.Context context)
			throws IOException, InterruptedException {

		super.setup(context);
		BufferedReader br=new BufferedReader(new
FileReader("/home/kjsce/Desktop/mit/mean"));
		String s1=br.readLine();
		String s2[]=s1.split(" ");
		mean[0]=Integer.parseInt(s2[0]);
		mean[1]=Integer.parseInt(s2[1]);
	}
	public void map(LongWritable ikey, Text ivalue, Context context)
			throws IOException, InterruptedException {
		String line=ivalue.toString();
		String[] tokens=StringUtils.split(line,' ');
		if(Math.abs(mean[0]-Integer.parseInt(tokens[0]))<Math.abs(mean[1]-
Integer.parseInt(tokens[0])))
		{
			Text t1=new Text("1");
			key.set(t1);
			value.set(Integer.parseInt(tokens[0]));
			context.write(key,value);
		}
		else
		{
			Text t2=new Text("2");
			key.set(t2);
			value.set(Integer.parseInt(tokens[0]));
			context.write(key,value);
```

```
                }
        }
}
```

REDUCER CLASS:

```
import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

import java.io.*;
public class K_Means_Reducer
                extends
                Reducer<Text, org.apache.hadoop.io.IntWritable, Text, IntWritable> {


        public void reduce(Text _key, Iterable<IntWritable> values, Context context)
                        throws IOException, InterruptedException {
                FileWriter fwOb = new FileWriter("/home/kjsce/Desktop/mit/mean",
false);
      PrintWriter pw = new PrintWriter(fwOb, false);

                // process values
                int mean=0,c=1;
                for (IntWritable val : values) {
                        String s=""+val;
                        mean+=Integer.parseInt(s);
                        mean=mean/c++;

                        context.write(_key,val);
                }


        }
}
```

DRIVER CLASS :

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
```

```
public class K_Means_driver {

	public static void main(String[] args) throws Exception {
		Configuration conf = new Configuration();
		Job job = Job.getInstance(conf, "JobName");
		job.setJarByClass(K_Means_driver.class);

		job.setMapperClass(K_Means_Mapper.class);

		job.setReducerClass(K_Means_Reducer.class);

		job.setMapOutputKeyClass(Text.class);
		job.setMapOutputValueClass(IntWritable.class);
		job.setOutputKeyClass(Text.class);
		job.setOutputValueClass(IntWritable.class);

		// TODO: specify input and output DIRECTORIES (not files)
		FileInputFormat.setInputPaths(job,new
Path("/home/kjsce/Desktop/mit/input*"));
		FileOutputFormat.setOutputPath(job,new
Path("/home/kjsce/Desktop/mit/output"));

		if (!job.waitForCompletion(true))
			return;
	}

}
```

INPUT 1st PASS MAPPER

2
3
4
8
15
18
20

OUTPUT 1st PASS MAPPER

(1,2)
(1,3)
(1,4)
(2,8)
(2,15)
(2,18)
(2,20)

INPUT 1st PASS REDUCER

1,{2,3,4}

2,{8,15,18,20}

OUTPUT 1st PASS REDUCER

1 2
1 3
1 4
2 8
2 15
2 18
2 20

INPUT 2nd PASS MAPPER

2
3
4
8
15
18
20

OUTPUT 2nd PASS MAPPER

(1,2)
(1,3)
(1,4)
(1,8)
(2,15)
(2,18)
(2,20)

INPUT 2nd PASS REDUCER

1,{2,3,4,8}

2,{15,18,20}

OUTPUT 2nd PASS REDUCER

1 2
1 3
1 4
1 8
2 15
2 18