DATA ANALYSIS USING HADOOP AND MAPREDUCE USING KNN ALGORITHM

CSE 4001: Parallel And Distributive Computing

D1+TD1

SUBMITTED TO: **PROF.MANOOV R**

submitted by

JESSICA SAINI (15BCE0164)
SUBHAM SAHU(15BCB0105)
TANNISHTHA DAS (15BCE2027)

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

NOVEMBER 2017

School of Computer Science and Engineering

DECLARATION

This is to certify that the Parallel and Distributed Computing report entitled "**DATA ANALYSIS USING HADOOP AND MAPREDUCE USING KNN ALGORITHM**" submitted by JESSICA SAINI(15BCE0164), TANNISHTHA DAS (15BCE2027) and SUBHAM SAHU(15BCB0105) to Vellore Institute of Technology, Vellore in partial fulfillment of the requirement for the award of the degree of J-component is a record of bonafide J-component undertaken by them under my supervision. The training fulfills the requirements as per the regulations of this Institute and in my opinion meets the necessary standards for submission. The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Manoov R

Guide

Assistant Professor

ACKNOWLEDGEMENT

The success and final outcome of this project required a lot of guidance and assistance from many people and I am extremely privileged to have got this all along the completion of my project. All that I have done is only due to such supervision and assistance and I would not forget to thank them.

I respect and thank Prof. Manoov R, for providing me an opportunity to do the project work and giving us all support and guidance which made me complete the project duly. I am extremely thankful to him for providing such a nice support and guidance, although he had busy schedule .

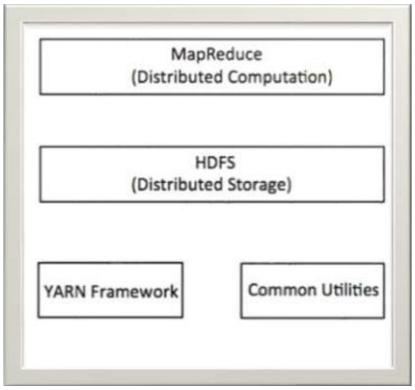
1. INTRODUCTION

Apache Hadoop is an open-source software framework used for distributed storage and processing of dataset of big data using the MapReduce programming model. It consists of computer clusters built from commodity hardware. All the modules in Hadoop are designed with a fundamental assumption that hardware failures are common occurrences and should be automatically handled by the framework.

The core of Apache Hadoop consists of a storage part, known as Hadoop Distributed File System (HDFS), and a processing part which is a MapReduce programming model. Hadoop splits files into large blocks and distributes them across nodes in a cluster.

It then transfers packaged codeinto nodes to process the data in parallel. This approach takes advantage of data locality, where nodes manipulate the data they have access to. This allows the dataset to be processed faster and more efficiently than it would be in a more conventional supercomputer architecture that relies on a parallel file system where computation and data are distributed via high-speed networking.

The base Apache Hadoop framework is composed of the following modules:



- **Hadoop Common** – contains libraries and utilities needed by other Hadoop modules;
- **Hadoop Distributed File System (HDFS)** – a distributed file-system that stores data on commodity machines, providing very high aggregate bandwidth across the cluster;
- **Hadoop YARN** – a platform responsible for managing computing resources in clusters and using them for scheduling users' applications; and
- **Hadoop MapReduce** – an implementation of the MapReduce programming model for large-scale data processing.

1.1 MapReduce

MapReduce is a programming model and an associated implementation for processing and generating big data sets with a parallel, distributed algorithm on a cluster.

A MapReduce program is composed of a **Map()** procedure (method) that performs filtering and sorting (such as sorting students by first name into queues, one queue for each name) and a **Reduce()** method that performs a summary operation (such as counting the number of students in each queue, yielding name frequencies). The "MapReduce System" (also called "infrastructure" or "framework") orchestrates the processing by marshalling the distributed servers, running the various tasks in parallel, managing all communications and data transfers between the various parts of the system, and providing for redundancy and fault tolerance.

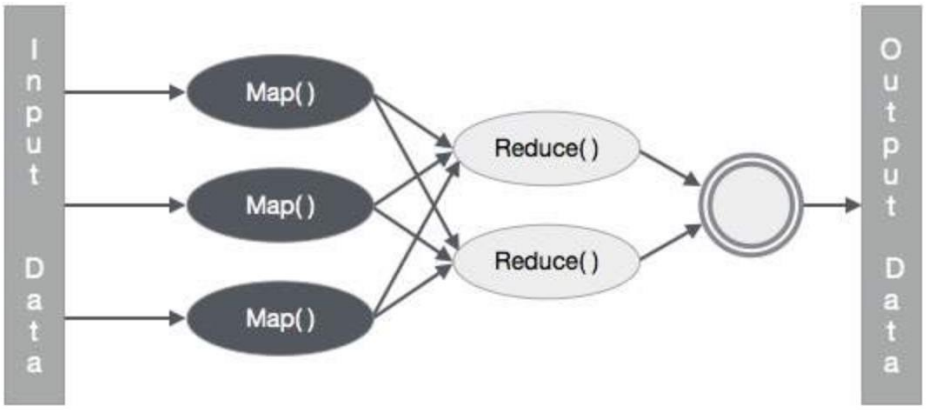


Figure 1: Working of Hadoop

1.1.1 Mapper

Mapper task is the first phase of processing that processes each input record (from RecordReader) and generates an intermediate key-value pair. Hadoop Mapper store intermediate-output on the local disk.

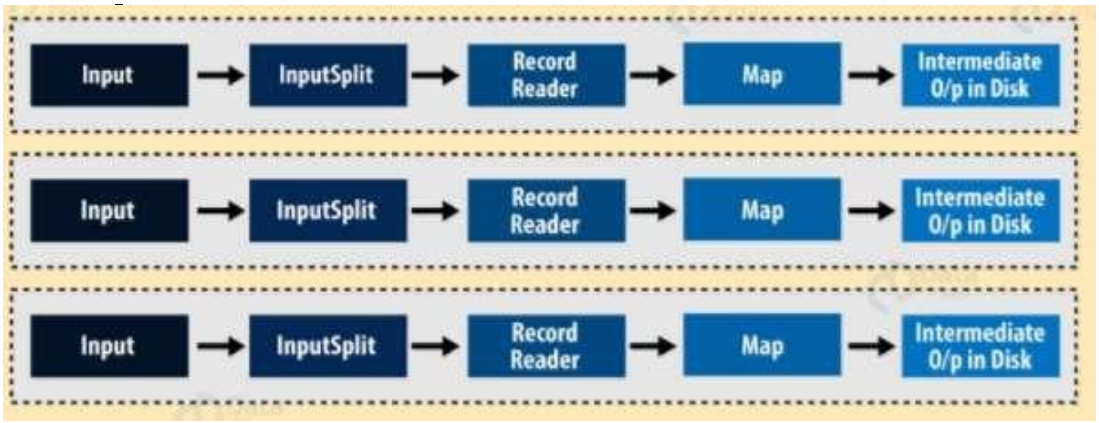


Figure 2: Working of Mapper

Mapper task processes each input record and it generates a new <key, value> pairs. The <key, value> pairs can be completely different from the input pair. In mapper task, the output is the full collection of all these <key, value> pairs. Before writing the output for each mapper task, partitioning of output take place on the basis of the key and then sorting is done. This partitioning specifies that all the values for each key are grouped together.

MapReduce frame generates one map task for each InputSplit (we will discuss it below.) generated by the InputFormat for the job.

Mapper only understands <key, value> pairs of data, so before passing data to the mapper, data should be first converted into <key, value> pairs.

1.1.2 Combiner:

Hadoop Combiner is also known as "Mini-Reducer" that summarizes the Mapper output record with the same Key before passing to the reducer.

On a large dataset when we run MapReduce job, so large chunks of intermediate data is generated by the Mapper and this intermediate data is passed on the Reducer for further processing, which leads to enormous network congestion. MapReduce framework provides a function known as Combiner that plays a key role in reducing network congestion.

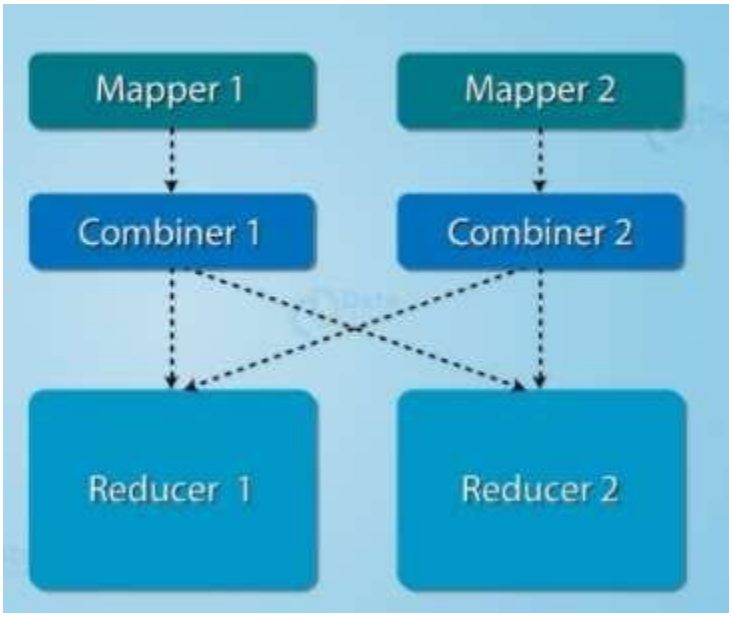


Figure 3: Working of Combiner

Advantages of Combiner in MapReduce

- Combiner reduces the time taken for data transfer between mapper and reducer.
- It decreases the amount of data that needed to be processed by the reducer.
- The Combiner improves the overall performance of the reducer.

Disadvantages of Combiner in MapReduce:

- MapReduce jobs cannot depend on the Hadoop combiner execution because there is no guarantee in its execution.
- In the local filesystem, the key-value pairs are stored in the Hadoop and run the combiner later which will cause expensive disk IO.

1.1.3 Reducer

Reducer takes the output of the Mapper (intermediate key-value pair) process each of them to generate the output. The output of the reducer is the final output, which is stored in HDFS.

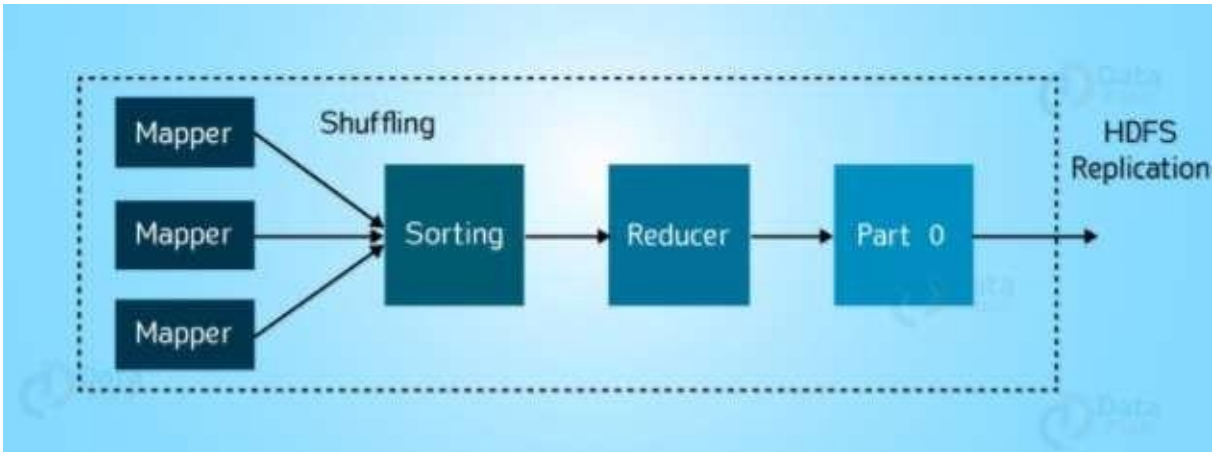


Figure 4: Working of Reducer

The Reducer process the output of the mapper. After processing the data, it produces a new set of output. At last HDFS stores this output data.

Reducer takes the output of the Mapper (intermediate key-value pair) process each of them to generate the output. The output of the reducer is the final output, which is stored in HDFS. Usually, in the Hadoop Reducer, we do aggregation or summation sort of computation

Hadoop Reducer takes a set of an intermediate key-value pair produced by the mapper as the input and runs a Reducer function on each of them. One can aggregate, filter, and combine this data (key, value) in several ways for a wide range of processing. Reducer first processes the intermediate values for key generated by the map function and then generates the output (zero or more key-value pair).

One-one mapping takes place between keys and reducers. Reducers run in parallel since they are independent of one another. The user decides the number of reducers. By default, number of reducers is 1.

1.1.4 Phases of Reducer:

- *Shuffle Phase*

In this phase, the sorted output from the mapper is the input to the Reducer. In Shuffle phase, with the help of HTTP, the framework fetches the relevant partition of the output of all the mappers.

- *Sort Phase*

In this phase, the input from different mappers is again sorted based on the similar keys in different Mappers. The shuffle and sort phases occur concurrently.

- *Reduce phase:*

In this phase, after shuffling and sorting, reduce task aggregates the key-value pairs. The OutputCollector.collect() method, writes the output of the reduce task to the Filesystem. Reducer output is not sorted.

- *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). Secondly, reduce task, which takes the output from a map as an input and combines those data tuples into a smaller set of tuples. As the sequence of the name MapReduce implies, the reduce task is always performed after the map job.

The major advantage of MapReduce is that it is easy to scale data processing over multiple computing nodes. Under the MapReduce model, the data processing primitives are called mappers and reducers. Decomposing a data processing application into mappers and reducers is sometimes nontrivial. But, once we write an application in the MapReduce form, scaling the application to run over hundreds, thousands, or even tens of thousands of machines in a cluster is

merely a configuration change. This simple scalability is what has attracted many programmers to use the MapReduce model.

- Generally MapReduce paradigm is based on sending the computer to where the data resides!
- MapReduce program executes in three stages, namely map stage, shuffle stage, and reduce stage.
    - **Map stage**: The map or mapper's job is to process the input data. Generally the input data is in the form of file or directory and is stored in the Hadoop file system (HDFS). The input file is passed to the mapper function line by line. The mapper processes the data and creates several small chunks of data.
    - **Reduce stage** : This stage is the combination of the **Shuffle** stage and the **Reduce** stage. The Reducer's job is to process the data that comes from the mapper. After processing, it produces a new set of output, which will be stored in the HDFS.
- During a MapReduce job, Hadoop sends the Map and Reduce tasks to the appropriate servers in the cluster.
- The framework manages all the details of data-passing such as issuing tasks, verifying task completion, and copying data around the cluster between the nodes.
- Most of the computing takes place on nodes with data on local disks that reduces the network traffic.
- After completion of the given tasks, the cluster collects and reduces the data to form an appropriate result, and sends it back to the Hadoop server.

Inputs and outputs:

The MapReduce framework operates on <key, value> pairs, that is, the framework views the input to the job as a set of <key, value> pairs and produces a set of <key, value> pairs as the output of the job, conceivably of different types.

The key and the value classes should be in serialized manner by the framework and hence, need to implement the Writable interface. Additionally, the key classes have to implement

	Input	Output
Map	<k1, v1>	list (<k2, v2>)
Reduce	<k2, list(v2)>	list (<k3, v3>)

Figure 5: Key-Value pair

Writable-Comparable interface to facilitate sorting by the framework. Input and Output types of a MapReduce job: (Input)

<k1, v1> -> map -> <k2, v2>-> reduce -> <k3, v3>(Output)

2. k-NN

k-NN is a non-parametric lazy learning algorithm. Being a non-parametric algorithm it does not make any assumptions on the underlying data distribution. This is a major advantage because majority of the practical data does not obey theoretical assumptions made and this is where non-parametric algorithms like kNN come to the rescue. kNN is also a lazy algorithm this implies that it does not use the training data points to do any generalization. So, the training phase is pretty fast. Lack of generalization means that kNN keeps all the training data. kNN makes decision based on the entire training data set.

The k-Nearest Neighbor Algorithm finds applications in some of the fascinating fields like Nearest Neighbor based Content Retrieval, Gene Expressions, Protein-Protein interaction and 3-D Structure predictions are to name a few.

Requisites for k-Nearest Neighbor Algorithm

- KNN assumes that the data is in a feature space. More exactly, the data points are in a metric space.
- The data can be scalars or possibly even multidimensional vectors. Since the points are in feature space, they have a notion of distance – This need not necessarily be Euclidean distance although it is one of the commonly used methods.
- The kNN uses training data as reference to classify the new data points collectively called testing dataset
- Each of the training data consists of a set of vectors and class label associated with each vector. In the simplest case, it will be either + or – (for positive or negative classes). But kNN, can work equally well with arbitrary number of classes.
- We are also given a single number ‘k’. This number decides how many neighbors (where neighbors is defined based on the distance metric) influence the classification. This is usually an odd number if the number of classes is 2. If k=1, then the algorithm is simply called the nearest neighbor algorithm.

- **The k-Nearest Neighbor Algorithm involves two phases.**

- The Training Phase
- The Testing Phase

1) The Training Phase

kNN Algorithm does not explicitly require any training phase for the data to be classified. The training phase usually involves storing the data vector co-ordinates along with the class label. The class label in general is used as an identifier for the data vector. This is used to classify data vectors during the testing phase

2) The Testing Phase

Given data points for testing, our aim is to find the class label for the new point. The algorithm is discussed for k=1 or 1 Nearest Neighbor rule and then extended for k=k or k- Nearest Neighbor rule.

2.1 K-Nearest Algorithm

Generalising the k-Nearest Neighbor Testing Phase

1. Determine the value of 'k' (input)

2. Prepare the training data set by storing the coordinates and class labels of the data points.

3. Load the data point from the testing data set.

4. Conduct a majority vote amongst the 'k' closest neighbors of the testing data point from the training data set based on a distance metric.

5. Assign the class label of the majority vote winner to the new data point from the testing data set.

6. Repeat this until all the Data points in the testing phase are classified.

2.3 ALGORITHMIC DESIGN OF THE K-NEAREST NEIGHBOR (KNN) TECHNIQUE IN THE MAPREDUCE PARADIGM

Algorithm 1 Mapper design for k-NN

```
0: procedure K-NN MAPDESIGN
0:    Create list to maintain data points in the testing data-set
0:    testList = new testList
0:    Load file containing testing data-set
0:    load testFile
0:    Update list with data points from file
0:    testList <= testFile
0:    Open file containing training data set
0:    OPEN trainFile
0:    Load training data points one at a time and compute
      distance with every testing data point
0:    distance (trainData, testData)
0:    Write the distance of test data points from all the
      training data points with their respective class labels in
      ascending order of distances
0:    testFile <= testData(dist, label)
0:    Call Reducer
0: end procedure=0
```

Figure 6: Mapper Design for KNN

Algorithm 2 Reducer design for k-NN

0: **procedure** K-NN REDUCEDESIGN
0: Load the value of 'k'
0: Load testFile
0: $OPEN\ testFile$
0: Load test data points one at a time
0: $READ\ testDataPoint$
0: Initialize counters for all class labels
0: $SET\ counters\ to\ ZERO$
0: Look through top 'k' distance for the respective test data point and increment the corresponding class label counter
0: $for\ i = 0\ to\ k$
0: $COUNTER_i++$
0: Assign the class label with the highest count for the testDataPoint in question
0: $testDataPoint = classLabel(COUNTER_{max})$
0: Update output file with classified test data point
0: $outFile = outFile + testDataPoint$
0: **end procedure**=0

Figure 7: Reducer design for Knn

Algorithm 3 Implementing kNN Function

0: **procedure** KNN FUNCTION
0: Read the value of 'k'
0: $SET\ 'k'$
0: Set paths for training and testing data directories
0: $SET\ trainFile$
0: $SET\ testFile$
0: Create new JOB
0: SET MAPPER to map class defined
0: SET REDUCER to reduce class define
0: Set paths for output directory
0: SUBMIT JOB
0: **end procedure**=0

Figure 8: Implementing Knn Function

3. CODE

3.1 Explanation of the code

- Set the mapper, template, reducer, outputkey, outputvalue class.
- Set the input and the output path.
- Open the file containing the dataset.
- Read the file line by line. Everytime the program encounters a null character print a new line.
- Print the test cases before sorting.
- Generate tokens to for different class and calculate the distance with every testing datapoint in the Mapper Class.

- Call the Reducer.
- Use KNN Algorithm to loop through k distance for the respective testdata and increment the label counter. Assign the class label with the highest count for the test datapoint.
- Update the file with classified test point data.
- Set paths for output directory and submit the job.
- Print the test cases after sorting.

//The code mentioned below is based on the algorithms mentioned in Figure 6, Figure 7 and Figure 8. The comments are written in green.

```
import java.io.IOException;
import org.apache.hadoop.conf.Configuration; //Importing the libraries
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
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 template containing the major code to implement the algorithm
public class Template  {

public static void main(String args[]) throws Exception {

Configuration conf = new Configuration();

Job job = Job.getInstance(conf, "Template");

job.setJarByClass(Template.class);
job.setMapperClass(TmpltMapper.class); //Set the Mapper Class
job.setCombinerClass(TmpltReducer.class); //Set the Template Class
job.setReducerClass(TmpltReducer.class); //Set the Reducer Class
 job.setOutputKeyClass(Text.class); //Set the Outputkey Class
job.setOutputValueClass(IntWritable.class); //Set the Output value Class
FileInputFormat.addInputPath(job, new Path(args[0])); //Add Input Path
FileOutputFormat.setOutputPath(job, new Path(args[1])); //Set the Output Path
//wait until the task setting the path is completed
job.waitForCompletion(true);

Path pt = new Path(args[1] + "/part-r-00000");
FileSystem fs = FileSystem.get(new Configuration());
//Read the test cases.
```

```
BufferedReader br = new BufferedReader(new InputStreamReader(
fs.open(pt))); //Open the file containing the dataset
String line; //Logic to read the file line by line
Vector<String> ss = new Vector<String>();
line = br.readLine();
//ss.add(line);
while (line != null) {
 System.out.println(line); //Add a line break everytime null character is read.
 ss.add(line);
line = br.readLine();
}
double[] dst = new double[ss.size()];
String[] ft = new String[ss.size()];
for (int i = 0; i < ss.size(); i++) {
String gg=ss.get(i).trim();
StringTokenizer st = new StringTokenizer(gg);
st.nextToken();
dst[i] = Double.parseDouble(st.nextToken());
ft[i] = st.nextToken();
}
//Print the test cases before sorting
System.out.println(" before sorted ");
for (int i = 0; i < dst.length; i++) {
System.out.println(dst[i] + " " + ft[i]);
}
double temp=0;
String temps = null;
int n=dst.length;
for (int i = 0; i < n; i++)
      {
         for (int j = i + 1; j < n; j++)
         {
            if (dst[i] > dst[j])
            {
               temp = dst[i];
               temps = ft[i];
               dst[i] = dst[j];
               ft[i] = ft[j];
               dst[j] = temp;
               ft[j] = temps;
            }
         }
      }
System.out.println();
```

```
System.out.println("After sorted");
//Print the test cases after sorting
for (int i = 0; i < dst.length; i++) {
System.out.println(dst[i] + " " + ft[i]);
}
int k = 3;
int v, f, p;
v = 0;
f = 0;
p = 0;
// Make sure the input arguments are legal
for (int i = 0; i < k; i++) {
if (ft[i].equals("fr"))
f++;
if (ft[i].equals("vg"))
v++;
if (ft[i].equals("pr"))
p++;
}
System.out.println();
int bigst = v;
String dsc = "vegitable";
if (bigst < f) {
bigst = f;
dsc = "fruit";
}
if (bigst < p) {
bigst = p;
dsc = "protien";
}
System.out.println("Classified is " + dsc + " cos veg: " + v+ " fruit: " + f + " protien: " + p);
}
//Creating a static class TmpltMapper whose logic is given in the figures mentioned above.
public static class TmpltMapper extends
Mapper<LongWritable, Text, Text, IntWritable> { //The Mapper Class
@Override
public void map(LongWritable key, Text value, Mapper.Context context)
//Code to map key and test value
throws IOException, InterruptedException {
 String line = value.toString(); //Generation of tokens
StringTokenizer tokenizer = new StringTokenizer(line, ",");
int sw = Integer.parseInt(tokenizer.nextToken());
int cr = Integer.parseInt(tokenizer.nextToken());
String ft = tokenizer.nextToken();
```

```
//Calculate the distance of the incoming data
double dist = Math.sqrt(Math.pow((6 - sw), 2)+ Math.pow((4 - cr), 2));
String rs = dist + "   " + ft;
context.write(new Text(" MKEY"),new Text(rs)); }
}
//Reducer Class
public static class TmpltReducer extends
Reducer<Text, IntWritable, Text, IntWritable> {
@Override
// Use KNN Algorithm to loop through k distance for the respective testdata.
//Assign the class label with the highest count for the test datapoint
public void reduce(Text key, Iterable<IntWritable> values,
Context context) throws IOException, InterruptedException {
//Output a file containing labels for TestRecords
    for (Text value : values) {
context.write(key, value);
} }

}
```

4. PROCEDURE AND IMPLEMENTATION

The following are the vital commands to implement the analysis of data using Hadoop and MapReduce.

- **Sudo su :**
  sudo is intended to run a solitary charge with root benefits. Be that as it may, not at all like su it prompts you for the watchword of the present client. This client must be in the sudoers document (or a gathering that is in the sudoers record).

- **ssh localhost**
  Setting up a protected association. The charge implies that an association is directed to the possess machine, to the present client.

- **hadoop namenode -format**
  Formatting the name node to start a new environment. NameNode is the centerpiece of HDFS. NameNode is also known as the Master. NameNode only stores the metadata of HDFS ñ the directory tree of all files in the file system, and tracks the files across the cluster. NameNode does not store the actual data or the dataset. The data itself is actually stored in the DataNodes.

- **start-all.sh**
  Start all the nodes. The $HADOOP_INSTALL/hadoop/bin directory contains some scripts used to launch Hadoop DFS and Hadoop Map/Reduce daemons. These are: start-dfs.sh - Starts the Hadoop DFS daemons, the namenode, datanodes, the jobtracker and tasktrackers.

- **Jps**
  Shows all the working nodes. The jps tool lists the instrumented HotSpot Java Virtual Machines (JVMs) on the target system. The tool is limited to reporting information on JVMs for which it has the access permissions.
  If jps is run without specifying a adoop, it will look for instrumented JVMs on the local host. If started with a adoop, it will look for JVMs on the indicated host, using the specified protocol and port. A jstatd process is assumed to be running on the target host.
  The list of JVMs produced by the jps command may be limited by the permissions granted to the principal running the command. The command will only list the JVMs for which the principle has access rights as determined by operating system specific access control mechanisms.

- **hadoop dfsadmin -report**
  Describes the health and other required aspects.

- **hadoop dfs -lsr /**
  Shows all the files in the hdfs. HDFS supports highly-available (HA) namenode services and wire compatibility. These two capabilities make it feasible to upgrade HDFS without incurring HDFS downtime. In order to upgrade a HDFS cluster without downtime, the cluster must be setup with HA.
  If there is any new feature which is enabled in new software release, may not work with old software release after upgrade. In such cases upgrade should be done by following steps.
    - Disable new feature.
    - Upgrade the cluster.
    - Enable the new feature.

- **hadoop dfs -mkdir test**
  Makes a new folder. In computer hardware and software development, testing is used at key checkpoints in the overall process to determine whether objectives are being met. For example, in software development, product objectives are sometimes tested by product user representatives. The mkdir command is is used to create new directories.
  A directory, referred to as a folder in some operating systems, appears to the user as a container for other directories and files.

- **hadoop dfs -put abc.txt /user/test**
  Adds the test file to generate the training data. Copy single src, or multiple srcs from local file system to the destination file system. Also reads input from stdin and writes to destination file system.

- **hadoop jar Knn.jar knn /user/test/abc.txt /user/test/out1**
  Generates training data. Runs a jar file. Users can bundle their Map Reduce code in a jar file and execute it using this command.

- **hadoop dfs -cat /user/test/out1/part-r-00000**
  Shows the output file.

- **hadoop dfs -rmr /**
  Formats the hdfs. Recursive version of delete. If the ñskipTrash option is specified, the trash, if enabled, will be bypassed and the specified file(s) deleted immediately. This can be useful when it is necessary to delete files from an over-quota directory.

- **stop-all.sh**
  Stops all the nodes. Used to stop hadoop daemons all at once. Issuing it on the master machine will start/stop the daemons on all the nodes of a cluster. Deprecated as you have already noticed.

- **Exit**
  exit is a command used in many operating system command line shells and scripting languages. The command causes the shell or program to terminate. If performed within an interactive command shell, the user is logged out of their current session, and/or userís current console or terminal connection is disconnected.

5. OUTPUT

Output Screenshots

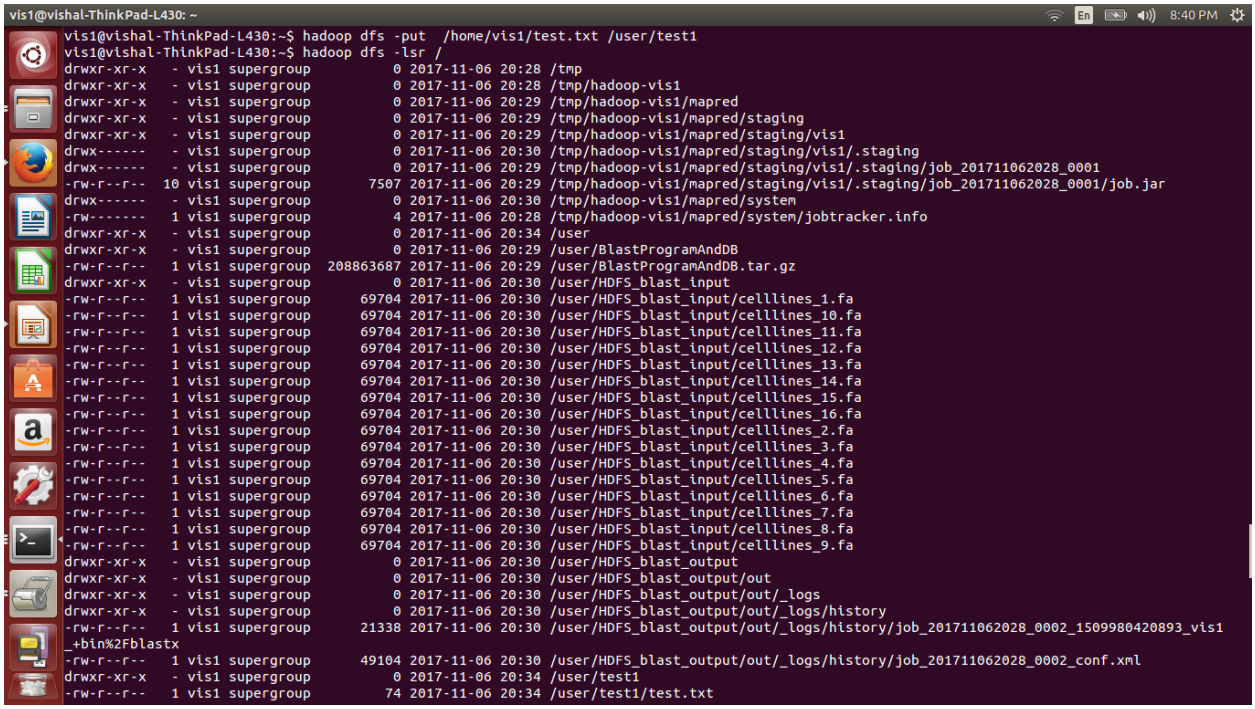


Figure 9: ScreenShot 1

- ScreenShot1: hadoop dfs -lsr /

Shows all the files in the hdfs. HDFS supports highly-available (HA) namenode services and wire compatibility.

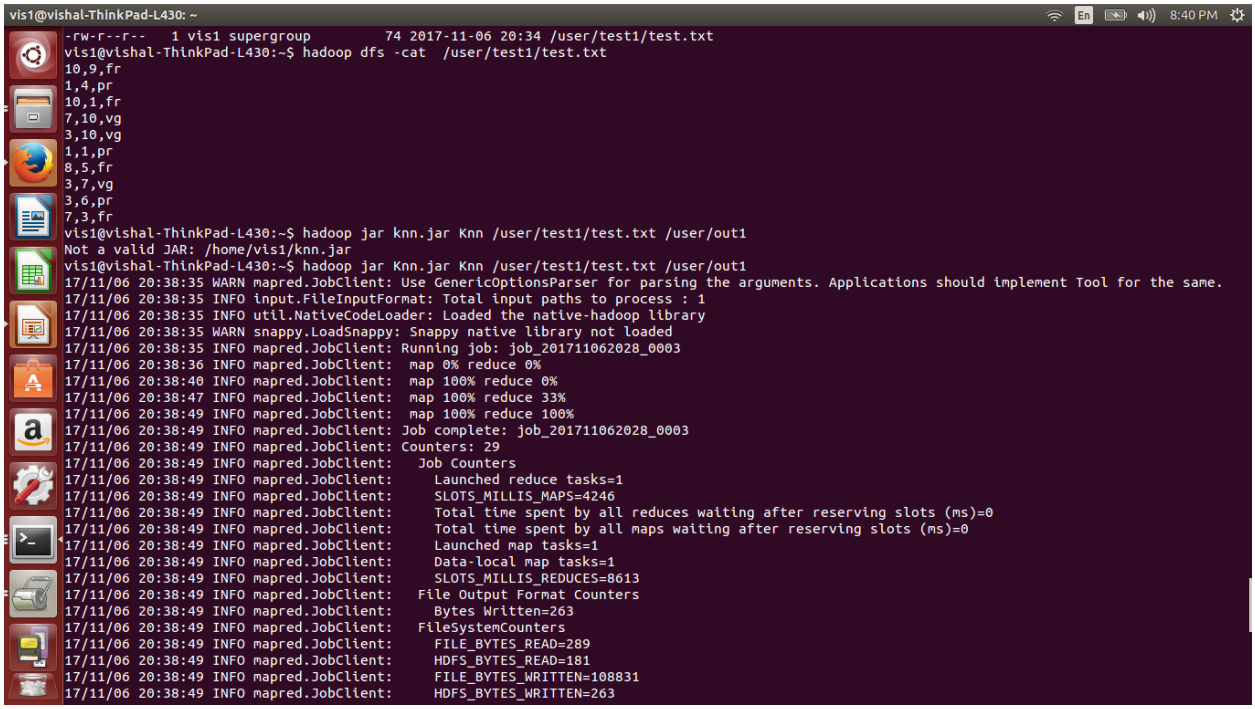


Figure 10: ScreenShot2

- **ScreenShot2: hadoop jar Knn.jar knn /user/test/abc.txt /user/test/out1**

Generates training data. Runs a jar file. Users can bundle their Map Reduce code in a jar file and execute it using this command. Maps and reduces the data. The counters value is 29.

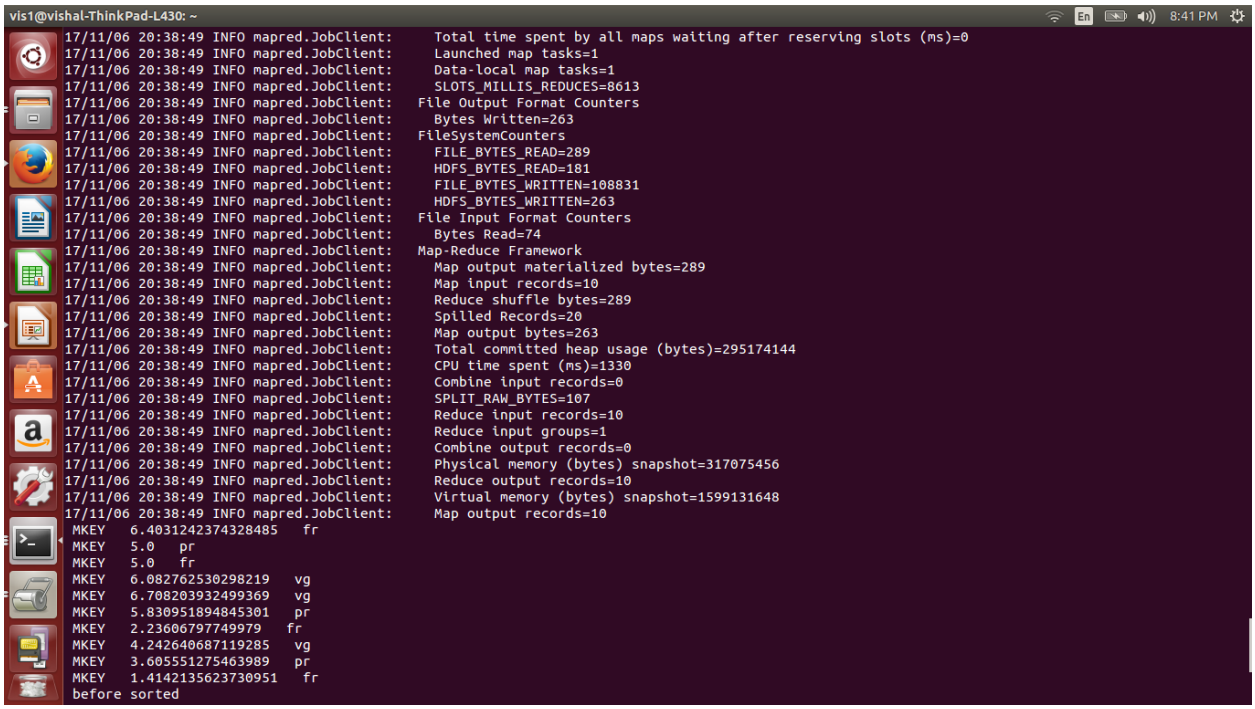


Figure 11: ScreenShot 3

Screenshot 3: The dataset is loaded and Mkey value for each data is generated

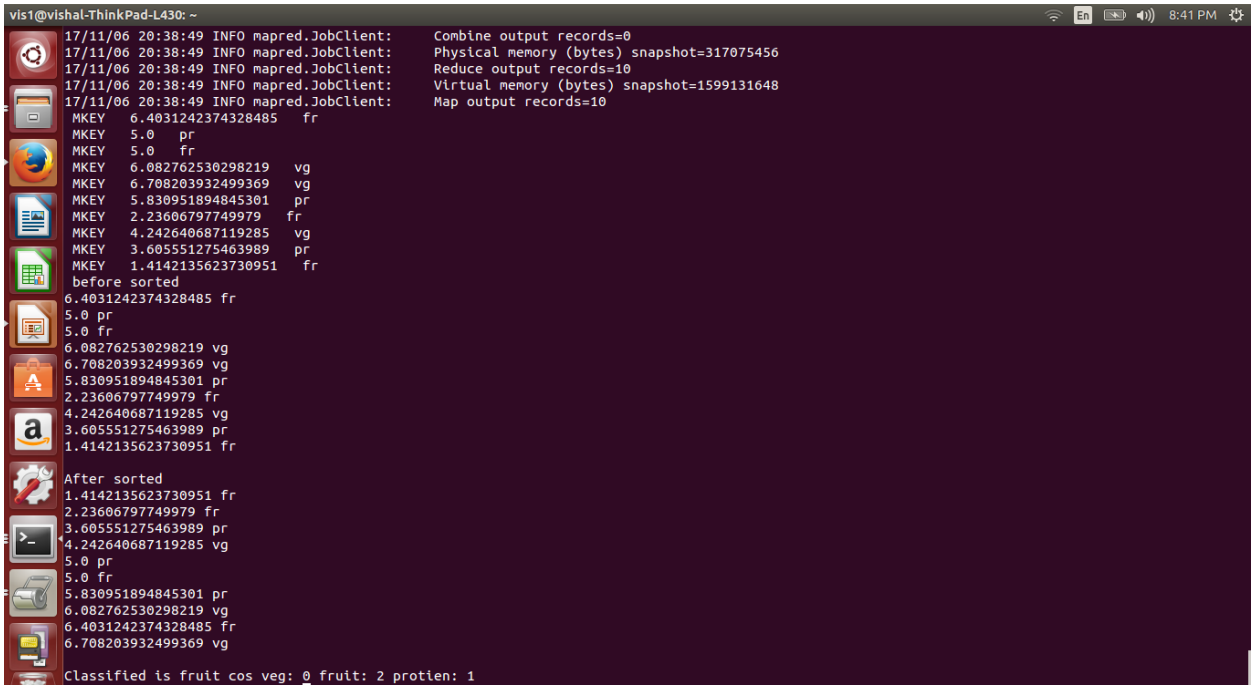


Figure 12: Screenshot4

Screenshot 4: The Reducer function is called. The dataset is sorted and classes for each are determined using KNN Algorithm.

6.CONCLUSION

In our project we implemented the k- Nearest Neighbor technique in Hadoop MapReduce environment. This was done to remove the limitation of computational capability by having a cluster of systems working together. This not only eliminates the limitation of computational demands but also speeds up the processing time by having more than one computer interconnected over a network working on the given task. An unsorted dataset was taken as the input. Upon calling Mapper and reducer functions, it was then sorted using Knn algorithm.

7. REFERENCES

[1] Agarwal R. and Srikant R., "Fast algorithms for mining association rules", VLDB'94, Chile, pp. 487–499, 1994.

[2] Anderson C., Domingos P., Weld D. S., "Relational Markov Models and their Application to Adaptive Web Navigation", Proceedings of the 8th ACM SIGKDD Conference, Canada, August 2002.

[3] Bamshad Mobasher, Robert Cooley, Jaideep Srivastava," Creating Adaptive Web Sites Through Usage-Based Clustering of URLs", proceedings of the 1999 workshop on knowledge and data engineering, pp 19, 1999.

[4] Bruha I., "From machine learning to knowledge discovery: Survey of preprocessing and postprocessing" Intelligent Data Analysis, Vol. 4, pp. 363-374, 2000.

[5] Buchner A. and Mulvenna M. D., "Discovering Internet Marketing Intelligence through Online Analytical Web Usage Mining", Proceedings of the ACM SIGMOD, Intl.Conf. on Management of Data (SIGMOD'99), pp. 54– 61, 1999.

[6] Burges C., "A tutorial on support vector machines for pattern recognition", Data Mining and Knowledge Discovery, Vol. 2, pp. 1-47, 1998.

[7] Cheng D., Kannan R., Vempala S. and Wang G., "A divide-and-merge methodology for clustering", ACM SIGMOD, pp. 196–212, 2005.

[8] Cooley R. , Mobasher B., and Srivastava J. , "Data Preparation for Mining World Wide Web Browsing Patterns", Knowledge and Information Systems, vol. 1(1), pp. 5–32, 1999.

[9] Cristianini N. and Shawe-Taylor J., "An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods", Cambridge University Press, Cambridge, 2000.

[10] Deshpande M., Karypis G., "Selective Markov Models for Predicting Web-Page Accesses", Proceedings of the 1st SIAM International Conference on Data Mining, 2004.

[11] Dhruv Gupta, Mark Digiovanni, Hiro Narita, and Ken Goldberg, "Jester 2.0 : Evaluation of a New Linear Time Collaborative Filtering Algorithm", SIGIR „99 Berkley, CA, USA ,ACM, 1999.

[12] Domingos P. and Pazzani M. , "On the optimality of the simple Bayesian classifier under zero-one loss". Machine Learning, pp.103-130, 1997.

[13] https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.pdf

[14] IEEE Journal: "Big data management processing with Hadoop MapReduce and spark technology: A comparison".

[15] https://hadoop.apache.org/docs/r1.2.1/hdfs_design