All the below MapReduce Programs for processing structured and unstructured data.

We have two MapReduce APIs available with MapReduce Framework; We can use any one of them. Old API supports all libraries but New API has yet to implement some of the library like chain mappers and reducers etc. Majority of the below programs are written in New API, a few programs are written in Old API.

All the class from New API having prefixed with;
org.apache.hadoop.mapreduce.

All the class from Old API having prefixed with;

org.apache.hadoop.mapred.

For every program, we have three classes Main Class (name of the class which having driver method i.e. main method), Mapper class to Implement the Map side Business Logic, and Reducer class to implement the Reducer side business Logic.

We can write MapReduce program in two ways.

1. All the three classes together
2. Three Individual classes

Approach One:

```
Main class
{
      Driver Method
      Mapper Class
      Reducer Class
}
```

Approach Two:

```
Main Class
{
      Driver Method
}
Mapper Class
```

Reducer Class

MapReduce Template for New API:

```
package twok.hadoop.mapreduce;

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;
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.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class NewAPIMRTemplate {

     public static class MyMapper extends Mapper<mapkeyin, mapvaluein,
mapkeyout, mapvalueout>
     {
          public void map( mapkeyin key, mapvaluein value, Context
context) throws IOException, InterruptedException
          {
               //Map side Business Logic
          }
```

```
    }

    public static class MyReducer extends Reducer<reducekeyin,
reducevaluein, reducekeyout, reducevalueout>

    {

        public void reduce( reducekeyin key, Iterable<reducevaluein>
values, Context context) throws IOException, InterruptedException

        {

            //Reduce side Business Logic

        }

    }

    public static void main(String args[]) throws IOException,
InterruptedException, ClassNotFoundException

    {

        Configuration conf = new Configuration();

        Job job = new Job(conf, "Finding the total stock vollumes");

        job.setJarByClass(NewAPIMRTemplate.class);

        job.setMapperClass(MyMapper.class);

        job.setReducerClass(MyReducer.class);

        job.setMapOutputKeyClass(mapkeyout.class);

        job.setMapOutputValueClass(mapvalueout.class);

        job.setOutputKeyClass(reducekeyout.class);

        job.setOutputValueClass(reducevalueout.class);
```

```
            job.setInputFormatClass(inputformat.class);
            job.setOutputFormatClass(outputformat.class);

            FileInputFormat.addInputPath(job, new Path(args[0]));
            FileOutputFormat.setOutputPath(job, new Path(args[1]));

            System.exit(job.waitForCompletion(true) ? 0 :1);

      }

}
```

MapReduce Template for Old API:

```
package twok.hadoop.mapreduce;

import java.io.IOException;
import java.util.Iterator;

import org.apache.hadoop.conf.Configuration;
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.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.MapReduceBase;
```

```
import org.apache.hadoop.mapred.Mapper;

import org.apache.hadoop.mapred.OutputCollector;

import org.apache.hadoop.mapred.Reducer;

import org.apache.hadoop.mapred.Reporter;

import org.apache.hadoop.mapred.TextInputFormat;

import org.apache.hadoop.mapred.TextOutputFormat;

import org.apache.hadoop.util.GenericOptionsParser;



public class OldAPIMRTemplate {


     public static class MyMapper extends MapReduceBase implements
Mapper<mapkeyin, mapvaluein, mapkeyout, mapvalueout>

     {

          public void map(mapkeyin key, mapvaluein value,

                    OutputCollector<mapkeyout, mapvalueout> collector,
Reporter reporter)

                    throws IOException {

               //Map side Business Logic

          }

     }


     public static class MyReducer extends MapReduceBase implements
Reducer<reducekeyin, reducevaluein, reducekeyout, reducevalueout>

     {


          @Override

          public void reduce(reducekeyin key, Iterator<reducevaluein>
values,

                    OutputCollector<reducekeyout, reducevalueout>
collector, Reporter reporter)

                    throws IOException {
```

```
            //Reduce side Business Logic
        }

    }

    public static void main(String args[]) throws IOException,
InterruptedException, ClassNotFoundException
    {
        Configuration conf = new Configuration();
        String otherArgs[] = new GenericOptionsParser(conf,
args).getRemainingArgs();

        JobConf job = new JobConf(conf);
        job.setJobName("Name of the Job");

        job.setJarByClass(OldAPIMRTemplate.class);

        job.setMapperClass(MyMapper.class);
        job.setReducerClass(MyReducer.class);

        job.setMapOutputKeyClass(mapkeyout.class);
        job.setMapOutputValueClass(mapvalueout.class);

        job.setOutputKeyClass(reducekeyout.class);
        job.setOutputValueClass(reducevalueout.class);

        job.setInputFormat(inputformat.class);
        job.setOutputFormat(outputformat.class);

        FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
        FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
```

```
            JobClient.runJob(job);
        }
}
```

Note: mapkeyout == reducekeyin

mapvalueout == reducevaluein

All these keyin, keyout, valuein, valueout are hadoop data types like IntWritable, LongWritable, Text etc. These are available in org.apache.hadoop.io package.

****

We categorized all the programs into a few groups based on their purpose.

- Data Aggregation
- Indexing
- Searching
- Clustering
- Text Analysis

Data Aggregation:

In this category, we are going to do basic data aggregation problems by using simple statistics.

Input Data set is New York Stock Exchange sample data.

This data set is tab separated one, having 9 columns. The columns are exchange name, stock name, date, open, high, low, close, volume, adj_close.

Data Loading into HDFS:

create the directory in HDFS:

```
cmd> hadoop fs -mkdir /markets
cmd> hadoop fs -put stocks /markets/stocks
cmd> hadoop fs -ls /markets
```

MapReduce Job One:

Objective: Finding the Aggregated Volume of every stock for the entire data set.

Using New API:

Java Source Code:

```
package twok.hadoop.mapreduce;

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
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.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
```

```
public class StockTotalVolume {


     public static class MyMapper extends Mapper<LongWritable, Text,
Text, LongWritable>{

          public void map(LongWritable key, Text value, Context context)
throws InterruptedException, IOException

          {

               String line = value.toString();

               String[] parts = line.split("\\t");

               if(parts.length == 9)

               {

                    String stockName = parts[1];

                    long volume = Long.valueOf(parts[7]);

                    context.write(new Text(stockName), new
LongWritable(volume));

               }

          }

     }


     public static class MyReducer extends Reducer<Text, LongWritable,
Text, LongWritable>

     {

          public void reduce(Text key, Iterable<LongWritable> values,
Context context) throws IOException, InterruptedException

          {

               long sum = 0;

               for(LongWritable value : values)

               {

                    sum = sum + value.get();

               }

               context.write(key, new LongWritable(sum));
```

```
            }
      }

      public static void main(String args[]) throws IOException,
InterruptedException, ClassNotFoundException
      {
            Configuration conf = new Configuration();

            Job job = new Job(conf, "Finding the stock Volume");
            job.setJarByClass(StockTotalVolume.class);

            job.setMapperClass(MyMapper.class);
            job.setCombinerClass(MyReducer.class);
            job.setReducerClass(MyReducer.class);

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

            job.setInputFormatClass(TextInputFormat.class);
            job.setOutputFormatClass(TextOutputFormat.class);

            FileInputFormat.addInputPath(job, new Path(args[0]));
            FileOutputFormat.setOutputPath(job, new Path(args[1]));

            System.exit(job.waitForCompletion(true) ? 0 : 1);
      }
}
```

Job Submission to Hadoop OR Usage:

cmd> hadoop jar stocktotalvolume.jar
twok.hadoop.mapreduce.StockTotalVolume /markets/stocks /markets/volume

Using Old API:

Java Source Code:

```
package twok.hadoop.mapreduce;

import java.io.IOException;
import java.util.Iterator;

import org.apache.hadoop.conf.Configuration;
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.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;
import org.apache.hadoop.mapred.TextInputFormat;
import org.apache.hadoop.mapred.TextOutputFormat;
```

```
import org.apache.hadoop.util.GenericOptionsParser;



public class OldAPIStocksTotalVolume {


     public static class MyMapper extends MapReduceBase implements
Mapper<LongWritable, Text, Text, IntWritable>

     {

          Text kword = new Text();

          IntWritable vword = new IntWritable();

          public void map(LongWritable key, Text value,

                    OutputCollector<Text, IntWritable> collector,
Reporter reporter)

                    throws IOException {

                    String line = value.toString();

                    String parts[] = line.split("\\t");

                    if(parts.length == 9)

                    {

                         kword.set(parts[1]);

                         vword.set(Integer.valueOf(parts[7]));

                         collector.collect(kword, vword);

                    }

          }

     }


     public static class MyReducer extends MapReduceBase implements
Reducer<Text, IntWritable, Text, LongWritable>

     {

          LongWritable vword = new LongWritable();

          public void reduce(Text key, Iterator<IntWritable> values,

                    OutputCollector<Text, LongWritable> collector,
Reporter reporter)
```

```
                    throws IOException {
                long sum = 0L;
                while(values.hasNext())
                {
                    sum = sum + values.next().get();
                }
                vword.set(sum);
                collector.collect(key, vword);
            }

    }

    public static void main(String args[]) throws IOException,
InterruptedException, ClassNotFoundException
    {
            Configuration conf = new Configuration();
            String otherArgs[] = new GenericOptionsParser(conf,
args).getRemainingArgs();

            JobConf job = new JobConf(conf);
            job.setJobName("Finding the Stocks Volume using Old MapReduce
API");

            job.setJarByClass(OldAPIStocksTotalVolume.class);

            job.setMapperClass(MyMapper.class);
            job.setCombinerClass(MyReducer.class);
            job.setReducerClass(MyReducer.class);

            job.setMapOutputKeyClass(Text.class);
            job.setMapOutputValueClass(IntWritable.class);
```

```
            job.setOutputKeyClass(Text.class);
            job.setOutputValueClass(LongWritable.class);

            job.setInputFormat(TextInputFormat.class);
            job.setOutputFormat(TextOutputFormat.class);

            FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
            FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));

            JobClient.runJob(job);
      }
}
```

Job Submission to Hadoop OR Usage:

cmd> hadoop jar oldapistockvolume.jar twok.hadoop.mapreduce.OldAPIStocksTotalVolume /markets/stocks /markets/volume1

MapReduce Job Two:

Objective: This job is finding the change in the stock value for the day i.e. close – open for every stock on every day.

Java Source Code:

```
package twok.hadoop.mapreduce;

import java.io.IOException;
```

```
import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.DoubleWritable;

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.input.TextInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

import org.apache.hadoop.util.GenericOptionsParser;


public class StockChangePerDay {


      /**

       * @param args

       * @throws IOException

       * @throws ClassNotFoundException

       * @throws InterruptedException

       */

      //This is the method for defining the MapReduce Driver

      public static void main(String[] args) throws IOException,
InterruptedException, ClassNotFoundException {

            Configuration conf = new Configuration();

            String otherArgs[] = new GenericOptionsParser(conf,
args).getRemainingArgs();


            if(otherArgs.length != 2)

            {
```

```
            System.out.println("Usage is: hadoop jar jarfile
MainClass input output");
            System.exit(1);
        }

        Job job = new Job(conf, "Stocks change per day");

        job.setJarByClass(StockChangePerDay.class);

        //To set Mapper and Reduce classes
        job.setMapperClass(MyMapper.class);
        job.setReducerClass(MyReducer.class);

        //Output Key-Value data types Type
        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(Text.class);

        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(DoubleWritable.class);

        //To inform input output Formats to MapReduce Program
        job.setInputFormatClass(TextInputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);

        //Inform input and output File or Directory locations
        FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
        FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));

        //Inform the job termination criteria
        System.exit(job.waitForCompletion(true) ? 0 : 1);
```

```
	}


	public static class MyMapper extends Mapper<LongWritable, Text,
Text, Text>

	{

		Text kword = new Text();

		Text vword = new Text();

		public void map(LongWritable key, Text value, Context context)
throws IOException, InterruptedException

		{

			String line = value.toString();

			String[] parts = line.split("\\t");

			if(parts.length == 9)

			{

				kword.set(parts[1] + "\t" + parts[2]);

				vword.set(parts[3] + ":" + parts[6]);

				context.write(kword, vword);

			}

		}

	}


	public static class MyReducer extends Reducer<Text, Text, Text,
DoubleWritable>

	{

		DoubleWritable vword = new DoubleWritable();

		public void reduce(Text key, Iterable<Text> values, Context
context) throws IOException, InterruptedException

		{

			double change = 0.0;

			for(Text value : values)

			{

				String[] parts = value.toString().split("\\:");
```

```
                        if(parts.length == 2)
                        {
                              change = Double.valueOf(parts[1]) -
Double.valueOf(parts[0]);
                        }
                  }
                  vword.set(change);
                  context.write(key, vword);
            }
      }
}
```

Job Submission to Hadoop OR Usage:

cmd> hadoop jar stockchange.jar twok.hadoop.mapreduce.StockChangePerDay /markets/stocks /markets/change

MapReduce Job Three:

Objective: The MapReduce job is for finding the total no of times per every stock.

Java Source Code:

```
package twok.hadoop.mapreduce;

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
```

```
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.input.TextInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

import org.apache.hadoop.util.GenericOptionsParser;


public class StockCount {


     /**

      * @param args

      * @throws IOException

      * @throws ClassNotFoundException

      * @throws InterruptedException

      */

     //This is the method for defining the MapReduce Driver

     public static void main(String[] args) throws IOException,
InterruptedException, ClassNotFoundException {

          Configuration conf = new Configuration();

          String otherArgs[] = new GenericOptionsParser(conf,
args).getRemainingArgs();


          if(otherArgs.length != 2)

          {

               System.out.println("Usage is: hadoop jar jarfile
MainClass input output");

               System.exit(1);
```

```
        }

        Job job = new Job(conf, "Stocks Counting");

        job.setJarByClass(StockCount.class);

        //To set Mapper and Reduce classes
        job.setMapperClass(MyMapper.class);
        job.setCombinerClass(MyReducer.class);
        job.setReducerClass(MyReducer.class);

        //Output Key-Value data types Type
        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(LongWritable.class);

        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(LongWritable.class);

        //To inform input output Formats to MapReduce Program
        job.setInputFormatClass(TextInputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);

        //Inform input and output File or Directory locations
        FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
        FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));

        //Inform the job termination criteria
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
```

```
     public static class MyMapper extends Mapper<LongWritable, Text,
Text, LongWritable>

     {

          Text kword = new Text();

          LongWritable vword = new LongWritable();

          public void map(LongWritable key, Text value, Context context)
throws IOException, InterruptedException

          {

               String line = value.toString();

               String[] parts = line.split("\\t");

               if(parts.length == 9)

               {

                    kword.set(parts[1]);

                    vword.set(1);

                    context.write(kword, vword);

               }

          }

     }


     public static class MyReducer extends Reducer<Text, LongWritable,
Text, LongWritable>

     {

          public void reduce(Text key, Iterable<LongWritable> values,
Context context) throws IOException, InterruptedException

          {

               long sum = 0;

               for(LongWritable value : values)

               {

                    sum = sum + value.get();

               }

               context.write(key, new LongWritable(sum));

          }
```

```
    }
}
```

Job Submission to Hadoop OR Usage:

cmd> hadoop jar stockcount.jar twok.hadoop.mapreduce.StockCount /markets/stocks /markets/count

MapReduce Job Four: This MapReduce program is for finding the min and max of open, high, low, close by passing job specific configuration parameters (Making Dynamic program for same business logic with different input data columns...)

Java Source Code:

```
package twok.hadoop.mapreduce;

import java.io.IOException;
import java.util.HashMap;
import java.util.Map;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.DoubleWritable;
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.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
```

```
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

import org.apache.hadoop.util.GenericOptionsParser;


public class StocksMinMaxOHLC {


     /**

      * @param args

      * @throws IOException

      * @throws ClassNotFoundException

      * @throws InterruptedException

      */

     public static void main(String[] args) throws IOException,
InterruptedException, ClassNotFoundException {

          Configuration conf = new Configuration();

          String otherArgs[] = new GenericOptionsParser(conf,
args).getRemainingArgs();

          if(otherArgs.length != 3)

          {

               System.out.println("Usage is: hadoop jar jarfile
MainClass input output parameter[open | high | low | close ]");

               System.exit(1);

          }

          Map<String, Integer> parameters= new HashMap<String,
Integer>();

          parameters.put("open", 3);

          parameters.put("high", 4);

          parameters.put("low", 5);

          parameters.put("close", 6);

          String parameter = otherArgs[2];


          conf.setInt("name", parameters.get(parameter));

          Job job = new Job(conf, "Finding min and max of open close
```

```
high low of every stock");
            job.setJarByClass(StocksMinMaxOHLC.class);

            job.setMapperClass(MyMapper.class);
            job.setReducerClass(MyReducer.class);

            job.setMapOutputKeyClass(Text.class);
            job.setMapOutputValueClass(DoubleWritable.class);

            job.setOutputKeyClass(Text.class);
            job.setOutputValueClass(Text.class);

            job.setInputFormatClass(TextInputFormat.class);
            job.setOutputFormatClass(TextOutputFormat.class);

            FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
            FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));

            System.exit(job.waitForCompletion(true) ? 0 : 1);

      }
      public static class MyMapper extends Mapper<LongWritable, Text,
Text, DoubleWritable>{
            Text kword = new Text();
            DoubleWritable vword = new DoubleWritable();
            int index = 0;
            public void setup(Context context)
            {
                  Configuration conf = context.getConfiguration();
                  index = Integer.valueOf(conf.get("name"));
```

```
        }
        public void map(LongWritable key, Text value, Context context)
throws InterruptedException, IOException
        {
              String line = value.toString();
              String[] parts = line.split("\\t");
              if(parts.length == 9)
              {
                    String stockName = parts[1];
                    double trade = Double.valueOf(parts[index]);
                    kword.set(stockName);
                    vword.set(trade);
                    context.write(kword, vword);
              }
        }
    }

    public static class MyReducer extends Reducer<Text, DoubleWritable,
Text, Text>
    {
        Text vword = new Text();
        public void reduce(Text key, Iterable<DoubleWritable> values,
Context context) throws IOException, InterruptedException
        {
              double min = Double.MAX_VALUE;
              double max = 0.0;
              for(DoubleWritable value : values)
              {
                    double current = value.get();
                    max = (max>current)?max:current;
                    min = (min<current)?min:current;
```

```
                }
                vword.set("Min: " + min + "\tMax: " + max);
                context.write(key, vword);
            }
        }
}
```

Job Submission to Hadoop OR Usage:

Usage is: hadoop jar jarfile MainClass input output parameter[open | high | low | close]

cmd> hadoop jar stockminmaxohlc.jar twok.hadoop.mapreduce.StocksMinMaxOHLC /markets/stocks /markets/open open

cmd> hadoop jar stockminmaxohlc.jar twok.hadoop.mapreduce.StocksMinMaxOHLC /markets/stocks /markets/high high

cmd> hadoop jar stockminmaxohlc.jar twok.hadoop.mapreduce.StocksMinMaxOHLC /markets/stocks /markets/low low

cmd> hadoop jar stockminmaxohlc.jar twok.hadoop.mapreduce.StocksMinMaxOHLC /markets/stocks /markets/close close

MapReduce Job Five: This MapReduce program is for finding the sum and average volume of the every stock for the entire data set.

Java Source Code:

```
package twok.hadoop.mapreduce;


import java.io.IOException;


import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
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.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;


public class StockVolumeAvgSum {


     public static class MyMapper extends Mapper<LongWritable, Text,
Text, LongWritable>{
          Text kword = new Text();
          LongWritable vword = new LongWritable();
          public void map(LongWritable key, Text value, Context context)
throws InterruptedException, IOException
          {
               String line = value.toString();
               String[] parts = line.split("\\t");
               if(parts.length == 9)
               {
```

```
                String stockName = parts[1];
                long volume = Long.valueOf(parts[7]);
                kword.set(stockName);
                vword.set(volume);
                context.write(kword, vword);
            }
        }
    }

    public static class MyReducer extends Reducer<Text, LongWritable,
Text, Text>
    {
        Text vword = new Text();
        public void reduce(Text key, Iterable<LongWritable> values,
Context context) throws IOException, InterruptedException
        {
            long sum = 0;
            double avg = 0.0;
            int counter = 0;
            for(LongWritable value : values)
            {
                sum = sum + value.get();
                counter++;
            }
            avg = (double) sum / counter;
            vword.set("sum: " + sum + "\tAverage: " + avg);
            context.write(key, vword);
        }
    }

    public static void main(String args[]) throws IOException,
```

```
InterruptedException, ClassNotFoundException

      {

            Configuration conf = new Configuration();

            String otherArgs[] = new GenericOptionsParser(conf,
args).getRemainingArgs();

            if(otherArgs.length != 2)

            {

                  System.out.println("Usage is: hadoop jar jarfile
MainClass input output");

                  System.exit(1);

            }

            Job job = new Job(conf, "Finding sum of the stock Volume");

            job.setJarByClass(StockVolumeAvgSum.class);


            job.setMapperClass(MyMapper.class);

            job.setReducerClass(MyReducer.class);


            job.setMapOutputKeyClass(Text.class);

            job.setMapOutputValueClass(LongWritable.class);

            job.setOutputKeyClass(Text.class);

            job.setOutputValueClass(Text.class);


            job.setInputFormatClass(TextInputFormat.class);

            job.setOutputFormatClass(TextOutputFormat.class);


            FileInputFormat.addInputPath(job, new Path(otherArgs[0]));

            FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));


            System.exit(job.waitForCompletion(true) ? 0 : 1);

      }

}
```

Job Submission to Hadoop OR Usage:

hadoop jar stockvolumeavgsum.jar twok.hadoop.mapreduce.StockVolumeAvgSum /markets/stocks /markets/avgsum

MapReduce Job Six: This program is for Finding the mix and max of open, high, close, low, adj_close, and volume as single report by using StockWritable Custom Data Type

Java Source Code:

StockWritable DataType:

```
package twok.hadoop.mapreduce;

import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;

import org.apache.hadoop.io.Writable;

public class StockWritable implements Writable {

	double open = 0.0;
	double high = 0.0;
	double low = 0.0;
	double close = 0.0;
	long volume = 0L;
```

```
@Override
public void readFields(DataInput in) throws IOException {
      this.open = in.readDouble();
      this.high = in.readDouble();
      this.low = in.readDouble();
      this.close = in.readDouble();
      this.volume = in.readLong();
}

@Override
public void write(DataOutput out) throws IOException {
      out.writeDouble(open);
      out.writeDouble(high);
      out.writeDouble(low);
      out.writeDouble(close);
      out.writeLong(volume);
}

//setter and getter Methods...
public void setOpen(Double stockOpen)
{
      open = stockOpen;
}
public Double getOpen()
{
      return open;
}
public void setHigh(Double stockHigh)
{
      high = stockHigh;
```

```
	}
	public Double getHigh()
	{
		return high;
	}
	public void setLow(Double stockLow)
	{
		low = stockLow;
	}
	public Double getLow()
	{
		return low;
	}
	public void setClose(Double stockClose)
	{
		close = stockClose;
	}
	public Double getClose()
	{
		return close;
	}
	public void setVolume(Long stockVolume)
	{
		volume = stockVolume;
	}
	public Long getVolume()
	{
		return volume;
	}
}
```

StockMinMaxAll.java

```
package twok.hadoop.mapreduce;

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
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.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;

public class StockMinMaxAll {

     /**
      * @param args
      * @throws IOException
      * @throws ClassNotFoundException
      * @throws InterruptedException
      */
     public static void main(String[] args) throws IOException,
InterruptedException, ClassNotFoundException {
```

```
            //Configuring/setting up MapReduce Driver
            Configuration conf = new Configuration();
            String otherArgs[] = new  GenericOptionsParser(conf,
args).getRemainingArgs();

          Job job = new Job(conf, "Fidning the mix and max of open, high,
close, low, adj_close, volume by using StockWritable Custom Data Type");

            job.setJarByClass(StockMinMaxAll.class);
            job.setMapperClass(MyMapper.class);
            job.setReducerClass(MyReducer.class);

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

            job.setInputFormatClass(TextInputFormat.class);
            job.setOutputFormatClass(TextOutputFormat.class);

            FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
            FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));

            System.exit(job.waitForCompletion(true) ? 0 : 1);
      }

      //Mapper Class for implementing Map side Business Logic
      public static class MyMapper extends Mapper<LongWritable, Text,
Text, StockWritable>
      {
```

```
            Text emitKey = new Text();
            StockWritable emitValue = new StockWritable();
            public void map(LongWritable key, Text value, Context context)
throws IOException, InterruptedException
            {
                  String line = value.toString();
                  String parts[] = line.split("\\t");
                  if(parts.length == 9)
                  {
                        emitKey.set(parts[1]);
                        emitValue.setOpen(Double.valueOf(parts[3]));
                        emitValue.setHigh(Double.valueOf(parts[4]));
                        emitValue.setLow(Double.valueOf(parts[5]));
                        emitValue.setClose(Double.valueOf(parts[6]));
                        emitValue.setVolume(Long.valueOf(parts[7]));
                        context.write(emitKey, emitValue);
                  }
            }
      }

      //Reducer Class for implementing Reducer side Business Logic
      public static class MyReducer extends Reducer<Text, StockWritable,
Text, Text>
      {
            Text emitValue  = new Text();
            public void reduce(Text key, Iterable<StockWritable> values,
Context context) throws IOException, InterruptedException
            {
                  double current = 0.0;
                  double minOpen = Double.MAX_VALUE;
                  double maxOpen = 0.0;
```

```
                double minHigh = Double.MAX_VALUE;
                double maxHigh = 0.0;
                double minLow = Double.MAX_VALUE;
                double maxLow = 0.0;
                double minClose = Double.MAX_VALUE;
                double maxClose = 0.0;
                long myCurrent = 0L;
                long minVolume = Long.MAX_VALUE;
                long maxVolume = 0L;
                int counter = 0;
                for(StockWritable value: values)
                {
                      current = value.getOpen();
                      minOpen = (minOpen < current) ? minOpen : current;
                      maxOpen = (maxOpen > current) ? maxOpen : current;
                      current = value.getHigh();
                      minHigh = (minHigh < current) ? minHigh : current;
                      maxHigh = (maxHigh > current) ? maxHigh : current;
                      current = value.getLow();
                      minLow = (minLow < current) ? minLow : current;
                      maxLow = (maxLow > current) ? maxLow : current;
                      current = value.getClose();
                      minClose = (minClose < current) ? minClose :
current;
                      maxClose = (maxClose > current) ? maxClose :
current;
                      myCurrent = value.getVolume();
                      minVolume = (minVolume < myCurrent) ? minVolume :
myCurrent;
                      maxVolume = (maxVolume > myCurrent) ? maxVolume :
myCurrent;
                      counter++;
```

```
                }

                String content = "Open: " + minOpen + " - " + maxOpen +
"\t" + "High: " + minHigh + " - " + maxHigh + "\t" + "Low: " + minLow +
"-" + maxLow + "\t" + "Close: " + minClose + " - " + maxClose + "\t" +
"Volume: " + minVolume + " - " + maxVolume + "\t" + "Total Records" +
counter;

                emitValue.set(content);

                context.write(key, emitValue);

           }

      }

}
```

Job Submission to Hadoop OR Usage:

cmd> javac -d StockWritable.java

cmd> jar -cvf stockwritable.jar -C ./com/hadoop/mapreduce StockWritable.class

cmd> cp stockwritable.jar $HADOOP_HOME/lib.

Cmd> stop-all.sh

cmd> start-all.sh

cmd> hadoop jar stockminmaxall.jar twok.hadoop.mapreduce.StockMiMaxAll /markets/stocks /markets/report

MapReduce Job Seven: This MapReduce program is for finding the min and max of open, high, low, close of every stock (Same as MapReduce job Four) by using Second Approach.

Java Source Code:

StockMinMaxMain.java:

```
package twok.hadoop.mapreduce;


import java.io.IOException;

import java.util.HashMap;

import java.util.Map;


import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.DoubleWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

import org.apache.hadoop.util.GenericOptionsParser;


public class StockMinMaxMain {

     public static void main(String[] args) throws IOException,
InterruptedException, ClassNotFoundException {

          Configuration conf = new Configuration();

          String otherArgs[] = new GenericOptionsParser(conf,
args).getRemainingArgs();


          if(args.length != 3)

          {

               System.out.println("Usage is: hadoop jar jarname
mainclass input output option");

               System.out.println("Avalable options are open | high |
low | close");

               System.exit(1);

          }
```

```
            //To make Dynamic this program to work with multiple
columns....

          Map<String, Integer> options =  new HashMap<String, Integer>();
           options.put("open", 3);
           options.put("high", 4);
           options.put("low", 5);
           options.put("close", 6);

           String option = otherArgs[2];
           conf.setInt("name", options.get(option));

           Job job = new Job(conf, "Find the volume of Stocks");
           job.setJarByClass(StockMinMaxMain.class);

           job.setMapperClass(StockMinMaxMapper.class);
           job.setReducerClass(StockMinMaxReducer.class);

           job.setMapOutputKeyClass(Text.class);
           job.setMapOutputValueClass(DoubleWritable.class);

           job.setOutputKeyClass(Text.class);
           job.setOutputValueClass(Text.class);

           job.setInputFormatClass(TextInputFormat.class);
           job.setOutputFormatClass(TextOutputFormat.class);

           FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
           FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
```

```
            System.exit(job.waitForCompletion(true) ? 0 : 1);
            }
}
```

StockMinMaxMapper.java:

```
package twok.hadoop.mapreduce;

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.io.DoubleWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;



public class StockMinMaxMapper extends Mapper<LongWritable, Text, Text,
DoubleWritable>
{
      int processColumn = 0;
      public void setup(Context context)
      {
            Configuration conf = context.getConfiguration();
            processColumn = Integer.valueOf(conf.get("name"));
      }

      Text kword = new Text();
      DoubleWritable vword = new DoubleWritable();
```

```
	public void map(LongWritable key, Text value, Context context)
throws IOException, InterruptedException
	{
		String line = value.toString();
		String dataParts[] = line.split("\\t");
		if(dataParts.length == 9)
		{
			kword.set(dataParts[1]);
			vword.set(Double.valueOf(dataParts[processColumn]));
			context.write(kword, vword);
		}
	}
}
```

StockMinMaxReducer.java

```
package twok.hadoop.mapreduce;

import java.io.IOException;

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

public class StockMinMaxReducer extends Reducer<Text, DoubleWritable,
Text, Text>
{
	Text vword = new Text();
	public void reduce(Text key, Iterable<DoubleWritable> values,
Context context) throws IOException, InterruptedException
	{
```

```
            double min = Double.MAX_VALUE;
            double max = 0.0;
            for(DoubleWritable value : values)
            {
                  double myvalue = value.get();
                  max = (max > myvalue) ? max : myvalue;
                  min = (min < myvalue) ? min : myvalue;
            }
            String minmax = "Min is " + min + ", Max is " + max;
            vword.set(minmax);
            context.write(key, vword);
      }
}
```

Job Submission to Hadoop OR Usage:

Usage is: hadoop jar jarfile MainClass input output parameter[open | high | low | close]

cmd> hadoop jar stockminmax.jar twok.hadoop.mapreduce.StockMinMaxMain /markets/stocks /markets/open open

cmd> hadoop jar stockminmax.jar twok.hadoop.mapreduce.StockMinMaxMain /markets/stocks /markets/high high

cmd> hadoop jar stockminmax.jar twok.hadoop.mapreduce.StockMinMaxMain /markets/stocks /markets/low low

cmd> hadoop jar stockminmax.jar twok.hadoop.mapreduce.StockMinMaxMain /markets/stocks /markets/close close

Kmeans Clustering Algorithm: It is useful to cluster the

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FSDataInputStream;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
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.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;

public class KmeansCluster
{
     public static final String hdfsPath = "/kmeans/cluster";
     public static class MyMapper extends Mapper<LongWritable, Text,
Text, Text>
     {
          Integer[][] centroids = new Integer[2][2];
          public void setup(Context context) throws IOException
          {
               FSDataInputStream in = null;
```

```
            BufferedReader br = null;
            FileSystem fs =
        FileSystem.get(context.getConfiguration());
            Path path = new Path(hdfsPath);
            in = fs.open(path);
            br  = new BufferedReader(new InputStreamReader(in));
            String line = "";
            int i=0;
            while ( (line = br.readLine() )!= null) {
            String[] arr = line.split("\\,");
            if (arr.length == 2)
            {
                  int j=0;
                  centroids[i][j] = Integer.valueOf(arr[0]);
                  j++;
                  centroids[i][j] = Integer.valueOf(arr[1]);
            }
            i++;
          }
          in.close();
        }
        public void map(LongWritable key, Text value, Context context)
throws InterruptedException, IOException
        {
            String line = value.toString();
            String dataPoints[] = line.split("\\,");
            float distance = 0.0f;
            Text emitValue = new Text();
            Text emitKey = new Text();
            float min = Float.MAX_VALUE;
            float current = 0.0f;
```

```
                String clusterPoint = "";
                for(int i=0; i<centroids.length; i++)
                {
                      int xdiff = centroids[i][0] -
Integer.valueOf(dataPoints[0]);
                      int ydiff = centroids[i][1] -
Integer.valueOf(dataPoints[1]);
                      int xcord = xdiff * xdiff;
                      int ycord = ydiff * ydiff;
                      distance = (float) Math.sqrt(xcord + ycord);
                      current = distance;
                      if(min >= current)
                      {
                            min = current;
                            clusterPoint = centroids[i][0] + "," +
centroids[i][1];
                      }
                }
                String myPoint = dataPoints[0] + "," + dataPoints[1];
                emitKey.set(clusterPoint);
                emitValue.set(myPoint);
                context.write(emitKey, emitValue);
          }
     }

     public static class MyReducer extends Reducer<Text, Text, Text,
Text>
     {
          Text emitKey = new Text();
          Text emitValue = new Text();
          public void reduce(Text key, Iterable<Text> values, Context
context) throws IOException, InterruptedException
```

```
        {
            float newX = 0.0f;
            float newY = 0.0f;
            int sumX = 0;
            int sumY = 0;
            int counter = 0;
            String clusterPoints = "";
            int i = 0;
            for(Text value : values)
            {
                String line = value.toString();
                String coordinates[] = line.split("\\,");
                sumX = sumX + Integer.valueOf(coordinates[0]);
                sumY = sumY + Integer.valueOf(coordinates[1]);

                counter++;
                if(i == 0)
                {
                    clusterPoints = line;
                    i = 1;
                }
                else
                {
                    clusterPoints = clusterPoints + ";" + line;
                }
            }
            newX = (float) sumX/counter;
            newY = (float) sumY/counter;
            String clusterKey = "Cluster: " + key.toString();
            String clusterInfo = newX + "," + newY + "\t" +
clusterPoints;
```

```
                emitKey.set(clusterKey);

                emitValue.set(clusterInfo);

                context.write(emitKey, emitValue);

          }

     }


     public static void main(String args[]) throws IOException,
InterruptedException, ClassNotFoundException

     {

          Configuration conf = new Configuration();

          String otherArgs[] = new GenericOptionsParser(conf,
args).getRemainingArgs();

          if(otherArgs.length != 2)

          {

                System.out.println("Usage is: hadoop jar jarfile
MainClass input output");

                System.exit(1);

          }

          Job job = new Job(conf, "Sample Kmeans Algorithm....");

          job.setJarByClass(KmeansCluster.class);


          job.setMapperClass(MyMapper.class);

          job.setReducerClass(MyReducer.class);


          job.setMapOutputKeyClass(Text.class);

          job.setMapOutputValueClass(Text.class);


          job.setOutputKeyClass(Text.class);

          job.setOutputValueClass(Text.class);


          job.setInputFormatClass(TextInputFormat.class);
```

```
        job.setOutputFormatClass(TextOutputFormat.class);

        FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
        FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));

        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```