Mukul Sharma **Homework - 4**

CS-6240-02

Psedocode – Secondary

Mapper Class :

Map(Object key, Text value, Context context){

1. Parse the input and extract all values
2. Check if the flight is not canceled and is in the year 2008
3. Create the flight key and a value as ArraydelayMinutes and write.

}

Partitioner :

```
//The partitioning is done by sing the hashCode of unique carrier
getPartition(FlightKey key, Text value, int num)
        {
                n = Math.abs(key.getFlightName().hashCode()) %  num;
                return n;
        }
```

KeyComparator :

```
//This does the sorting of input key on the basis of the UniqueCarrier and the month of that carrrier
compare(WritableComparable w1, WritableComparable w2) {
                FlightKey ip1 = (FlightKey) w1;
                FlightKey ip2 = (FlightKey) w2;

            int cmp = ip1.getFlightName().compareTo(ip2.getFlightName());
            if(cmp == 0)
            return (ip1.getMonth().compareTo(ip2.getMonth()));
            else
                 return cmp;
          }
```

GroupComparator :

```
//For Grouping, we only do it using the UniqueCarrier
compare(WritableComparable w1, WritableComparable w2) {
             FlightKey ip1 = (FlightKey) w1;
             FlightKey ip2 = (FlightKey) w2;
             return (ip1.getFlightName().compareTo(ip2.getFlightName()));
            }
```

//Reducer Code :

```
reduce(FlightKey key, Iterable<Text> values, Context context)
{
//Actual processing
for(Text vItem : values)
                        {
                                airline = key.getFlightName();
                                currentMonth = key.getMonth().get();
                                currentDelay = Float.parseFloat(vItem.toString());

                                if(currentMonth != lastMonth)
                                {
                                        averageDelay[lastMonth-1] =
(int)Math.ceil(totalDelay/flights);
                                        flights = 0;
                                        totalDelay = 0;
                                        lastMonth = currentMonth;
                                }
                                totalDelay = totalDelay + currentDelay;
                                flights ++;
                        }

                        //average for last month
                        averageDelay[lastMonth-1] = (int)Math.ceil(totalDelay/flights);

                        //Create output string
                        for(int i = 0; i < averageDelay.length; i++)
                        {
                                if(i<0)
                                        {
                                                outputStr.append(",");
                                        }
                                outputStr.append("(");
                                outputStr.append(i+1);
                                outputStr.append(",");
                                outputStr.append(averageDelay[i]);
                                outputStr.append(")");
                        }

                        //Emit
                        pairs.set(outputStr.toString());
                        Text rAirline = new Text(airline);
                        context.write(rAirline, pairs);
}
```

Source-Code :

Secondary.java

```
public class Secondary {

        //------Key Comparator------//
        public static class KeyComparator extends WritableComparator {
           protected KeyComparator() {
             super(FlightKey.class, true);
           }

           @Override
           public int compare(WritableComparable w1, WritableComparable w2) {
                   FlightKey ip1 = (FlightKey) w1;
                   FlightKey ip2 = (FlightKey) w2;

              int cmp = ip1.getFlightName().compareTo(ip2.getFlightName());
              if(cmp == 0)
              return (ip1.getMonth().compareTo(ip2.getMonth()));
              else
                    return cmp;
           }
         }

         /*-------- Group Comparator ------- */
         public static class GroupComparator extends WritableComparator {
           protected GroupComparator() {
             super(FlightKey.class, true);
           }

           @Override
           public int compare(WritableComparable w1, WritableComparable w2) {
             FlightKey ip1 = (FlightKey) w1;
             FlightKey ip2 = (FlightKey) w2;
             return (ip1.getFlightName().compareTo(ip2.getFlightName()));

           }
         }

  /*--------- Partitioner -----------*/
   public static class airlinePartitioner extends Partitioner<FlightKey, Text>
   {
          @Override
          public int getPartition(FlightKey key, Text value, int num)
          {
                  int n = Math.abs(key.getFlightName().hashCode()) %  num;
                  return n;
          }
   }

          /*----------- Mapper Class ----------*/
   public static class flightDelayMapper
                  extends Mapper<Object, Text, FlightKey, Text>
          {
                  @Override
                  public void map(Object key, Text value, Context context) throws                  IOException,
InterruptedException
                  {
```

```
		if(((LongWritable) key).get()>0)
		{
			// Get data from CSV file line by line
			CSVParser parser = new CSVParser();
			String[] lineEntries = parser.parseLine(value.toString());

			// Extract all values
			String ArrDelayMinutes = lineEntries[37];
			String cancelled = lineEntries[41];
			String diverted = lineEntries[43];
			String month = lineEntries[2];
			String year = lineEntries[0];
			String flightNum = lineEntries[6];

			FlightKey nKey = new FlightKey();
			Text nVal;

			//Check if the current flight is the eligible flight
			if(hasValidDate(year) && notCanceledOrDiverted(cancelled,
	diverted)
					&& flightNum.length()!=0 && month.length()!=0
					&& ArrDelayMinutes.length()!=0)
			{	String id = flightNum;
				IntWritable m = new
IntWritable(Integer.parseInt(month));
				nKey.set(id, m);
				nVal = new Text(ArrDelayMinutes);
				context.write(nKey, nVal);
			}
		}
	}

	/* 2. The Flight lies IN the year 2008. */
	boolean hasValidDate(String year){
		if (year.equals("2008"))
			return true;
		else
			return false;
	}

	/* 3. The flight is not cancelled or diverted */
	boolean notCanceledOrDiverted(String cancelled, String diverted){
		if( cancelled.equals("0.00") && diverted.equals("0.00"))
			return true;
		else
			return false;
	}

}


/******* R E D U C E *******/
public static class flightDelayReducer extends		Reducer<FlightKey,Text,Text,Text>
{
	private String airline = null;
	private Text pairs = new Text();


	@Override
```

```
        public void reduce(FlightKey key, Iterable<Text> values, Context context)throws
IOException, InterruptedException
        {
                float totalDelay = 0;
                float currentDelay = 0;
                int lastMonth = 1;
                int flights = 0;
                int currentMonth = 0;
                int[] averageDelay = new int[12];
                StringBuilder outputStr = new StringBuilder("");

                for(Text vItem : values)
                {
                        airline = key.getFlightName();
                        currentMonth = key.getMonth().get();
                        currentDelay = Float.parseFloat(vItem.toString());

                        if(currentMonth != lastMonth)
                        {
                                averageDelay[lastMonth-1] =
    (int)Math.ceil(totalDelay/flights);
                                flights = 0;
                                totalDelay = 0;
                                lastMonth = currentMonth;
                        }
                        totalDelay = totalDelay + currentDelay;
                        flights ++;
                }

                //average for last month
                averageDelay[lastMonth-1] = (int)Math.ceil(totalDelay/flights);

                //Create output string
                for(int i = 0; i < averageDelay.length; i++)
                {
                        if(i<0)
                                {
                                        outputStr.append(",");
                                }
                        outputStr.append("(");
                        outputStr.append(i+1);
                        outputStr.append(",");
                        outputStr.append(averageDelay[i]);
                        outputStr.append(")");
                }

                //Emit
                pairs.set(outputStr.toString());
                Text rAirline = new Text(airline);
                context.write(rAirline, pairs);
        }
}

/* D r i v e r */
public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        String[] otherArgs = new GenericOptionsParser(conf,                    args).getRemainingArgs();
  if (otherArgs.length != 2) {
    System.err.println("Usage: Secondary <input> <output>");
```

```
            System.exit(2);
        }
            Job job = new Job(conf, "Average Flight Delay Pattern");
            job.setJarByClass(Secondary.class);
            job.setMapperClass(flightDelayMapper.class);
            job.setReducerClass(flightDelayReducer.class);

            job.setPartitionerClass(airlinePartitioner.class);
            job.setGroupingComparatorClass(GroupComparator.class);
            job.setSortComparatorClass(KeyComparator.class);

            job.setNumReduceTasks(10);
            job.setMapOutputKeyClass(FlightKey.class);
            job.setMapOutputValueClass(Text.class);
            job.setOutputKeyClass(IntWritable.class);
            job.setOutputValueClass(Text.class);
            FileInputFormat.addInputPath(job, new Path(args[0]));
            FileOutputFormat.setOutputPath(job, new Path(args[1]));

            if(job.waitForCompletion(true)){
                System.exit(0);
            }
            else
                System.exit(1);
    }

}
```

FlightKey.java

```
// Class to represent each FlightKey
public class FlightKey implements WritableComparable<FlightKey>{
    private String flight = null;
    private IntWritable month = new IntWritable();

    public String getFlightName(){
        return flight;
    }

    public IntWritable getMonth(){
        return month;
    }

    public void setFlightName(String id){
        this.flight = id;
    }

    public void setMonth(IntWritable month){
        this.month = month;
    }

    public FlightKey(){
        setFlightName(null);
        setMonth(null);
    }

    public void set(String id, IntWritable month ){
```

```
		this.flight = id;
		this.month = month;
	}

	@Override
	public void readFields(DataInput arg0) throws IOException {
		// TODO Auto-generated method stub
		if(month == null){
			month = new IntWritable();
		}
		flight = arg0.readUTF();
		month.readFields(arg0);
	}

	@Override
	public void write(DataOutput arg0) throws IOException {
		// TODO Auto-generated method stub
		arg0.writeUTF(flight);
		month.write(arg0);

	}

	@Override
	public int compareTo(FlightKey arg0) {
		// TODO Auto-generated method stub
		int result = this.flight.compareTo(arg0.getFlightName());
		if(result == 0)
			return this.month.compareTo(arg0.getMonth());
		else
		return result;
	}

	@Override
	public int hashCode(){
		int result = flight.hashCode()*5+month.hashCode();
		return result;
	}

}
```

HBASE : Hpopulate.java :

```
package hw4HBase;

import au.com.bytecode.opencsv.CSVParser;

public class HPopulate {

	public static final String F_TABLE_NAME = "FlightData";
	public static final String flight_fam = "family";

	//Create all columns
	public static final String[] columns = {"Year","Quarter","Month",
		"DayofMonth","DayOfWeek","FlightDate","UniqueCarrier",
		"AirlineID","Carrier","TailNum","FlightNum","Origin",
		"OriginCityName","OriginState","OriginStateFips","OriginStateName",
		"OriginWac","Dest","DestCityName","DestState","DestStateFips",
		"DestStateName","DestWac","CRSDepTime","DepTime","DepDelay",
		"DepDelayMinutes","DepDel15","DepartureDelayGroups","DepTimeBlk",
		"TaxiOut","WheelsOff","WheelsOn","TaxiIn","CRSArrTime","ArrTime",
		"ArrDelay","ArrDelayMinutes","ArrDel15","ArrivalDelayGroups",
		"ArrTimeBlk","Cancelled","CancellationCode","Diverted","CRSElapsedTime ",
		"ActualElapsedTime","AirTime","Flights","Distance","DistanceGroup",
		"CarrierDelay","WeatherDelay","NASDelay","SecurityDelay","LateAircraft Delay"};

	/*------- Mapper Class ------- */
	public static class HPopulateMapper extends Mapper<Object, Text, Text, IntWritable>{

		CSVParser parser = new CSVParser();
		Configuration config;
		HTable table;
		List <Put> p;
	   long count = 0;

		//Table Setup
		protected void setup (Context context) throws IOException,
InterruptedException{
			config = HBaseConfiguration.create();
			table = new HTable(config, F_TABLE_NAME);
			table.setAutoFlush(false);
			p = new LinkedList<Put>();
		}

		// Map Function
		public void map(Object key, Text value, Context context) throws IOException{
			//get data from CSV file line by line
			String[] lineEntries = parser.parseLine(value.toString());
			int fligthMonth = Integer.parseInt(lineEntries[2]);
			int flightYear = Integer.parseInt(lineEntries[0]);
			if(flightYear == 2008){
				String myRow = lineEntries[6] + lineEntries[2];
				int lengthofKeyPart1 = myRow.length();
				Put puts = new
Put(Bytes.toBytes(myRow+System.nanoTime()));
				for(int i=0; i< 54 ; i++){
					puts.add(Bytes.toBytes(flight_fam),
Bytes.toBytes(columns[i]),
```

```
                                        Bytes.toBytes(lineEntries[i]));
                                }
                                p.add(puts);
                        }
                }

                // Cleanup; Close table
                protected void cleanup(Context context) throws          IOException,InterruptedException {
                        table.put(p);
                        table.close();
        }
        }

        /*---------- Driver Function ---------*/
        public static void main(String[] args) throws Exception{

                Configuration conf = HBaseConfiguration.create();
                String[] otherargs = new GenericOptionsParser(conf,          args).getRemainingArgs();

                if(otherargs.length != 2){
                        System.err.println("Usage <input path> <output>");
                        System.exit(2);
                }

                HBaseAdmin admin = new HBaseAdmin(conf);
                HTableDescriptor htd = new HTableDescriptor(F_TABLE_NAME);
                HColumnDescriptor hcd = new HColumnDescriptor(flight_fam);
                htd.addFamily(hcd);
                admin.createTable(htd);
                admin.close();
                Job job = new Job(conf, "HBase Populate");
                job.setJarByClass(HPopulate.class);
                job.setMapperClass(HPopulateMapper.class);
                job.setNumReduceTasks(10);
                job.setOutputKeyClass(Text.class);
           job.setOutputValueClass(IntWritable.class);
                job.setOutputFormatClass(TableOutputFormat.class);
                job.getConfiguration().set(TableOutputFormat.OUTPUT_TABLE,          F_TABLE_NAME);

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

                if(job.waitForCompletion(true)){
                        admin.close();
                }
                else{
                        System.exit(1);
                }
        }
}
```

Hcompute.java :

```
public class HCompute {

	private static String t = "FlightData";
	private static String flight_fam = "family";
	private static String fmonth = "Month";
	private static String fdelay = "ArrDelayMinutes";
	private static String uc = "UniqueCarrier";
	private static String can = "Cancelled";
	private static String y = "Year";

	/*------- MAPPER -------*/
	public static class HComputeMapper extends TableMapper<Text, Text>{
		private Text fKey = new Text();
		private Text fVal = new Text();

		@Override
		public void map(ImmutableBytesWritable rkey,
					Result values, Context context)
				throws IOException, InterruptedException{

			// Extract all values
			String carrier = Bytes.toString(values.getValue
			(Bytes.toBytes(flight_fam), Bytes.toBytes(uc)));
			String month = Bytes.toString(values.getValue
			(Bytes.toBytes(flight_fam), Bytes.toBytes(fmonth)));
			String year = Bytes.toString(values.getValue
			(Bytes.toBytes(flight_fam), Bytes.toBytes(y)));
			String delay = Bytes.toString(values.getValue
			(Bytes.toBytes(flight_fam), Bytes.toBytes(fdelay)));
			String canceled = Bytes.toString(values.getValue
			(Bytes.toBytes(flight_fam), Bytes.toBytes(can)));

			fKey.set(carrier);
			fVal.set(month + "," + delay);
			context.write(fKey, fVal);
		}

	}

	/*------- REDUCER -------*/
	public static class HComputeReducer extends Reducer<Text, Text, Text, Text> {

		@Override
		public void reduce (Text key, Iterable<Text> values, Context context){
			String flight = key.toString();
			double[] totalDelay = new double[12];
			int[] count= new int[12];
			Arrays.fill(count, 0);
			Arrays.fill(totalDelay, 0);

			for (Text val : values){
				String parts[] = val.toString().split(",");
				int month = Integer.parseInt(parts[0]);
				double delayMinutes = Double.parseDouble(parts[1]);
				totalDelay[month -1] += delayMinutes;
				count[month - 1] += 1;
```

```
                        }

                        // Creating output String
                        StringBuilder sBuilder  = new StringBuilder();
                        for(int i=0; i< totalDelay.length; i++){
                                if(i > 0){
                                        sBuilder.append(",");
                                }
                                int avgDelay = (int) Math.ceil(((double)
totalDelay[i])/count[i]);
                                sBuilder.append("(").append(i+1).
                                Append(",").
                                append(avgDelay).append(")");
                        }

                        try {
                                context.write(new Text (flight), new
Text(sBuilder.toString()));
                        } catch (IOException e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                        } catch (InterruptedException e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                        }

                }
        }

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

                if(otherArgs.length != 2){
                        System.err.println("Usage: HCompute <Input> <Output>");
                        System.exit(2);
                }

                Scan scan = new Scan();
                scan.setCaching(500);
                scan.setCacheBlocks(false);

                Job job = new Job(conf, "Monthly Flight Delay");
                job .setJarByClass(HCompute.class);
                TableMapReduceUtil.initTableMapperJob(t, scan,
                                HComputeMapper.class, Text.class, Text.class, job);
                job.setMapOutputKeyClass(Text.class);
                job.setMapOutputValueClass(Text.class);
                job.setReducerClass(HComputeReducer.class);
                job.setNumReduceTasks(10);

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

                if(job.waitForCompletion(true)){
                        System.exit(0);
```

```
                }
                else
                        System.exit(1);
        }
}
```

Performance Comparison:

Running times

	Java	HPopulate	HCompute
6 Machines	299 Seconds	2390 Seconds	2103 Seconds
11 Machines	256 seconds	375 Seconds	297 Seconds

(i) Coding and setup effort
The secondary sort program doesn't require any setup when compared to Hbase programs.
In Hbase you need to write Hpopulate and Hcompute to first populate the table and then perform the computations.

(ii) performance,
In terms of performance,
For the java program, the 11 machine cluster performs better than the 6 machine cluster as in this case the load gets distributed among various machines .

For the Hbase program as it translates the the entire CSV file to Htable, it takes very long time and the execution of Hcompute then uses this HBase table to compute the desired result.

In the end it turns out that the total running time of the Hbase programs that is the running time of the Hpopulate and Hcompute is much more than the java program because of the requirement to first create the hbase table and then read data from it to compute the result.

(iii) scalability AND Loadbalancing
The Secondry program is capable of scaling well so as we increase the number of workers, the task can be easily distributed this also shows that this program is load balanced as the division of job on the basis of uniquecarrier results to enough jobs that can be executed on multiple machines.

The Hcompute program is also good in scalability as we are creating the table only once and then multiple operations can be performed on thatsame table, but for load balancing we will have to break the chunk size in smaller parts to achieve good load balancing.

Hence the above analysis describes the performance differences in the above three programs.