Big Data Exp 3 : 1211061

Matrix Multiplication:

```
//IntPair.java

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

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Writable;
import org.apache.hadoop.io.WritableComparable;

public class IntPair implements WritableComparable<IntPair>{

	private IntWritable i;
	private IntWritable k;

	IntPair(){

		i=new IntWritable();
		k=new IntWritable();
	}
	public void set(int i, int k){

		this.i.set(i);
		this.k.set(k);
	}
	public int getI()
	{

		return(i.get());
	}
	public int getK()
	{
		return(k.get());
	}

	public String toString()
	{
		return i.get() + "," + k.get();

	}
	@Override
	public void readFields(DataInput input) throws IOException {
		// TODO Auto-generated method stub
		i.readFields(input);
		k.readFields(input);

	}
	@Override
	public void write(DataOutput output) throws IOException {
		// TODO Auto-generated method stub

		i.write(output);
		k.write(output);

	}
	@Override
	public int hashCode() {
		final int prime = 31;
		int result = 1;
		result = prime * result + ((i == null) ? 0 : i.hashCode());
		result = prime * result + ((k == null) ? 0 : k.hashCode());
		return result;
	}

	@Override
	public boolean equals(Object obj) {
		if (this == obj)
			return true;
		if (obj == null)
			return false;
		if (getClass() != obj.getClass())
			return false;
		IntPair other = (IntPair) obj;
		if (i == null) {
			if (other.i != null)
				return false;
		} else if (!i.equals(other.i))
			return false;
		if (k == null) {
			if (other.k != null)
				return false;
		} else if (!k.equals(other.k))
			return false;
		return true;
	}

	@Override
	public int compareTo(IntPair second) { //sorting and grouping of key. it is complex key
		// TODO Auto-generated method stub
		int cmp=this.i.compareTo(second.i);
		if(cmp !=0)
		{
			return cmp;
		}
		else // i is same of both now return the comparison of k
			return this.k.compareTo(second.k);
	}

}
```

```
//Relation.java

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

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.Writable;

public class Relation implements Writable {
        private Text fromMorN;
        private IntWritable iOrK;
        private IntWritable mijOrnik;

public Relation(){

        fromMorN=new Text();
        iOrK=new IntWritable();
        mijOrnik=new IntWritable();
}

public void set(String sourceMatrix, int matrixCordinate, int matrixCellvalue){

        fromMorN.set(sourceMatrix);
    iOrK.set(matrixCordinate);
    mijOrnik.set(matrixCellvalue);

}

public String getFromMatrix(){
        return fromMorN.toString();
}

public int getIorK(){
        return iOrK.get();
}

public int getmijOrnik(){
        return mijOrnik.get();
}


        @Override
        public void readFields(DataInput input) throws IOException {
                // TODO Auto-generated method stub

                fromMorN.readFields(input);
                iOrK.readFields(input);
                mijOrnik.readFields(input);
        }
```

```
	@Override
	public void write(DataOutput output) throws IOException {
		// TODO Auto-generated method stub

		fromMorN.write(output);
		iOrK.write(output);
		mijOrnik.write(output);


	}

}
```

Driver:

```
//Step1Driver.java

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.jobcontrol.JobControl;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.jobcontrol.ControlledJob;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class Step1Driver {

	public static void main(String[] args) throws Exception {
		Configuration conf = new Configuration();
		Job job1 = Job.getInstance(conf, "step1ProductOfallcordinates");
		job1.setJarByClass(Step1Driver.class);
		job1.setMapperClass(Step1Mapper.class);

		job1.setReducerClass(Step1Reducer.class);

		// TODO: specify output types

		job1.setMapOutputKeyClass(IntWritable.class);
		job1.setMapOutputValueClass(Relation.class);
		job1.setOutputKeyClass(IntPair.class);
		job1.setOutputValueClass(IntWritable.class);

		// TODO: specify input and output DIRECTORIES (not files)
		FileInputFormat.setInputPaths(job1, new Path("/home/kjsce/Desktop/Matrix"));
		FileOutputFormat.setOutputPath(job1, new
Path("/home/kjsce/Desktop/Matrix_Output1"));
```

```
                }

                /************************ job 1 ends */


                Job job2 = Job.getInstance(conf, "AdditionofProduct");
                job2.setJarByClass(Step1Driver.class);
                job2.setMapperClass(Step2Mapper.class);

                job2.setReducerClass(Step2Reducer.class);

                // TODO: specify output types

                job2.setMapOutputKeyClass(IntPair.class);
                job2.setMapOutputValueClass(IntWritable.class);
                job2.setOutputKeyClass(IntPair.class);
                job2.setOutputValueClass(IntWritable.class);

                // TODO: specify input and output DIRECTORIES (not files)
                FileInputFormat.setInputPaths(job2, new
Path("/home/kjsce/Desktop/Matrix_Output1"));
                FileOutputFormat.setOutputPath(job2, new
Path("/home/kjsce/Desktop/Matrix_Output"));


                /********************************  job 2 ends********/

                ControlledJob cj1=new ControlledJob(conf);
                cj1.setJob(job1);
                ControlledJob cj2=new ControlledJob(conf);
                cj2.setJob(job2);

                cj2.addDependingJob(cj1);
                JobControl jobControl=new  JobControl("Matrixmultiplication");
                jobControl.addJob(cj1);
                jobControl.addJob(cj2);

                Thread newThread=new Thread(jobControl);
                newThread.setDaemon(true);// now it is a daemon thread. JVM stops.
                newThread.start(); // making jobcontrol as a thread.

                while(!jobControl.allFinished()){
                        System.out.println("Still multiplying");
                        newThread.sleep(4000);

                }



        }

}
```

Reducers:

```
//Step1Reducer.java

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

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

public class Step1Reducer extends
                Reducer<IntWritable, Relation, IntPair, IntWritable> {
/*bug of eclipse Iterable<text> , Text _key*/

        private IntPair key=new IntPair();
        private IntWritable value=new IntWritable();
        public void reduce(IntWritable _key, Iterable<Relation> values, Context context)
                        throws IOException, InterruptedException {

                ArrayList<Relation> mRels=new ArrayList<>();
                ArrayList<Relation> nRels=new ArrayList<>();


                //separating mrelation and nrelation

                for(Relation value: values){

                        //for every relation create a new object at the reducer side
                        Relation temp=new Relation();
                        //transfer the data from old object to new object

                        temp.set(value.getFromMatrix(), value.getIorK(), value.getmijOrnik());

                        if(value.getFromMatrix().equals("M")){

                                mRels.add(temp);

                        }

                        else
                        {
                                nRels.add(temp);

                        }
                }

                for (Iterator iterator = mRels.iterator(); iterator.hasNext();)
                {
                        Relation mrelation = (Relation) iterator.next();
                        for (Iterator iterator2 = nRels.iterator(); iterator2.hasNext();)
                {
                    Relation nrelation = (Relation) iterator2.next();
                    key.set(mrelation.getIorK(), nrelation.getIorK());
                    value.set(mrelation.getmijOrnik()* nrelation.getmijOrnik());
                    context.write(key, value);
                }
        }
    }
}


//Step2Reducer.java

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

public class Step2Reducer extends
        Reducer<IntPair, IntWritable, IntPair, IntWritable> {
    private IntWritable value=new IntWritable();

    public void reduce(IntPair _key, Iterable<IntWritable> values, Context context)
            throws IOException, InterruptedException {
        int sum=0;
        for(IntWritable value:values){
            sum+=value.get();

        }
        value.set(sum);
        context.write(_key, value);
    }

}
```

Mappers:

```
//Step1Mapper.java

import java.io.IOException;

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


public class Step1Mapper extends Mapper<LongWritable, Text,IntWritable,Relation> {

    private IntWritable key=new IntWritable();
    private Relation value=new Relation();
```

```
	public void map(LongWritable ikey, Text ivalue, Context context)
			throws IOException, InterruptedException {
		String line=ivalue.toString();
		String[] tokens=StringUtils.split(line,',');
		if(tokens[0].equals("M")){
			key.set(Integer.parseInt(tokens[2]));
			value.set(tokens[0],Integer.parseInt(tokens[1]),Integer.parseInt(tokens[3]));
		}
		else
		{
			key.set(Integer.parseInt(tokens[1]));
			value.set(tokens[0],Integer.parseInt(tokens[2]),Integer.parseInt(tokens[3]));
		}

		context.write(key,value);

	}

}

//Step2Mapper.java

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

public class Step2Mapper extends
		Mapper<LongWritable, Text, IntPair, IntWritable> {

	private IntPair key=new IntPair();
	private IntWritable value=new IntWritable();

	public void map(LongWritable ikey, Text ivalue, Context context)
			throws IOException, InterruptedException {


		String line=ivalue.toString();
		String[] tokens=StringUtils.split(line,'\t');
		String[]  subtokens=StringUtils.split(tokens[0],',');

		key.set(Integer.parseInt(subtokens[0]), Integer.parseInt(subtokens[1]));
		value.set(Integer.parseInt(tokens[1]));
		context.write(key, value);



	}

}
```

Matrix_input:

```
M,0,0,1
M,0,1,2
M,1,0,3
M,1,1,2
N,0,0,1
N,0,1,5
N,1,0,2
N,1,1,6
```

1st Map Reduce output:

```
1,1     15
1,0     3
0,1     5
0,0     1
1,1     12
1,0     4
0,1     12
0,0     4
```

2nd Map Reduce (final) output:

```
0,0     5
0,1     17
1,0     7
1,1     27
```