Task 3 - Calculate total revenue where the movie rating is over 6.5 for each company

How we implement

- We upload *tmdb_5000_movies.csv* to HDFS, and then perform the following map-reduce task:
- The mapper reads one line the csv file whose grids are separated by ",", parses it with the JAVA library *opencsv* and get the rating, revenue and film company list. The company list is a JSON array, and is then parsed by *gson*. If **rating >= 6.5** For each company making the movie, the mapper generates <Company Id, (Company_name, revenue)>.
- The reducer simply acts like Word Count and sums up all the high-rate revenues for each company. It outputs <Company Id, (Company_name, total revenue)>.

- Code as follows:

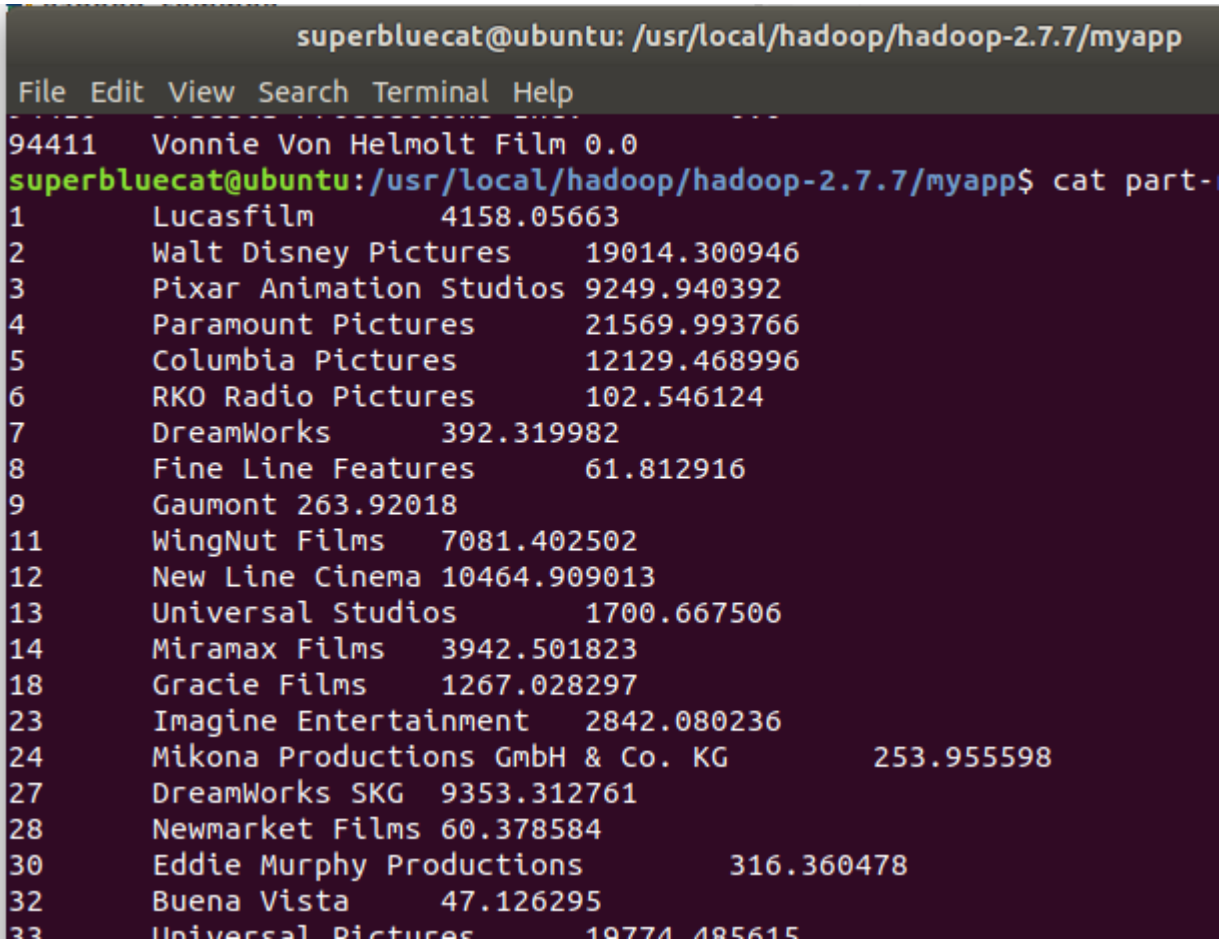
```
import java.io.IOException;
import java.util.StringTokenizer;
import java.io.DataOutput;
import java.io.DataInput;

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.DoubleWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.Writable;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.Mapper.Context;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import com.opencsv.CSVParser;
import com.google.gson.JsonArray;
import com.google.gson.JsonElement;
import com.google.gson.JsonIOException;
import com.google.gson.JsonObject;
import com.google.gson.JsonParser;
import com.google.gson.JsonSyntaxException;

public class Movie {
    public static class Pair implements Writable
    {
        String s = new String();
        DoubleWritable f = new DoubleWritable();
        Pair(){}
```

```
        Pair(String s_, DoubleWritable f_)
        {
            s = s_;
            f.set(f_.get());
        }
        @Override
        public void write(DataOutput out) throws IOException {
            out.writeUTF(this.s);
            this.f.write(out);
        }
        @Override
        public void readFields(DataInput in) throws IOException {
            this.s = in.readUTF();
            this.f.readFields(in);
        }
        @Override
        public String toString()
        {
            return s + "\t" + f.toString();
        }
    }
    public static class FilmMapper
    extends Mapper<Object, Text, LongWritable, Pair >{

        public void map(Object key, Text value, Context context
                ) throws IOException, InterruptedException {
            String[] units = new CSVParser().parseLine(value.toString());

            if(units[9].equals("production_companies")) return;

            //System.out.println(units[9]);
            double score = Double.parseDouble(units[18]);
            if(score >= 6.5)
            {
            JsonArray companies = new JsonParser().parse(units[9]).getAsJsonArray();
            for(JsonElement val : companies)
            {
                JsonObject obj = val.getAsJsonObject();
                //System.out.println(obj.get("id"));
                context.write(new LongWritable(obj.get("id").getAsLong()),
                        new Pair(obj.get("name").getAsString(),
                                new DoubleWritable(Double.parseDouble(units[12]))));
            }
            }
        }
}
    public static class DoubleSumCombiner
    extends Reducer<LongWritable, Pair, LongWritable, Pair> {
    private Pair result = new Pair();

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

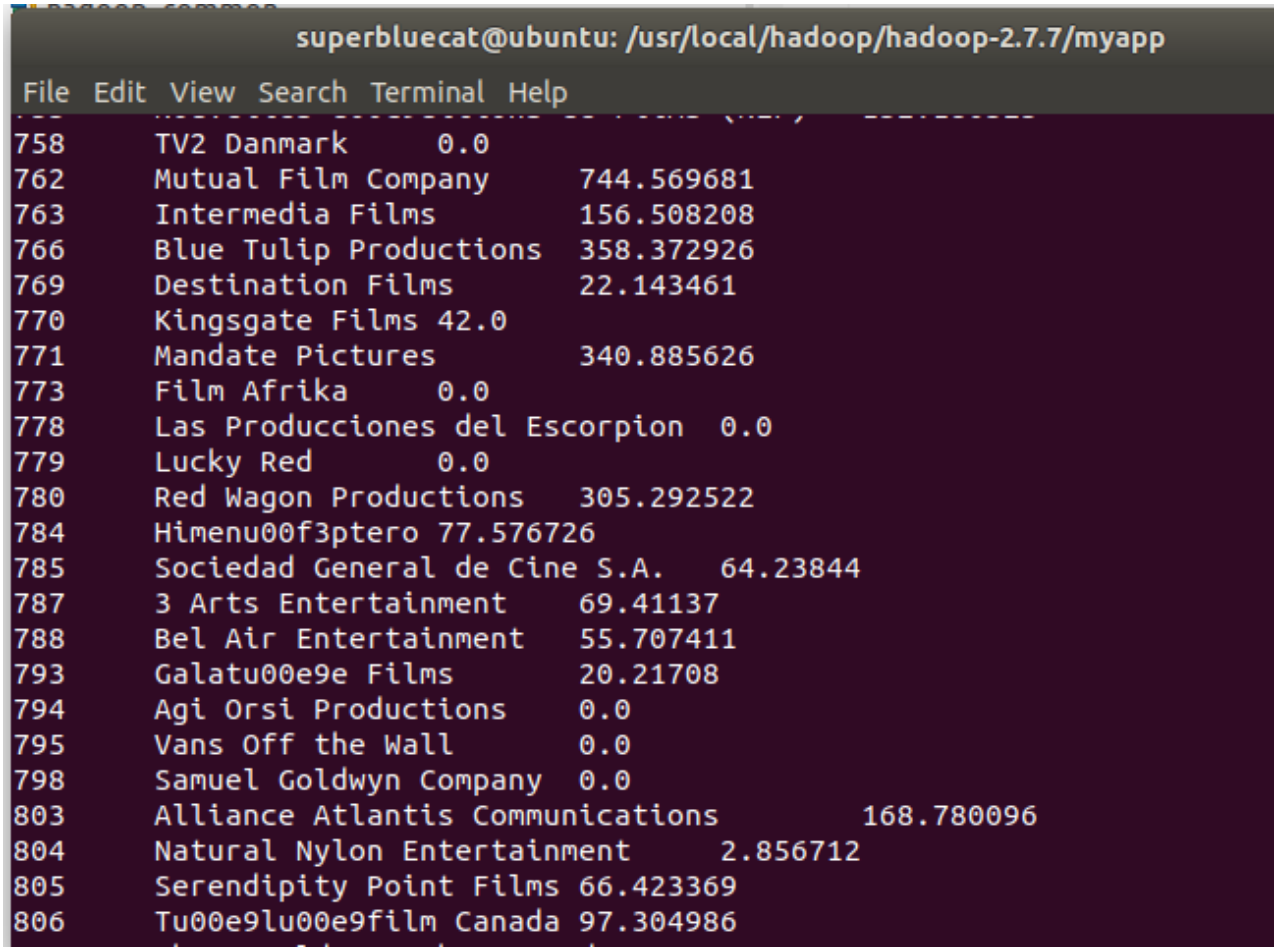
```
    double sum = 0;
    for (Pair val : values) {
      sum += val.f.get();
      result.s = val.s;
    }
    result.f.set(sum);
    context.write(key, result);
     }
     }

     public static class DoubleSumReducer
     extends Reducer<LongWritable, Pair, LongWritable, Pair> {
     private Pair result = new Pair();

     public void reduce(LongWritable key, Iterable<Pair> values,
                    Context context
                    ) throws IOException, InterruptedException {
    double sum = 0;
    for (Pair val : values) {
      sum += val.f.get();
      result.s = val.s;
    }
    result.f.set(sum / 1e6);
    context.write(key, result);
     }
     }

       public static void main(String[] args) throws Exception {
             if(args.length!=2){
                 System.err.println("Uage: wordcount <in> <out>");
                 System.exit(2);
             }
             Configuration conf = new Configuration();
             Job job = Job.getInstance(conf, "Movie");
             job.setJarByClass(Movie.class);
             job.setMapperClass(FilmMapper.class);
             job.setCombinerClass(DoubleSumCombiner.class);
             job.setReducerClass(DoubleSumReducer.class);
             job.setOutputKeyClass(LongWritable.class);
             job.setOutputValueClass(Pair.class);
             FileInputFormat.addInputPath(job, new Path(args[0]));
             FileOutputFormat.setOutputPath(job, new Path(args[1]));
             System.exit(job.waitForCompletion(true) ? 0 : 1);
           }
     }
```

Results

- Total revenue for each company is output as shown in figures. **The sums are displayed in Millions.** Those famous companies are certainly good at making nice movies and big names.



- Still, some companies ends up zero such as Sony. We believe that's just a few losses of data from the datasets: **Some movies are high in rating, but their revenues aren't there, therefore 0 instead of NULL.**



Reflections

- MapReduce is very powerful in processing such independent data by paralleling.
- Some packages such as csv or JSON are convenient for string processing.
- Although not emphasized in Hadoop documents, **when we define new classes for values, we should implement it based on *Writable* and override the *write and read* methods.**
- Combiner can be utilized for better performance.

Task 4 - Self multiply of a matrix

How we implement

- We perform the MapReduce job according to Prof. Chen's teaching in class.
- First, we create *input/tmp* from *adj.txt*. Each line contains a triple **(r, c, value)** which contains the index and value of **non-zero** elements. The size **N** is also calculated in the process.
- Mapper treats input/tmp as input. For each **(r, c, value)** mapper generates 2N key-values:

  (r, c, value) -> <(r, i), (0, c, value)> (r, c, value) -> <(i, c), (1, r, value)> i = 1 to N

  the 0/1 in the output triple stands for left/right operand of the number in the result element.
- Reducer collects triples for each element in answer, multiply and sums the corresponding values.

  <(r,c) , result>
- Another function convert the triple expression to normal matrix as final answer.

- Code as follows:

```
import java.io.IOException;
import java.util.StringTokenizer;
import java.io.DataOutput;
import java.io.DataInput;
import org.apache.hadoop.fs.FileSystem;

import org.apache.hadoop.fs.FSDataOutputStream;
import org.apache.hadoop.fs.FSDataInputStream;
import java.io.BufferedReader;

import java.io.InputStreamReader;

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.DoubleWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.Writable;
import org.apache.hadoop.io.WritableComparable;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.Mapper.Context;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import com.opencsv.CSVParser;
import com.google.gson.JsonArray;
import com.google.gson.JsonElement;
import com.google.gson.JsonIOException;
import com.google.gson.JsonObject;
import com.google.gson.JsonParser;
import com.google.gson.JsonSyntaxException;

public class MM {
    public static int N = 0;

    public static class Pair implements WritableComparable<Pair>
    {
        IntWritable r = new IntWritable();
        IntWritable c = new IntWritable();
        Pair(){}
        Pair(int r_, int c_)
        {
            r.set(r_);
            c.set(c_);
        }
        @Override
        public void write(DataOutput out) throws IOException {
            this.r.write(out);
```

```
            this.c.write(out);
        }
        @Override
        public void readFields(DataInput in) throws IOException {
            this.r.readFields(in);
            this.c.readFields(in);
        }
        @Override
        public String toString()
        {
            return r.toString() + "\t" + c.toString();
        }
        @Override
        public int compareTo(Pair other) {
            if(this.r.compareTo(other.r) != 0)
                return this.r.compareTo(other.r);
            return this.c.compareTo(other.c);
         }
    }

    public static class Triple implements Writable
    {
        Pair a = new Pair();
        LongWritable b = new LongWritable();
        Triple(){}
        Triple(int r_, int c_, long b_)
        {
            a.r.set(r_);
            a.c.set(c_);
            b.set(b_);
        }
        @Override
        public void write(DataOutput out) throws IOException {
            this.a.write(out);
            this.b.write(out);
        }
        @Override
        public void readFields(DataInput in) throws IOException {
            this.a.readFields(in);
            this.b.readFields(in);
        }
        @Override
        public String toString()
        {
            return a.toString() + "\t" + b.toString();
        }
    }

    public static int createInput(Configuration conf, String f)
    {
    try
    {
        FileSystem fs = FileSystem.get(conf);
```

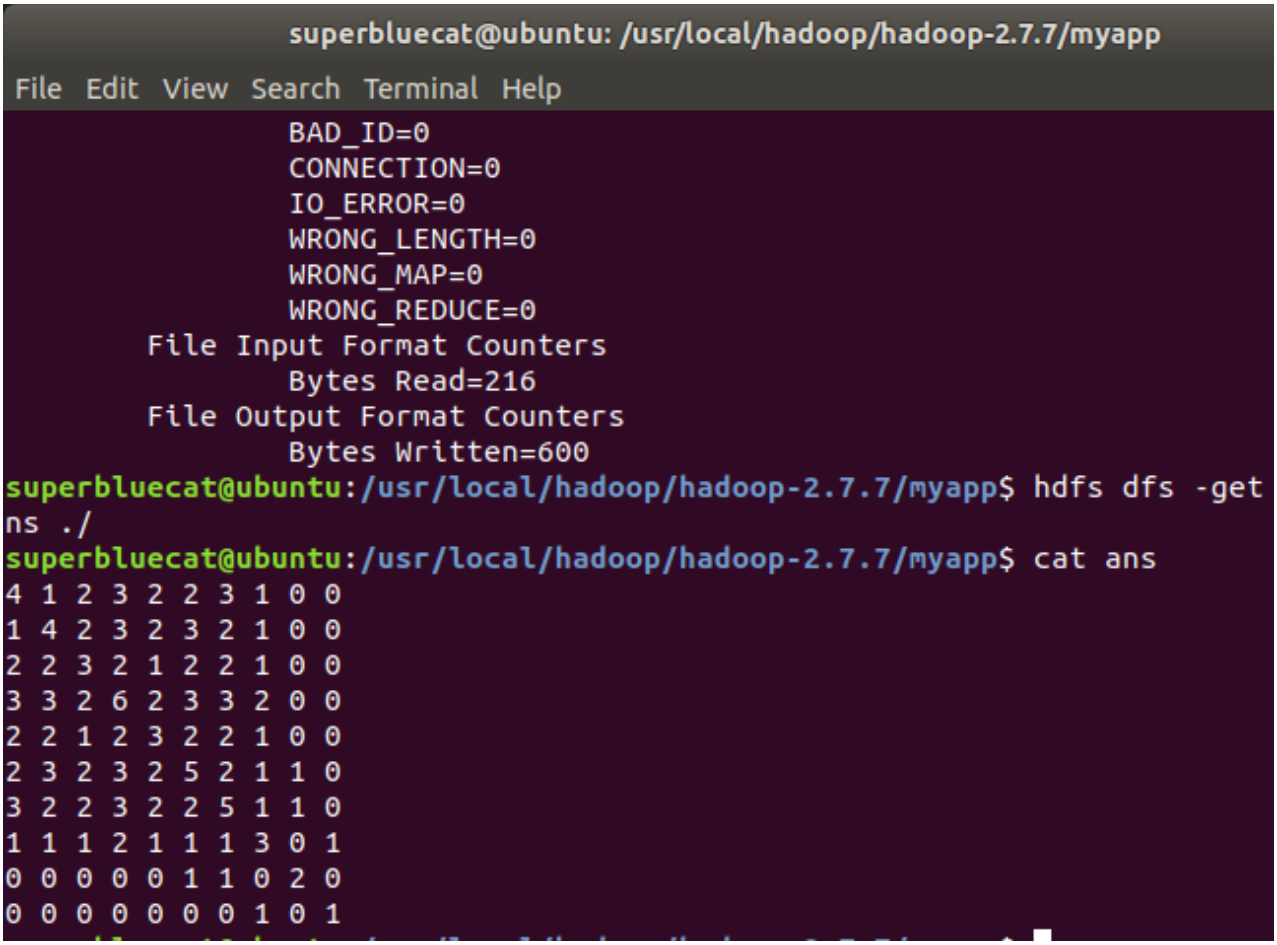
```
        FSDataInputStream in = fs.open(new Path(f));
        BufferedReader d = new BufferedReader(new InputStreamReader(in));
        FSDataOutputStream os =fs.create(new Path("input/tmp"));

        int cnt = 0;
        String s;
        while((s = d.readLine()) != null)
        {
            N++;
            String [] nums = s.split(" ");
            for(int i = 0;i < nums.length; ++i)
            {
                int val = Integer.parseInt(nums[i]);
                if(val != 0)
                {
                    String Line = Integer.toString(cnt) + " " +
                            Integer.toString(i) + " " + Integer.toString(val) + "\n";
                    byte[] buff = Line.getBytes();
                    os.write(buff,0,buff.length);
                }
            }
            cnt++;
        }
        in.close();
        d.close();
        os.close();
        return cnt;
    }
    catch (Exception e){
        e.printStackTrace();
        return 0;
    }}

    public static class MatrixMapper
    extends Mapper<Object, Text, Pair, Triple >{

        public void map(Object key, Text value, Context context
                ) throws IOException, InterruptedException {
            String[] units = value.toString().split(" ");
            int r = Integer.parseInt(units[0]);
            int c = Integer.parseInt(units[1]);
            int val = Integer.parseInt(units[2]);
            for(int i = 0;i < N;i++)
                context.write(new Pair(r, i), new Triple(0, c, val));
            for(int i = 0;i < N;i++)
                context.write(new Pair(i, c), new Triple(1, r, val));
        }
    }
    /*public static class identityCombiner
    extends Reducer<Pair, Triple, Pair, Triple> {


    public void reduce(Pair key, Iterable<Triple> values,
```

```
                Context context
                ) throws IOException, InterruptedException {
for (Triple val : values)
    context.write(key, val);
 }
 }*/

 public static class ProdSumReducer
 extends Reducer<Pair, Triple, Pair, LongWritable> {
 private LongWritable ans = new LongWritable();

 public void reduce(Pair key, Iterable<Triple> values,
                Context context
                ) throws IOException, InterruptedException {
long[] left = new long[N], right = new long[N];
for (Triple val : values) {
  if(val.a.r.get() == 0)
      left[val.a.c.get()] = val.b.get();
  else
      right[val.a.c.get()] = val.b.get();
}
long sum = 0;
for(int i = 0;i < N;i++)
    sum += left[i] * right[i];
ans.set(sum);
context.write(key, ans);
 }
 }

 public static void createOutput(Configuration conf, String f)
 {
 try
 {
     FileSystem fs = FileSystem.get(conf);
     FSDataInputStream in = fs.open(new Path(f + "/part-r-00000"));
     BufferedReader d = new BufferedReader(new InputStreamReader(in));
     FSDataOutputStream os =fs.create(new Path(f + "/ans"));

     long [][] Matrix = new long [N][N];
     String s;
     while((s = d.readLine())!=null)
     {
         String [] units = s.split("\t");
         int r = Integer.parseInt(units[0]);
         int c = Integer.parseInt(units[1]);
         int val = Integer.parseInt(units[2]);
         Matrix[r][c] = val;
     }
     for(int i = 0;i < N;i++)
     {
         for(int j = 0;j < N;j++)
         {
             byte[] buff = (Long.toString(Matrix[i][j]) + " ").getBytes();
```

```
                os.write(buff, 0, buff.length);
            }
            byte[] buff = ("\n").getBytes();
            os.write(buff, 0, buff.length);
        }


        in.close();
        d.close();
        os.close();

    }
    catch (Exception e){
        e.printStackTrace();


    }}

      public static void main(String[] args) throws Exception {
            if(args.length!=2){
                System.err.println("Uage: wordcount <in> <out>");
                System.exit(2);
            }
            Configuration conf = new Configuration();
            createInput(conf, args[0]);
            //System.out.println(N);
            //Configuration conf = new Configuration();
            Job job = Job.getInstance(conf, "MM");
            job.setJarByClass(MM.class);
            job.setMapperClass(MatrixMapper.class);
            //job.setCombinerClass(identityCombiner.class);
            job.setReducerClass(ProdSumReducer.class);
            job.setOutputKeyClass(Pair.class);
            job.setOutputValueClass(LongWritable.class);
            job.setMapOutputKeyClass(Pair.class);
            job.setMapOutputValueClass(Triple.class);
            FileInputFormat.addInputPath(job, new Path("input"));
            FileOutputFormat.setOutputPath(job, new Path(args[1]));
            job.waitForCompletion(true);
            //conf = new Configuration();
            createOutput(conf, args[1]);
          }
    }
```

Result

- We get the self-multiply answer successfully.



- The matrix is very small and the answer comes very quickly.

Reflections

- We didn't find a way to involve the line number directly to Mapper, and therefore had to build a tmp file. As is shown in Stack Overflow:

  2

  - The default InputFormats such as TextInputFormat will give the byte offset of the record rather than the actual line number - this is mainly due to being unable to determine the true line number when an input file is splittable and being processed by two or more mappers.
  - You can create your own InputFormat to produce line numbers rather than byte offsets but you need to configure input format to return false from the isSplittable method (a large input file would not be processed by multiple mappers). If you have small files, or files that are close in size the HDFS block size then this is not a problem.
  - You can also use pig to clean your data and get those particular interested lines and process that particular data .

    I feel this is a draw back of Hadoop, Hadoop fails when you want to share global state across different systems.

  Therefore MapReduce do have shortcomings when it comes to relevant computing.

- Although not emphasized in Hadoop documents, **when we define new classes for keys, we should implement it based on *WritableComparable* and override the *write and read and CompareTo* methods.**
- MapOutputValue and MapOutputKey should be specified when it's different from output of reducer!