作业1

> 姓名： 吴双
>
> 学号： 10164102141

1 软件体系结构

Map

> 要求
>
> > 班级学生成绩的随机生成
> >
> > > 输入：本班同学的学号 输出：<学号，成绩>
>
> 数据
>
> > `stuID.csv文件`，每一列为一个学号，在 hdfs 上，shell input:

```
hdfs dfs put stuID.csv input
```

分组MapReduce

> 要求
>
> > 求平均成绩：将全班同学每隔5号分为一组，求每组的平均成绩
> >
> > > 输入： <学号，成绩> 输出：<组号，平均分>
> >
> > 数据
> >
> > > 一个`score.csv文件`，每一列为学号和学生成绩，放入`HDFS`中，shell input:

```
hdfs dfs put score.csv input
```

NaturalJoin

> 要求：

> 匹配出person中每个人所在的位置信息；
>
> 每条记录各个字段之间以空格为分隔符。

数据：

> `person.txt`
>
> `address.txt`

Kmeans

> 数据
>
> > `k-means.dat`
> >
> > > 第一行标明K的值和数据个数N, 均为整形, 由”,”隔开 (如 3,10 表示K=3, N=10)。
> > >
> > > 之后N行中每行代表一个二维向量, 向量元素均为整形, 由”,”隔开 (如 1,2 表示向量(1, 2))。
> >
> > 输出:
> >
> > > K行, 每行是一个聚类图心的二维向量, 向量元素均为浮点型 (如 1.1,2.3)。
> >
> > 思路
> >
> > - 首先需要初始化中心点，这里使用前四行作为初始中心点，现将其写入`cache`中。
> > - 每次`map`时使用`setup`函数读入变量中，依据所有点与中心店的距离选择其属于的类。
> > - 在`reduce`中，根据类别进行分组，对每组聚类重新选择中心点，将中心点输出到目标文件中。
> > - 判断两次的中心点是否满足阈值条件，若不满足，则将新生成的中心点移动到`cache`中，作为下一次迭代的中心点。
> > - 迭代结束的标志为：满足最大迭代次数或满足阈值条件。

2 详细设计

基础MapReduce

code:

```
1  import org.apache.hadoop.conf.Configuration;
2  import org.apache.hadoop.fs.Path;
3  import org.apache.hadoop.io.IntWritable;
4  import org.apache.hadoop.io.LongWritable;
5  import org.apache.hadoop.io.Text;
6  import org.apache.hadoop.mapreduce.Job;
7  import org.apache.hadoop.mapreduce.Mapper;
8  import org.apache.hadoop.mapreduce.Reducer;
9  import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
10 import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
11 import org.apache.hadoop.util.GenericOptionsParser;
12
13 import java.io.IOException;
14 import java.util.Random;
15
16 public class Score {
```

```
17
18      public Score() {
19
20      }
21
22      public static class StuIDMapper
23              extends Mapper<Object, Text, LongWritable, IntWritable> {
24
25          private final static IntWritable one = new IntWritable(1);
26          LongWritable ID = new LongWritable();
27
28          public void map(Object key, Text value, Context context)
29                  throws IOException, InterruptedException {
30
31              ID.set(Long.valueOf(value.toString()));
32
33              context.write(this.ID, one);
34
35          }
36      }
37
38      public static class ScoreReducer
39              extends Reducer<LongWritable, IntWritable, LongWritable, IntWritable> {
40          private IntWritable score = new IntWritable();
41          private Random rand = new Random();
42
43          public void reduce(LongWritable ID, Iterable<IntWritable> values, Context
    context)
44                  throws IOException, InterruptedException {
45              score.set(rand.nextInt(100) + 1);
46              context.write(ID, this.score);
47          }
48
49      }
50
51      public static void main(String[] args) throws Exception {
52
53          org.apache.hadoop.conf.Configuration conf = new Configuration();
54          String[] otherArgs = (new GenericOptionsParser(conf,
    args)).getRemainingArgs();
55          if (otherArgs.length < 2) {
56              System.err.println("Usage: wordcount <in>[<in>...] <out>");
57              System.exit(2);
58          }
59
60          Job job = Job.getInstance(conf, "student score");
61          job.setJarByClass(Score.class);
62          job.setMapperClass(Score.StuIDMapper.class);
63          job.setReducerClass(Score.ScoreReducer.class);
64          job.setOutputKeyClass(LongWritable.class);
65          job.setOutputValueClass(IntWritable.class);
66
67          for (int i = 0; i < otherArgs.length - 1; i++) {
```

```
            FileInputFormat.addInputPath(job, new Path(otherArgs[i]));
        }

        FileOutputFormat.setOutputPath(job, new Path(otherArgs[otherArgs.length -
1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }

}
```

运行

```
hadoop jar Score.jar input/stuID.csv output
hdfs dfs -ls output/*
```

```
4190558352      38
4190816219      78
4191096730      16
4191634667      33
4192944206      100
4193320729      72
4194369068      29
4195553731      66
4197472805      38
4199736992      91
4200463785      97
4200761214      39
4209095544      30
```

分组MapReduce

code:

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.FloatWritable;
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.Partitioner;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.KeyValueLineRecordReader;
import org.apache.hadoop.mapreduce.lib.input.KeyValueTextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;

```

```java
import java.io.IOException;
import java.util.Random;

public class GetAvgScore {

    public GetAvgScore() {

    }

    public static class StuScoreMapper
            extends Mapper<Text, Text, IntWritable, IntWritable> {

        IntWritable Group = new IntWritable();
        IntWritable Score = new IntWritable();

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

            // every 5 students as a group,[1,2,3,4,5] => group1
            Group.set((Integer.parseInt(key.toString()) - 1) / 5 + 1);
            Score.set(Integer.parseInt(value.toString()));

            context.write(this.Group, this.Score);

        }
    }

    public static class AvgScoreReducer
            extends Reducer<IntWritable, IntWritable, IntWritable, FloatWritable> {
        private FloatWritable avgscore = new FloatWritable();

        public void reduce(IntWritable Group, Iterable<IntWritable> Score, Context
context)
                throws IOException, InterruptedException {

            int sum = 0;
            int count = 0;
            for (IntWritable val : Score) {
                sum += val.get();
                count++;
            }
            avgscore.set((float) sum / count);
            context.write(Group, avgscore);
        }

    }


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

        org.apache.hadoop.conf.Configuration conf = new Configuration();
//        set seperator
        conf.set(KeyValueLineRecordReader.KEY_VALUE_SEPERATOR, ",");
```

```
        String[] otherArgs = (new GenericOptionsParser(conf,
args)).getRemainingArgs();
        if (otherArgs.length < 2) {
            System.err.println("Usage: wordcount <in>[<in>...] <out>");
            System.exit(2);
        }

        Job job = Job.getInstance(conf, "student avg score");
        job.setInputFormatClass(KeyValueTextInputFormat.class);
        job.setJarByClass(GetAvgScore.class);

        job.setMapperClass(GetAvgScore.StuScoreMapper.class);
        job.setReducerClass(GetAvgScore.AvgScoreReducer.class);

        job.setMapOutputKeyClass(IntWritable.class);
        job.setMapOutputValueClass(IntWritable.class);
        job.setOutputKeyClass(IntWritable.class);
        job.setOutputValueClass(FloatWritable.class);

        for (int i = 0; i < otherArgs.length - 1; i++) {
            FileInputFormat.addInputPath(job, new Path(otherArgs[i]));
        }

        FileOutputFormat.setOutputPath(job, new Path(otherArgs[otherArgs.length -
1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }

}
```

运行

```
hadoop jar avgcore.jar input/score.csv output/avgscore/
hdfs dfs -cat output/avgscore/*
```

```
hadoop jar Natural\ join.jar input/person.txt input/address.txt output/natural_join
```

natural join

code:

```
import org.apache.commons.lang.StringUtils;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
```

```java
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.FileSplit;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;


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


public class natural_join {
    public natural_join() {

    }

    public static class joinMapper
            extends Mapper<Object, Text, Text, Text> {

        private static final String PERSON_FLAG = "person";
        private static final String ADDRESS_FLAG = "address";

        private FileSplit fileSplit;
        private Text outKey = new Text();
        private Text outValue = new Text();


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

            fileSplit = (FileSplit) context.getInputSplit();
            String filePath = fileSplit.getPath().toString();

            String line = value.toString();
            String[] fields = StringUtils.split(line, " ");

            // 判断记录来自哪个文件
            if (filePath.contains(PERSON_FLAG)) {
                if (fields.length < 3)
                    return;
                //   fields[2] is code
                outKey.set(fields[2]);
                outValue.set(PERSON_FLAG + "," + line);
            } else if (filePath.contains(ADDRESS_FLAG)) {
                //   fields[0] is city code
                outKey.set(fields[0]);
                outValue.set(ADDRESS_FLAG + "," + fields[1]);
            }

            context.write(outKey, outValue);

```

```
59
60          }
61      }
62
63      public static class joinReducer
64              extends Reducer<Text, Text, Text, Text> {
65          private static final String PERSON_FLAG = "person";
66          private static final String ADDRESS_FLAG = "address";
67
68          private String fileFlag = null;
69          private String cityName = null;
70
71          private Text outCity = new Text();
72          private Text outPerson = new Text();
73
74          public void reduce(Text key, Iterable<Text> values, Context context)
75                  throws IOException, InterruptedException {
76
77              List<String> perosonInfo = new ArrayList<>();
78              for (Text val : values) {
79                  String[] fields = StringUtils.split(val.toString(), ",");
80                  fileFlag = fields[0];
81                  // choose what file it is
82                  if (fileFlag.equals(ADDRESS_FLAG)) {
83                      cityName = fields[1];
84                      outCity.set(cityName);
85                  } else if (fileFlag.equals(PERSON_FLAG)) {
86                      perosonInfo.add(fields[1]);
87                  }
88              }
89
90              // Cartesian product
91              for (String person : perosonInfo) {
92                  outPerson.set(person);
93                  context.write(outPerson, outCity);
94              }
95          }
96
97      }
98
99      public static void main(String[] args) throws Exception {
100
101         org.apache.hadoop.conf.Configuration conf = new Configuration();
102         conf.set("mapreduce.output.textoutputformat.separator", " ");
103         String[] otherArgs = (new GenericOptionsParser(conf,
    args)).getRemainingArgs();
104         if (otherArgs.length <= 2) {
105             System.err.println("Usage: natural <in> <in> <out>");
106             System.exit(2);
107         }
108
109         Job job = Job.getInstance(conf, "natural join");
110         job.setJarByClass(natural_join.class);
```

```
111
112         job.setMapperClass(natural_join.joinMapper.class);
113         job.setReducerClass(natural_join.joinReducer.class);
114
115         job.setOutputKeyClass(Text.class);
116         job.setOutputValueClass(Text.class);
117
118         for (int i = 0; i < otherArgs.length - 1; i++) {
119             FileInputFormat.addInputPath(job, new Path(otherArgs[i]));
120         }
121
122         FileOutputFormat.setOutputPath(job, new Path(otherArgs[otherArgs.length -
    1]));
123         System.exit(job.waitForCompletion(true) ? 0 : 1);
124     }
125 }
```

运行

```
1 hadoop jar Natural\ join.jar input/person.txt input/address.txt output/natural_join
2 hdfs dfs -cat output/natural_join/*
```

```
23 Alva 210000 Nanjing
19 Alger 210000 Nanjing
32 Ansel 210000 Nanjing
1 Aaron 210000 Nanjing
41 Bancroft 210000 Nanjing
27 Andre 210000 Nanjing
62 Bruno 212000 Zhenjiang
59 Boyce 212000 Zhenjiang
46 Barry 213000 Changzhou
37 Atwood 213000 Changzhou
68 Clare 213000 Changzhou
44 Barlow 213000 Changzhou
18 Alfred 214000 Wuxi
```

Kmeans

code:

class Point:

```
1 class Point{
2     double x;
3     double y;
4     Point(){
5
6     }
```

```
    Point(double x,double y){
        this.x=x;
        this.y=y;
    }

    public double EuclideanDis(Point other) {
        double distance = 0;

        distance = Math.pow((this.x - other.getX()),2) + Math.pow((this.y - other.getY()),2);

        return Math.sqrt(distance);
    }

    public double getX()
    {
        return x;
    }
    public double getY(){
        return y;
    }
}
```

kmeansMapper:

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

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.URI;
import java.util.ArrayList;
import java.util.List;

public class kmeansMapper extends Mapper<LongWritable, Text, IntWritable, Text> {

    private List<Point> means;

    /**
     * reading the data from the distributed cache
     */
    public void setup(Context context) throws IOException, InterruptedException {
        means = new ArrayList<Point>();
```

```
28
29  //
30  //        URI[] cacheFiles  = context.getCacheFiles();
31  //        BufferedReader br = new BufferedReader(new
    FileReader(cacheFiles[0].toString()));
32
33          Configuration conf = new Configuration();
34          FileSystem fs = FileSystem.get(conf);
35          FSDataInputStream hdfsInStream = fs.open(new Path("output/cache/part-r-
    00000"));
36          InputStreamReader isr = new InputStreamReader(hdfsInStream, "utf-8");
37          BufferedReader br = new BufferedReader(isr);
38
39
40          String lineString = null;
41          while((lineString = br.readLine()) != null){
42              String[] keyValue = StringUtils.split(lineString,",");
43
44              Point tmpCluster = new
    Point(Double.parseDouble(keyValue[0]),Double.parseDouble(keyValue[1]));
45              means.add(tmpCluster);
46
47          }
48          br.close();
49      }
50
51      public void map(LongWritable key, Text keyvalue, Context context) throws
    IOException, InterruptedException{
52          // ignore first line
53          if (key.get() == 0)
54              return;
55
56          String[] keyValue = StringUtils.split(keyvalue.toString(),",");
57
58          String X = keyValue[0];
59          String Y = keyValue[1];
60
61          Point tmpPoint = new Point(Double.parseDouble(X),Double.parseDouble(Y));
62
63          context.write(new IntWritable(findClosest(tmpPoint)), new Text(X + "," + Y));
64      }
65
66      /**
67       * method that returns the closest mean from the point
68       * @param value
69       * @return
70       */
71      private int findClosest(Point value){
72          int argmin = 0;
73          double minimalDistance = Double.MAX_VALUE ;
74          for(int i = 0; i<means.size(); i++){
75              Point tmpCluster = means.get(i);
76              double distance = value.EuclideanDis(tmpCluster);
```

```
            if(distance < minimalDistance){
                minimalDistance = distance;
                argmin = i;
            }
        }
        return argmin;
    }
}
```

kmeansReducer:

```
import org.apache.commons.lang.StringUtils;
import org.apache.hadoop.io.DoubleWritable;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

import java.io.IOException;

public class kmeansReducer extends Reducer<IntWritable, Text, DoubleWritable,
DoubleWritable> {

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

        double sumX = 0.0;
        double sumY = 0.0;
        int count = 0;
        for(Text value : values){
            String[] keyValue = StringUtils.split(value.toString(),",");
            sumX +=  Double.parseDouble(keyValue[0]);
            sumY +=  Double.parseDouble(keyValue[1]);
            count ++;
        }


        context.write(new DoubleWritable(sumX/count), new
DoubleWritable(sumY/count));

    }
}
```

kmeansMain:

```
import org.apache.commons.lang.StringUtils;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FSDataInputStream;
import org.apache.hadoop.fs.FSDataOutputStream;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.DoubleWritable;
```

```java
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.util.GenericOptionsParser;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.URI;
import java.net.URISyntaxException;

public class kmeansMain {

    private static final String CACHED_PATH = "output/cache";
    private static final String ACTUAL_PATH = "output/means";

    // directory store acutal result
    private static final String CACHED_MEANS = "output/cache/part-r-00000";
    private static final String ACTUAL_MEANS = "output/means/part-r-00000";


    public static void writeFileByline(String dst, String contents)  throws IOException{
        Configuration conf = new Configuration();
        Path dstPath = new Path(dst);
        FileSystem fs = dstPath.getFileSystem(conf);
        FSDataOutputStream outputStream = null;

        if (!fs.exists(dstPath)) {
            outputStream = fs.create(dstPath);
        }else{
            outputStream = fs.append(dstPath);
        }
        contents = contents + "\n";
        outputStream.write(contents.getBytes("utf-8"));
        outputStream.close();
    }

    public static int readFileByLines(String fileName,String meansPath) throws IOException {
        Configuration conf = new Configuration();
        FileSystem fs = FileSystem.get(URI.create(fileName), conf);
        FSDataInputStream hdfsInStream = fs.open(new Path(fileName));
        InputStreamReader isr = new InputStreamReader(hdfsInStream, "utf-8");
        BufferedReader br = new BufferedReader(isr);
        // get first line k
        String line =  br.readLine();
        int k = Integer.parseInt(StringUtils.split(line, ",")[0]);
        int count = 0;

        while ((line = br.readLine()) != null && count < k) {
```

```
            writeFileByline(meansPath, line);
            count++;
        }
        return k;
    }


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

        Configuration conf = new Configuration();
        String[] otherArgs = (new GenericOptionsParser(conf,
args)).getRemainingArgs();
        if (otherArgs.length < 2) {
            System.err.println("Usage: kmeans <in> <out>");
            System.exit(2);
        }

        int code = 0;

        Path inputPath = new Path(otherArgs[0]);
        Path outputDir = new Path(otherArgs[1] + "");

        Path cacheMeansPath = new Path(CACHED_MEANS);
        Path actualMeansPath = new Path(ACTUAL_MEANS);

        Path cachePath = new Path(CACHED_PATH);
        Path actualPath = new Path(ACTUAL_PATH);

        int k = readFileByLines(otherArgs[0],ACTUAL_MEANS);
        int maxIterations = 500;
        double threshold = 0.000001;


        // Delete output if exists
        FileSystem hdfs = FileSystem.get(conf);
        if (hdfs.exists(outputDir))
            hdfs.delete(outputDir, true); // recursive delete

        boolean changed = false;
        int counter = 0;

        while(!changed && counter < maxIterations){

            // Delete output if exists
            if (hdfs.exists(cachePath))
                hdfs.delete(cachePath, true);
            //moving the previous iteration file to the cache directory
            hdfs.rename(actualPath, cachePath);

            conf.set("threshold", threshold+"");
            //passing K to the map reduce as a parameter
            conf.set("k", k+"");
```

```
111             conf.set("mapreduce.output.textoutputformat.separator", ",");
112 
113 
114             Job kmeans = Job.getInstance(conf, "Kmeans "+ (counter + ""));
115 
116             // add cache
117             kmeans.addCacheFile(cacheMeansPath.toUri());
118 
119             kmeans.setJarByClass(kmeansMapper.class);
120             FileInputFormat.addInputPath(kmeans, inputPath);
121             // set out put path : output/means
122             FileOutputFormat.setOutputPath(kmeans, actualPath);
123 
124             kmeans.setMapperClass(kmeansMapper.class);
125             kmeans.setMapOutputKeyClass(IntWritable.class);
126             kmeans.setMapOutputValueClass(Text.class);
127 
128             kmeans.setReducerClass(kmeansReducer.class);
129             kmeans.setOutputKeyClass(DoubleWritable.class);
130             kmeans.setOutputValueClass(DoubleWritable.class);
131 
132             // Execute job
133             code = kmeans.waitForCompletion(true) ? 0 : 1;
134 
135             //checking if the mean is stable
136             BufferedReader file1Reader = new BufferedReader(new
    InputStreamReader(hdfs.open(cacheMeansPath)));
137             BufferedReader file2Reader = new BufferedReader(new
    InputStreamReader(hdfs.open(actualMeansPath)));
138             for(int i = 0; i<k; i++){
139                 String[] keyValue1 = file1Reader.readLine().split(",");
140                 String[] keyValue2 = file2Reader.readLine().split(",");
141 
142                 Point p1 = new
    Point(Double.parseDouble(keyValue1[0]),Double.parseDouble(keyValue1[1]));
143                 Point p2 = new
    Point(Double.parseDouble(keyValue2[0]),Double.parseDouble(keyValue2[1]));
144 
145                 if(p1.EuclideanDis(p2) <= threshold){
146                     changed = true;
147                 }else{
148                     changed = false;
149                     break;
150                 }
151             }
152             file1Reader.close();
153             file2Reader.close();
154             counter++;
155             System.out.println("KMEANS finished iteration:>> "+counter + " || means
    stable: "+ changed);
156 
157         }
158 
```

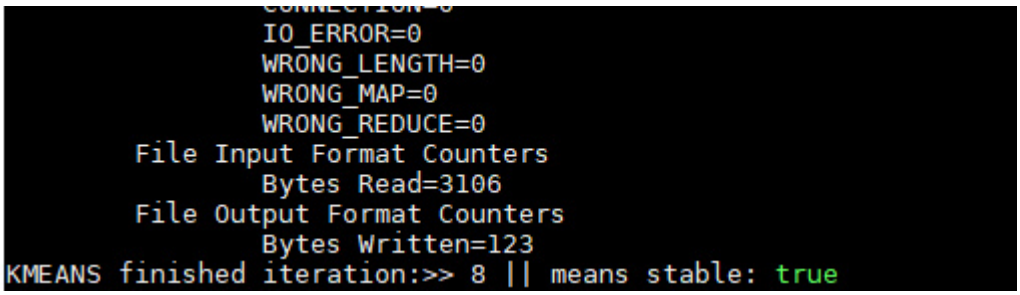
```
159
160             hdfs.rename(actualPath, outputDir);
161
162             System.exit(code);
163
164         }
165
166     }
```

运行

同样的方法，打包成`JAR`包运行，shell input:

```
1  hadoop jar Kmeans.jar input/k-means.dat output/kmeans
```

output:



查看运行结果，也就是中心点, shell input:

```
1  hdfs dfs -cat output/kmeans/*
```

output: