# 中山大学数据科学与计算机学院
# 移动信息工程专业-云计算应用开发
# 本科生项目报告

**（2016 学年秋季学期）**

课程名称：Cloud Computing　　任课老师：吴维刚　　组号：105

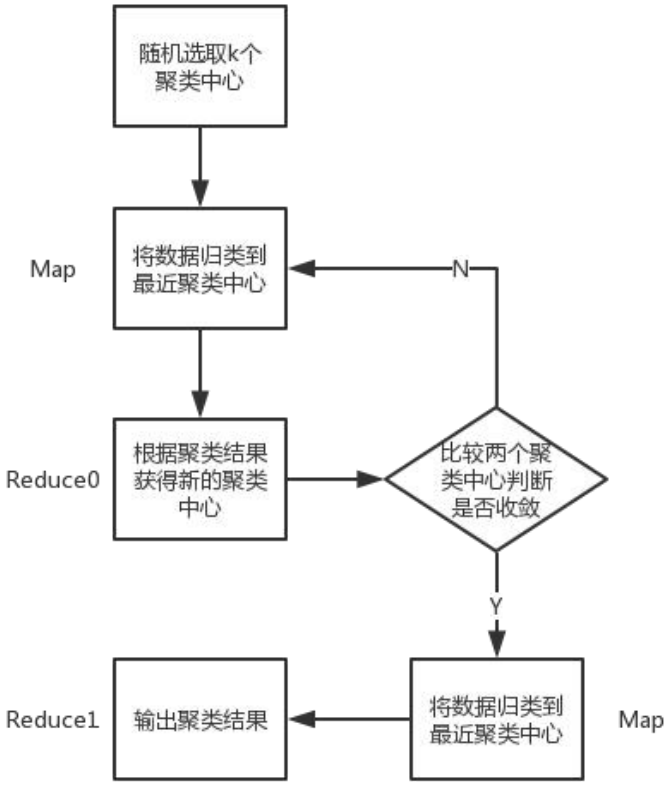
| | | | |
|---|---|---|---|
| 组员 1 姓名 | **谢子聪** | 组员 1 学号 | 14353336 |
| 组员 2 姓名 | **余珂** | 组员 2 学号 | 14353373 |
| 组员 3 姓名 | **郑铠奇** | 组员 3 学号 | 14353421 |

## 一、 问题场景

k 均值聚类主要以距离作为评价指标用来数据聚集。在对大规模数据进行 k 均值聚类的时候，单台计算机的处理能力是很大的瓶颈。但是 k 均值聚类有两个特点，一个是在分类的时候，单条数据只与中心作计算，而与其他数据无关，二是选取新的中心时用的是均值。这两个特点就使得 k 均值聚类可以使用 mapreduce 来进行并行计算。并行计算中，聚类中心的信息作为全局变量，可以让聚类归属分配时不需要占用带宽，并且利用 combine 先进行同一聚类的归并，可以减少 reduce 中的计算量。

## 二、 算法思路

### 1. 流程图




上图中两个 Map 是一样的，都是对单条数据进行归类。而 Reduce0 输出的是 k 个聚类中心，Reduce1 输出的是每条数据属于哪个中心。

## 2. 算法思路

**1） 随机选取 k 个聚类中心。**

**2） Map 阶段：根据聚类中心对数据进行聚类。**

使用 setup 函数（初始化，map 阶段只执行一次）我们可以从中心文件中得到聚类中心。

输入输出关系如下：

(Object, Text) -> (IntWritable, Point)

因为输入是从文本中逐行读取 Text，所以 key 无所谓。当然也可以先将其转为 sequence 文件中，这样就可以使用自定义的键值对。

然后将输入值:Text 转为 Point，Point 是一个自定义类，保存数据信息。

输出键 IntWriteble 是聚类中心的编号。利用 mapreduce 的自动归并，这样属于相同聚类中心的数据就会合并在一起。

**3） Combine 阶段：在节点先计算一次中心，用来压缩数据量。**

因为 k 均值聚类计算中心是均值，所以我们可以先在节点计算这部分数据的中心，然后再合并。这样就可以缓解 Reduce 阶段的运算压力。这个阶段键值对的类型不变。

**4） Reduce 阶段：每个节点的中心再计算中心。**

为了方便对数据进行读取，输出键值对的类型是(Text, IntWritable)。

Text 是聚类中心的坐标，IntWritable 是编号。

**5） 判断是否收敛。**一次 mapreduce 完成后，与上一次的聚类中心进行比较，如果没有收敛，则跳到 2)继续进行聚类。

**6） 如果收敛，则执行一个新的 mapreduce 获得所需的输出。**同样使用 2)的 map。但 reduce 阶段就直接输出(Text, IntWritable)，即数据和其对应的中心编号作为结果即可。

# 三、 算法关键代码

1） Point 类。保存一条数据，n 维的点。实现从文件流读取，与 Text 相互转换，设置值，计算距离，做加法乘法等操作。

```
public class Point implements Writable {
    public ArrayList<Double> values;
    public static int length; // 维数
    public void clear() { // 初始化或清空 }
    Nice！！！Cool！！
    public void write(DataOutput out) throws IOException { }
    public void readFields(DataInput in) throws IOException { }
    public void readText(Text text) { // 从 Text 中读入 }
    public double getDistance(Point p) { // 计算点之间的距离 }
    public void plus(Point p) { // 点相加 }
    public void mul(double value) { // 点乘以一个数 }
    public String toString() { // 将点坐标格式化为 String }
    public Text toText() { // 转为 Text }
}
```

2）一些全局变量。中心的个数，数据的维数，输入输出，中心文件的地址等

```
public class Clustering {
        private static int centerNum = 5; // 分类数
        private static int dim = 2; // 维数
        private static String namePath = "hdfs://192.168.15.128:9000";
        private static String dataPath = "/user/clustering/five_cluster";
        private static String outputPath = "/user/clustering/output";
        private static String centerPath = "/user/clustering/center";
        private static String resultName = "/part-r-00000";
```

3）初始化配置。设置 Configuration，JobConfig，Point 的维度以及随机选取中心。

```
        System.setProperty("hadoop.home.dir", "C:\\hadoop-2.6.0\\");
        conf = new Configuration();
        conf.set("fs.defaultFS", "hdfs://192.168.15.128:9000");
        FileSystem fs = FileSystem.get(URI.create(namePath), conf);
        JobConf jobConf = new JobConf();
        jobConf.setJarByClass(Clustering.class);
```

```
        jobConf.setMapOutputKeyClass(IntWritable.class);
        jobConf.setMapOutputValueClass(Point.class);
        jobConf.setOutputKeyClass(Text.class);
        jobConf.setOutputValueClass(IntWritable.class);
        jobConf.set("fs.defaultFS", "hdfs://192.168.15.128:9000");
        Point.length = dim; // 设置点的维度
        randomCenters(); // 随机设置中心到 output
```

其中随机选取中心 randomCenters 的函数实现：利用 set 的唯一性，得到 k 个不同的随机数：

```
// 随机选择中心
public static void randomCenters() throws IOException{
        ArrayList<Point> points = getPoints(dataPath);
        int size = points.size();
        HashSet<Integer> set = new HashSet<>();
        Random r = new Random();
        while (set.size() < centerNum) //取中心数个不同的随机数
                set.add(r.nextInt(size));
        FileSystem fs = FileSystem.get(URI.create(namePath), conf);
        Path file = new Path(outputPath + resultName);
        deletePath(fs, file); //删除原来的中心文件
        OutputStream os = fs.create(file);
        BufferedWriter br = new BufferedWriter(new
                                OutputStreamWriter(os, "UTF-8"));
        for (int index : set) { // 将中心写入文件
                br.write(points.get(index).toText().toString());
                br.write('\n');
        }
        br.close();
}
```

其中 getPoints(path)，从文件中获得点坐标集，这个函数在代码中**多次用到**。

```
// 获得点坐标集
public static ArrayList<Point> getPoints(String path) throws IOException {
        FileSystem fs = FileSystem.get(URI.create(namePath), conf);
        FSDataInputStream is = fs.open(new Path(path));
        LineReader lineReader = new LineReader(is, conf);
        Text line = new Text();
        ArrayList<Point> points = new ArrayList<>();
        while (lineReader.readLine(line) > 0) {
                Point point = new Point();
```

```
                point.readText(line);
                points.add(point);
        }
        lineReader.close();
        return points;
}
```

4）循环进行 k 均值聚类直到收敛

```
        while (true) {
                // 拷贝 output 到 center
                FileUtil.copy(fs, new Path(outputPath + resultName), fs,
                        new Path(centerPath), false, true, conf);
                // 删除 output
                deletePath(fs, new Path(outputPath));
                Job job = Job.getInstance(jobConf);
                job.setMapperClass(ClassifierMapper.class);
                job.setCombinerClass(ClusterCombiner.class);
                job.setReducerClass(CenterReducer.class);
                FileInputFormat.addInputPath(job, new Path(dataPath));
                FileOutputFormat.setOutputPath(job, new
                                                Path(outputPath));
                // 执行并等待结束返回 true
                if (job.waitForCompletion(true)) {
                        if (convergence()) break; // 判断是否收敛
                }
        }
```

Map 的实现：setup 获得中心点，map 进行分类。

```
// Mapper 归类每一个点
public static class ClassifierMapper extends Mapper<Object, Text,
                                                IntWritable, Point> {
        private ArrayList<Point> centers; // 中心
        public void setup(Context context)
                        throws IOException, InterruptedException {
                super.setup(context);
                centers = getPoints(centerPath); // 从中心文件中获取中心
        }
        // map 将点归类到最近的中心上
        public void map(Object key, Text value, Context context) throws
                                IOException, InterruptedException {
                double minDist = Double.MAX_VALUE;
```

```
                int id = 0;
                Point point = new Point();
                point.readText(value);
                for (int i = 0; i < centerNum; i++) {
                        double dist = point.getDistance(centers.get(i));
                        if (dist < minDist) {
                                minDist = dist;
                                id = i;
                        }
                }
                context.write(new IntWritable(id+1), point);
        }
}
```

Combiner：归并后计算中心，输出的键值对格式不变。

```
// Combiner 先在节点计算一次中心
public static class ClusterCombiner extends Reducer<IntWritable, Point,
                                                IntWritable, Point> {
        public void reduce(IntWritable key, Iterable<Point> points,
                                Context context) throws IOException,
                                                InterruptedException {
                ArrayList<Double> values = new
                                ArrayList<>(Collections.nCopies(dim, 0.0));
                Point center = new Point();
                center.values = values;
                int cnt = 0;
                for (Point point : points) {
                        center.plus(point);
                        ++cnt;
                }
                if (cnt > 0) center.mul(1.0/cnt); // 获得中心
                context.write(key, center);
        }
}
```

Reducer：大体同 Combiner，只是最后输出格式不一样”中心坐标 编号”：

```
                context.write(center.toText(), key);
```

判断是否收敛的函数：

```
// 判断是否收敛
```

```
public static boolean convergence() throws IOException {
        ArrayList<Point> oldCenters = getPoints(centerPath);
        ArrayList<Point> newCenters = getPoints(outputPath +
                                                resultName);
        for (int i = 0; i < centerNum; i++) {
                double dist = oldCenters.get(i).getDistance(
                                        newCenters.get(i));
                if (dist > 1e-6) return false;
        }
        return true;
}
```

5）收敛后，再执行一次 mapreduce 聚类获得需要的输出

```
        // 收敛后，根据 center 进行聚类（使用 map-reduce）
        // 目前是直接保存点的坐标
        // 改进方法：可以给每条数据增加编号，也可以 map 的 value 直接保存为 Text
        // 这样就能进行真正意义上的 Combiner 和 Reducer 来提高效率和节省空间
        // 但是为了展示结果方便就不这样写了
        Job job = Job.getInstance(jobConf);
        job.setMapperClass(ClassifierMapper.class);
        job.setReducerClass(PointCenterReducer.class);
        job.setOutputKeyClass(Text.class);
        FileInputFormat.addInputPath(job, new Path(dataPath));
        FileOutputFormat.setOutputPath(job, new Path(outputPath));
        deletePath(fs, new Path(outputPath));
        job.waitForCompletion(true);
```

其中 reducer 就是直接格式化后输出，输出格式为”点坐标 中心编号“：

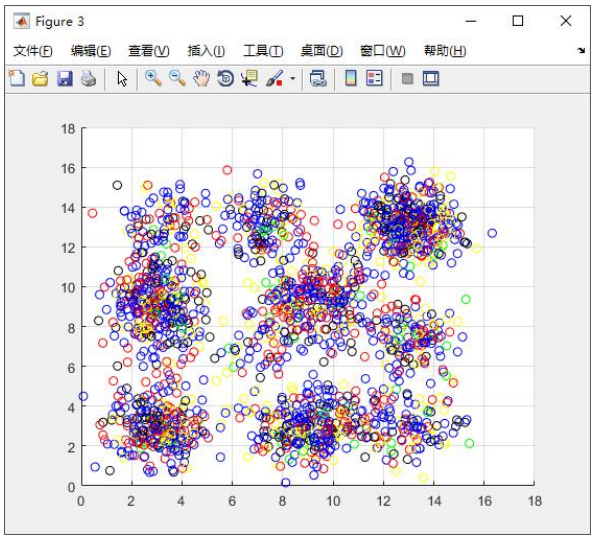
```
// Reducer 聚类
public static class PointCenterReducer extends Reducer<IntWritable,
                                                Point, Text, Text> {
        public void reduce(IntWritable key, Iterable<Point> points,
                                Context context) throws IOException,
                                                InterruptedException {
                StringBuilder builder = new StringBuilder();
                for (Point point : points) {
                        builder.append(point.toString())
                                .append(key).append('\n');
                 }
                context.write(new Text(""), new
                                Text(builder.toString()));
```

```
        }
}
```

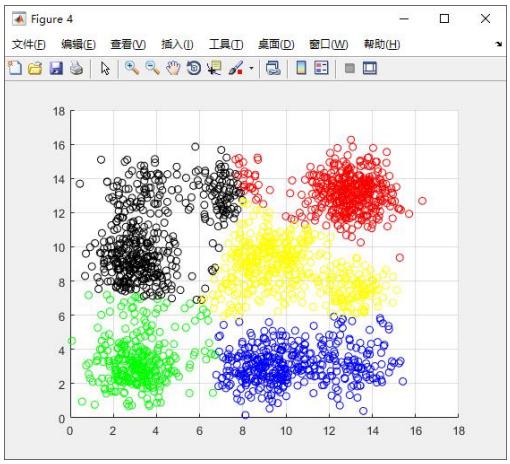
# 四、 结果展示

1. 用 matlab 对 mapreduce 进行 k 均值聚类前后的数据集绘图如下。

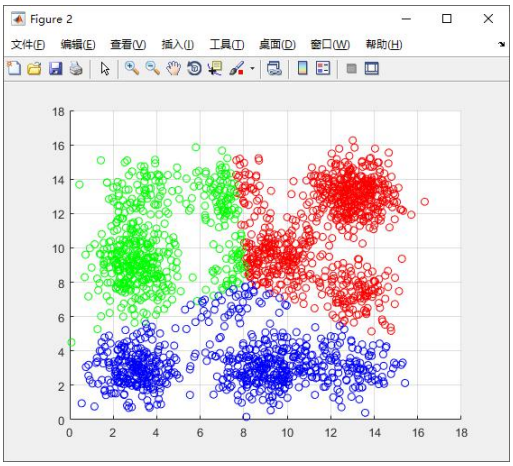
数据集 five_cluster:




K 为 5 的聚类结果：




K 为 3 的聚类结果：




2. mapreduce 的输出信息展示

```
2016-12-29 21:42:16,617 INFO [org.apache.hadoop.mapred.Merger] - Merging 1 sorted segments
2016-12-29 21:42:16,617 INFO [org.apache.hadoop.mapred.Merger] - Down to the last merge-pass, with 1 segments left of total size: 62 b
2016-12-29 21:42:16,617 INFO [org.apache.hadoop.mapred.LocalJobRunner] - 1 / 1 copied.
2016-12-29 21:42:16,637 INFO [org.apache.hadoop.mapred.Task] - Task:attempt_local1835152195_0012_r_000000_0 is done. And is in the pro
2016-12-29 21:42:16,638 INFO [org.apache.hadoop.mapred.LocalJobRunner] - 1 / 1 copied.
2016-12-29 21:42:16,638 INFO [org.apache.hadoop.mapred.Task] - Task attempt_local1835152195_0012_r_000000_0 is allowed to commit now
2016-12-29 21:42:16,642 INFO [org.apache.hadoop.mapreduce.lib.output.FileOutputCommitter] - Saved output of task 'attempt_local1835152
2016-12-29 21:42:16,642 INFO [org.apache.hadoop.mapred.LocalJobRunner] - reduce > reduce
2016-12-29 21:42:16,642 INFO [org.apache.hadoop.mapred.Task] - Task 'attempt_local1835152195_0012_r_000000_0' done.
2016-12-29 21:42:16,642 INFO [org.apache.hadoop.mapred.LocalJobRunner] - Finishing task: attempt_local1835152195_0012_r_000000_0
2016-12-29 21:42:16,642 INFO [org.apache.hadoop.mapred.LocalJobRunner] - reduce task executor complete.
2016-12-29 21:42:17,513 INFO [org.apache.hadoop.mapreduce.Job] - Job job_local1835152195_0012 running in uber mode : false
2016-12-29 21:42:17,513 INFO [org.apache.hadoop.mapreduce.Job] -  map 100% reduce 100%
2016-12-29 21:42:17,513 INFO [org.apache.hadoop.mapreduce.Job] - Job job_local1835152195_0012 completed successfully
2016-12-29 21:42:17,515 INFO [org.apache.hadoop.mapreduce.Job] - Counters: 38
        File System Counters
                FILE: Number of bytes read=8368
                FILE: Number of bytes written=5966368
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=927764
                HDFS: Number of bytes written=2602
                HDFS: Number of read operations=491
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=188
        Map-Reduce Framework
                Map input records=2000
                Map output records=2000
                Map output bytes=40000
                Map output materialized bytes=72
                Input split bytes=120
                Combine input records=2000
                Combine output records=3
                Reduce input groups=3
                Reduce shuffle bytes=72
                Reduce input records=3
                Reduce output records=3
                Spilled Records=6
                Shuffled Maps =1
                Failed Shuffles=0
                Merged Map outputs=1
                GC time elapsed (ms)=0
                CPU time spent (ms)=0
                Physical memory (bytes) snapshot=0
```

# 五、 遇到的问题与解决

1. eclipse 的版本问题。neon 版将 hadoop 插件放入 plugins 里无效。需要使用 mars 版代替。

2. windows 下编译运行一大堆错误，大部分错误在下面这个网址里得到了解决方法：http://www.bkjia.com/ASPjc/931209.html

3. 默认访问的是本地的文件系统，解决方法：在代码里指定下：

Conf.set("fs.defaultFS", "fdfs://~~~")

4. 在 java 里写 hdfs 的文件操作相当痛苦。解决方法：搜索，然后看官方文档，有些操作其实已经集成了，比如文件的转移，删除，拷贝等。