云计算项目设计
K-means 算法实现
实验报告

姓名：宋晓彤

学号：16340192

班级：16 级软工 5 班

目录

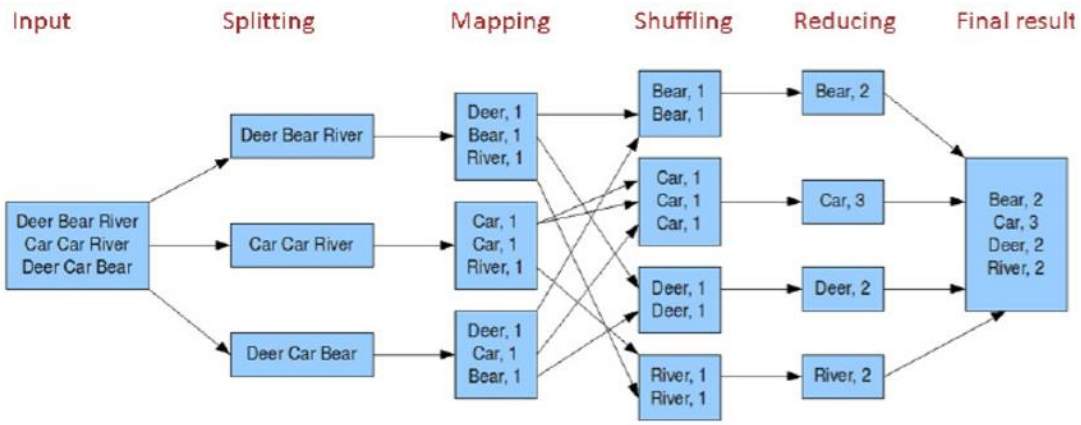
一、 实验目的

基于云平台的 Hadoop 分布式系统基础架构，用 MapReduce 的编程模型完成 K-means 算法程序的分布式运行，并验证正确。

二、 实验原理

1. Hadoop

Hadoop 框架中最核心设计就是：HDFS 和 MapReduce。HDFS 提供了海量数据的存储,MapReduce 提供了对数据的计算。Hadoop 为每一个 input split 创建一个 task 调用 Map 计算，在此 task 中依次处理此 split 中的一个个记录(record),map 会将结果以 key--value 的形式输出，hadoop 负责按 key 值将 map 的输出整理后作为 Reduce 的输入,Reduce Task 的输出为整个 job 的输出，保存在 HDFS 上。



2. kmeans 算法

K-means 算法是最为经典的基于划分的聚类方法，是十大经典数据挖掘之一。K-means 算法的基本思想是：以空间中 C 个点为形心进行聚类，对最靠近他们的对象归类。通过迭代的方法，逐次更新各簇的形心的值，直至得到最好的聚类结果。(形心可以是实际的点、或者是虚拟点)

首先从 C 个数据对象任意选择 k 个对象作为初始聚类中心；而对于所剩下其它对象，则根据它们与这些聚类中心的相似度（距离），分别将它们分配给与其最相似的（聚类中心所代表的）聚类；然后再计算每个所获新聚类的聚类中心（该聚类中所有对象的均值）；不断重复这一过程直到标准测度函数开始收敛为止。一般都采用均方差作为标准测度函数. k 个聚类具有以下特点：各聚类本身尽可能的紧凑，而各聚类之间尽可能的分开。

三、 实验设计

1. K-means 算法设计

K-means 的实现流程我们可以用下面的流程图表示

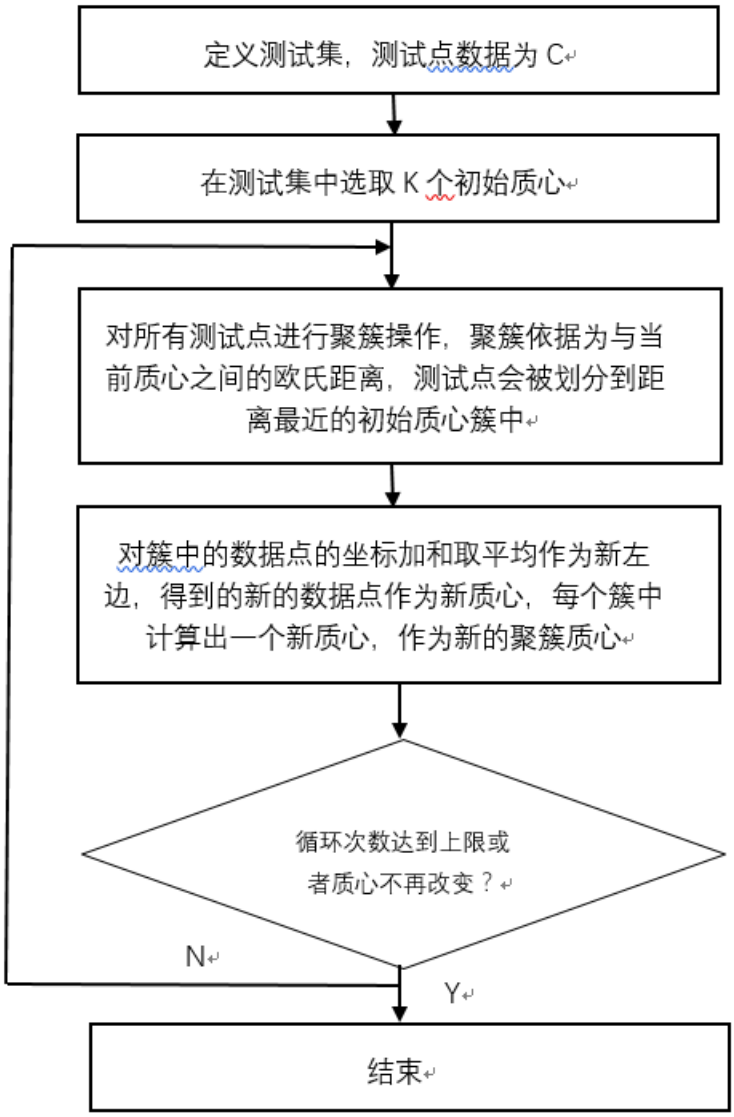
(1) 首先定义一个测试集，其中为所有的测试点，共 C 个。为了缩短实验时间，在设定时，我选择只设定 5 个初始点。

(2) 选择了前 K 个点做为初始质心，对剩下 C-K 个点进行了聚簇操作，聚出 K 个簇，同样出于缩短实验时间，便于检验算法正确的打算，选择选取两个点作为初始质心。

(3) 对 2 个簇中的点分别取平均值点，作为新的质点。

(4) 用得到的新质点作为聚簇的依据，对 5 个点重新进行聚簇，同时循环此操作。

(5) 当循环次数达到预设，或者每个簇中的测试点不再发生变化，即得到的新质点与上一次的老质点没有区别，那么表示聚簇完成，结束程序运行。



2. Hadoop 分布式计算设计

整个实验过程可以分成 3 个阶段：

（1） map 函数的输入为数据块的集合，输入格式为<key1，value1>，其中 key1 为对象 id，value1 为数据对象向量。Map 函数最主要的功能就是将本节点上的数据对象划到离其最近的簇向量中去，此时将有可能改变簇中心向量，所以需要重新计算簇中心，其输出的格式也是<key2，value2>，其中 key2 为簇向量标识符，value2 为数据对象向量。

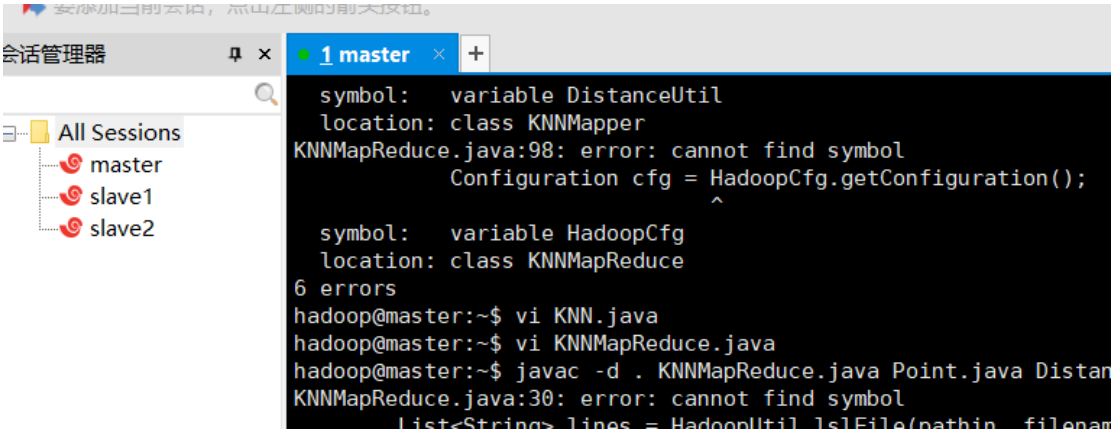
（2） Combine 函数的输入为 Map 所输出的<key2，value2>，其函数输出为<key3，value3>，key3 仍是簇类向量标识符，而 value3 则是相同 key3 的所有向量组合和这些向量的数目。

（3） Reduce 函数处理输入同一个簇的所有数据对象向量，然后重新生成新的簇类中心向量，其输入输出均为键值对形式，输入信息是各个子节点的 combine 结果，输出信息是簇类标识符和新的簇类中心向量。在每次 Reduce 执行完成后，都需要将新的簇类中心向量与前一次的簇类中心向量进行比较，当满足聚类准则函数时将停止迭代即此时聚类已收敛，输出最终结果。

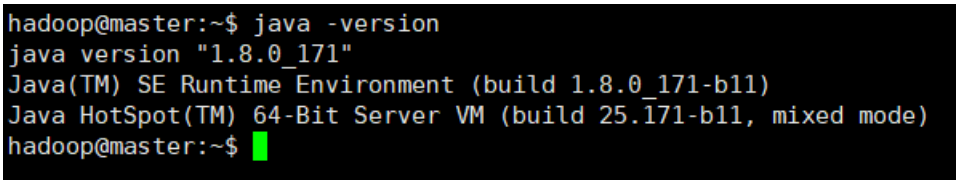
四、 环境配置

在实验过程中，根据 lab2，lab3 进行环境的配置

1. 完成三台虚拟机 master、slave1、slave2 的创建，并使用 xshell 连接；



2. 完成三台虚拟机之间的 ping 通，安装及配置 jdk 和 hadoop 环境，同时启动及验证。



```
hadoop@master:~$ javac -version
javac 1.8.0_171
hadoop@master:~$
```

```
hadoop@master:~$ jps
3504 Jps
2370 NodeManager
2053 SecondaryNameNode
1785 NameNode
2251 ResourceManager
1901 DataNode
hadoop@master:~$
```

```
hadoop@slave1:~$ jps
1877 NodeManager
1941 Jps
1375 DataNode
hadoop@slave1:~$
```

```
hadoop@slave2:~$ jps
1540 DataNode
2345 NodeManager
2461 Jps
hadoop@slave2:~$
```

五、 代码设计

1. **根据质心进行聚类**：将训练集的数据读入，通过 split 划分，然后计算每个训练样本，也即是每一行与中心点的距离，在本次实验中我们使用的是欧式距离。在计算得出每个样本与所有中心的差距的过程中，求得最小距离，并记录与之相对应的中心点序号。将此距离样本最近的中心点序号和样本以键值对的形式作为 map 的结果输出。

```
// do the cluster
public static class KmeansMapper extends
        Mapper<Object, Text, IntWritable, Text> {
    public void map(Object key, Text value, Context context)
            throws IOException, InterruptedException {
        String line = value.toString();
        String[] fields = line.split(",");

        List<ArrayList<Double>> mycenters = myutils.GetOldCenters(context
            .getConfiguration().get("centersPath"));
        // the first index
        int beginindex = Integer.parseInt(context.getConfiguration().get("BeginIndex"));
        int K = Integer.parseInt(context.getConfiguration().get("KPath"));

        double mindis = 100000;
        int centerIndex = K;
        for (int i = 0; i < K; i++) {
            double nowdis = 0;
            // calculate the distance
            for (int i = beginindex; i < fields.length; i++)
```

2. **根据新的聚类结果得到新质点**：将 values 中的取值划分转化为字符串存入数组。然后进行计算，算出这个类新的中心点，然后将这个值也转化为字符串。然后将字符串和对应的键生成的序号输出到结果文件。

```
// conclude the operations and prepare for the new recycle
public static class KmeansReducer extends Reducer<IntWritable, Text, IntWritable, Text> {
    public void reduce(IntWritable key, Iterable<Text> values,
            Context context) throws IOException, InterruptedException {
        // use the myutilslist record all the records
        List<ArrayList<Double>> myutilsList = new ArrayList<ArrayList<Double>>();
        String tmpResult = "";
        for (Text val : values)
        {
            String line = val.toString();
            String[] fields = line.split(",");
            List<Double> tmpList = new ArrayList<Double>();
            for (int i = 0; i < fields.length; i++) {
                tmpList.add(Double.parseDouble(fields[i]));
            }
            myutilsList.add((ArrayList<Double>) tmpList);
        }
```

3. **记录历次迭代旧的中心点：**通过 FSDataInputStream 获得服务器上文件的内容流，然后通过 LineReader 一行一行地得出字符串。通过使用 splite 函数来将字符串划分从而取得多维的数据，存入 oldcenters 列表当中。

```
// get the old center to compare
    public static List<ArrayList<Double>> GetOldCenters(String inputPath) {
        // all of the centers in each time
        List<ArrayList<Double>> oldcenters = new ArrayList<ArrayList<Double>>();
        Configuration conf = new Configuration();
        try {
            FileSystem hdfs = FileSystem.get(conf);
            Path inPath = new Path(inputPath);
            FSDataInputStream fsIn = hdfs.open(inPath);
            LineReader lineIn = new LineReader(fsIn, conf);
            Text line = new Text();
            while (lineIn.readLine(line) > 0) {
                String readl = line.toString();
                // the test number
                String[] fields = readl.split(",");
                List<Double> tmpList = new ArrayList<Double>();
                for (int i = 0; i < fields.length; i++)
```

4. **删除旧的无用质点**

```
// delete the old center
    public static void delete(String path) {
        Configuration conf = new Configuration();
        try {
            FileSystem hdfs = FileSystem.get(conf);
            Path inPath = new Path(path);
            hdfs.delete(inPath);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
// get the file and change the center
```

5. **判断程序是否结束：**通过判断两次迭代的中心点是否相同来判断此时的聚类结果是否已经收敛，通过计算两次迭代产生的质点之间的距离的大小来判断两次迭代的质点是否已经趋于稳定。

```
// judge if the algorithm is over
    public static boolean isFinished(String oldPath, String newPath,
            String inputK, String BeginIndex, double threshold)
            throws IOException {
        // get the old centers to compare
        List<ArrayList<Double>> oldCenters = myutils.GetOldCenters(oldPath);
        // define the list of the newcenters
        List<ArrayList<Double>> newCenters = new ArrayList<ArrayList<Double>>();
        int beginindex = Integer.parseInt(BeginIndex);
        int K = Integer.parseInt(inputK);
        Configuration conf = new Configuration();
        FileSystem hdfs = FileSystem.get(conf);
        // do k times to get the new center
        for (int i = 0; i < K; i++)
        {
            Path refreshpath = new Path(newPath + i);
            if (!hdfs.exists(refreshpath))
```

六、 运行过程

1. 查看文件系统

Browse Directory

/Kmeans

Permission	Owner	Group	Size	Replication	Block Size	Name
drwxr-xr-x	hadoop	supergroup	0 B	0	0 B	input
drwxr-xr-x	hadoop	supergroup	0 B	0	0 B	newcenter
drwxr-xr-x	hadoop	supergroup	0 B	0	0 B	output

Browse Directory

/Kmeans/input

Permission	Owner	Group	Size	Replication	Block Size	Name
-rw-r--r--	hadoop	supergroup	22 B	3	128 MB	k.data
-rw-r--r--	hadoop	supergroup	8 B	3	128 MB	oldcenter.data

Browse Directory

/Kmeans/output

Permission	Owner	Group	Size	Replication	Block Size	Name
-rw-r--r--	hadoop	supergroup	0 B	3	128 MB	_SUCCESS
-rw-r--r--	hadoop	supergroup	18 B	3	128 MB	part-r-00000
-rw-r--r--	hadoop	supergroup	26 B	3	128 MB	part-r-00001

2. 编译 java 文件后打包

```
hadoop@master:~$ ls
data                  id_rsa.pub                  Kmeans.jar          Point.java
DistanceType.java     jdk-8u171-linux-x64.tar.gz  Kmeans.java         setenv.sh
hadoop-2.6.0.tar.gz   Kmeans                      KNNMapReduce.java
```

3. 运行程序得到结果

（1） 检测数据

```
hadoop@master:~$ /usr/local/hadoop/bin/hdfs df
0,2
0,0
1.5,0
5,0
5,2
hadoop@master:~$
```

（2） 初始质心

```
hadoop@master:~$ /usr/local/hadoop/b
0,0
5,0
hadoop@master:~$
```

（3） 第一次迭代

```
迭代次数： 1
18/06/17 18:20:18 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
^Z
[2]+  Stopped                 /usr/local/hadoop/bin/hadoop jar Kmeans.jar Kmeans /Kmeans
/input /Kmeans/output oldcenter.data /Kmeans/newcenter/oldcenter.data /Kmeans/output/par
t-r-0000 0 2
hadoop@master:~$ /usr/local/hadoop/bin/hadoop jar Kmeans.jar Kmeans /Kmeans/input /Kmean
s/output /usr/local/hadoop/oldcenter.data /Kmeans/newcenter/oldcenter.data /Kmeans/outpu
t/part-r-0000 0 2
迭代次数： 1
18/06/17 18:21:05 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
18/06/17 18:21:09 INFO input.FileInputFormat: Total input paths to process : 2
18/06/17 18:21:10 INFO mapreduce.JobSubmitter: number of splits:2
```

（4） 第二次迭代

```
迭代次数： 2
18/06/17 18:24:35 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
18/06/17 18:24:36 INFO input.FileInputFormat: Total input paths to process : 2
18/06/17 18:24:37 INFO mapreduce.JobSubmitter: number of splits:2
18/06/17 18:24:37 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1529208868
708_0003
18/06/17 18:24:38 INFO impl.YarnClientImpl: Submitted application application_1529208868
708_0003
18/06/17 18:24:38 INFO mapreduce.Job: The url to track the job: http://master:8088/proxy
/application_1529208868708_0003/
18/06/17 18:24:38 INFO mapreduce.Job: Running job: job_1529208868708_0003
18/06/17 18:25:01 INFO mapreduce.Job: Job job_1529208868708_0003 running in uber mode :
false
18/06/17 18:25:01 INFO mapreduce.Job:  map 0% reduce 0%
18/06/17 18:25:35 INFO mapreduce.Job:  map 50% reduce 0%
18/06/17 18:25:38 INFO mapreduce.Job:  map 100% reduce 0%
```

（5） 检验结束

```
距离： 0.0
18/06/17 18:26:33 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
18/06/17 18:26:33 INFO input.FileInputFormat: Total input paths to process : 2
18/06/17 18:26:33 INFO mapreduce.JobSubmitter: number of splits:2
18/06/17 18:26:34 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_152920886
708_0004
18/06/17 18:26:34 INFO impl.YarnClientImpl: Submitted application application_152920886
708_0004
18/06/17 18:26:34 INFO mapreduce.Job: The url to track the job: http://master:8088/prox
/application_1529208868708_0004/
18/06/17 18:26:34 INFO mapreduce.Job: Running job: job_1529208868708_0004
18/06/17 18:26:58 INFO mapreduce.Job: Job job_1529208868708_0004 running in uber mode :
false
18/06/17 18:26:58 INFO mapreduce.Job:  map 0% reduce 0%
18/06/17 18:27:29 INFO mapreduce.Job:  map 50% reduce 0%
```

（6） 结果质心

```
hadoop@master:~$ /usr/local/hadoop/b
0.375,0.5
5.0,0.6666666666666666
hadoop@master:~$
```

七、 心得体会

本次实验过程中，我初步了解了 Hadoop 这个存储平台，重点在于 HDFS 分布式文件系统以及 MapReduce 模型的使用。

我认为分布式最大的好处其实是让许多机器跑同一份代码，大大提高了运行的效率，虽然这次实验的时候并没有用很大的数据去检测，只是用小数据进行了算法正确性的检验，但是在进行 lab4 的 wordcount 实验的时候还是有了一些体验的。

说回本次实验，在 HDFS 分布式文件系统中有两个概念是需要清楚的，一是 NameNode，它记录的是文件在 DataNode 的位置信息和元数据信息，二是 DataNode，即存储管理者，支持一次写入，多次读取，程序是被划分为多个大小为 64M 的数据块，分步并冗余存放在各个 DataNode 节点中的。

在开启 DataNode 服务的时候还因为多次格式化导致无法开启该服务，后来查了许多博客之后发现是因为在格式化的时候刷新了 NameNode 导致不匹配，所以开启不成功，只要更改一下 NameNode 的编码，然后在

下次进行格式化的时候注意在选择的时候不要选 Y 就行了。

K-means 聚类算法也是在网上搜索很久，一开始不是很理解但是后来发现这种算法可以让机器通过聚类程序对某种数据进行递归性质的归类，达到智能的识别功能。

总的来说，通过这次实验，以及之前做的 lab4 的 wordcount 实验，我大概了解了分布式计算的过程，尤其是实现 MapReduce 的过程，但是其实了解得不够透彻，也比较依赖网上的资料和经验，这是比较不足的地方，但是对 map 和 reduce 的运行过程和原理还是有了一定的了解，这是学到的一些收获。