《大数据系统课程设计》

实习报告

班级 191211

学号 20211003686

姓名 刘勇

指导老师 李程俊

完成日期 2023/9/2

一、实验内容

本次课设需要完成矩阵的预处理和矩阵的乘法。

其中，矩阵的预处理在网上无法搜索到相同的方法，所以更加重要。

x1	x2	x3	x4	x5	x6
x1	x2	x3	x4	x5	x6
x1	x2	x3	x4	x5	x6
x1	x2	x3	x4	x5	x6
x1	x2	x3	x4	x5	x6
x1	x2	x3	x4	x5	x6

任意一个值，与它周边一圈的值的均值，差异不能过大。否则，则用均值代替这个值。请考虑用 MapReduce 实现。

二、算法设计

2.1 矩阵存储

矩阵的存储方式，有两种，采用原始表示方式，就是矩阵是什么样子，就保存成什么样子，比如

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 0 \\ 6 & 7 & 8 \end{pmatrix}$$

矩阵

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 0 \\ 6 & 7 & 8 \end{pmatrix}$$

存储方式

还有一种，就是以稀疏矩阵存储，仅存储非零值，比如

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 0 \\ 6 & 7 & 8 \end{pmatrix}$$

矩阵

行 列 值

$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 2 \\ 1 & 3 & 3 \\ 2 & 1 & 4 \\ 2 & 2 & 5 \\ 3 & 1 & 6 \\ 3 & 2 & 7 \\ 3 & 3 & 8 \end{pmatrix}$$

存储方式

在上图中，第一种方式存储的数据量小于第二种，但这只是因为例子中的数

据设计成这样。在现实中，使用分布式计算矩阵乘法的环境中，大部分矩阵是稀疏矩阵，且数据量极大，在这种情况下，第二种数据结构的优势就显现了出来。因此，我们采用第二种方法存储矩阵。

2.2 矩阵乘法

设$A=(a_{ij})_{mn}$，$B=(b_{jk})_{nl}$，那么

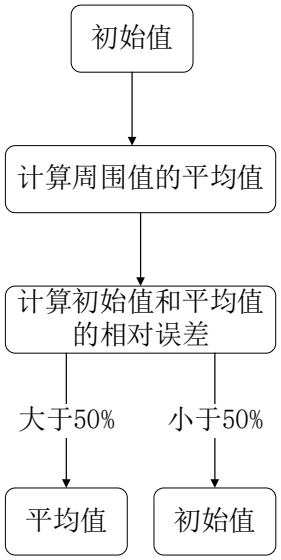
$$C=AB=(c_{ik})_{ml}=(a_{i1}b_{1j}+a_{i2}b_{2j}+\cdots+a_{in}b_{nj})_{ml}=\left(\sum_{k=1}^{n}a_{ik}b_{kj}\right)_{ml}$$

矩阵乘法要求矩阵A的列数与右矩阵B的行数相等，$m\times n$的矩阵A，与$n\times l$的矩阵B相乘，结果为$m\times l$的矩阵C。

2.3 矩阵预处理

根据题目要求，矩阵中任意一个值，与它周边一圈的值的均值，差异不能过大。否则，则用均值代替这个值。

首先，我们选取矩阵中任意一个值作为初始值，计算与其相邻的 8 个值的均值，如果该平均值与初始值之间的差距超过 50%，则输出平均值，否则输出初始值。



现然后我们来分析哪些操作是相互独立的（从而可以进行分布式计算）。很显然，c_{11}和c_{12}的计算是互不干扰的。事实上，C中各个元素的计算都是相互独立的。因此，在 Map 阶段，可以把计算c_{11}所需要的元素都集中到一个 key 中，

然后，在 Reduce 阶段就可以从中解析出各个元素来计算c_{11}，C 的其它元素同理。

同时，a_{11} 会被$c_{11},c_{12},...,c_{1l}$的就算所使用，b_{11} 会被$c_{11},c_{21},...,c_{m1}$的计算所使用，也就是说，在 Map 阶段，当我们从 HDFS 取出一行记录时，如果该记录是 A 的元素，则需要存储成l 个<key, value>对，并且这l 个 key 互不相同；如果该记录是B 的元素，则需要存储成m 个<key, value>对，同样的，m 个 key 也不相同；但同时，用于计算c_{11} 的、存放$a_{11},a_{12},...,a_{1l}$和$b_{11},b_{21},...,b_{m1}$的<key, value>对的 key 都应该相同，这样才能被传递到一个 Reduce 中。

2.4 MapReduce 计算矩阵相乘

1、在 Map 阶段

把来自表 A 的元素 a_{ij}，标识成 l 条<key, value>的形式，其中 $key=(i,k),k=1,2,...,l;\ value=('a',j,a_{ij})$；

把来自表 B 的元素 b_{jk}，标识成 m 条<key, value>的形式，其中 $key=(i,k),k=1,2,...,m;\ value=('b',j,b_{jk})$。

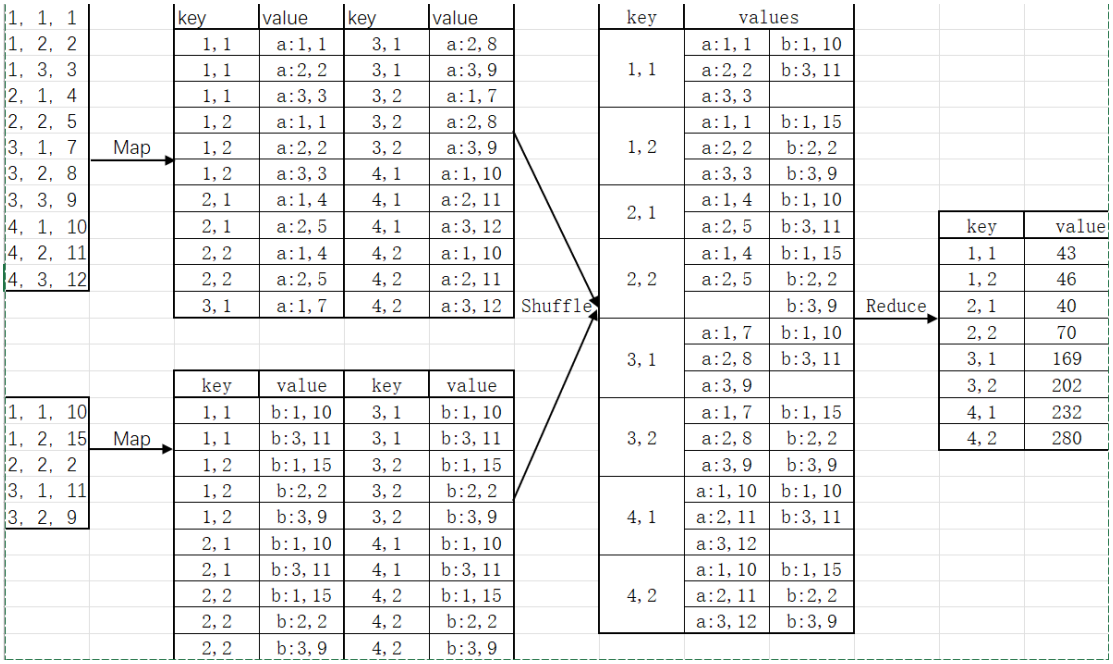
2、在 Shuffle 阶段

相同 key 的 value 会被加入到同一个列表中，形成<key, list(value)>对，传递给 Reduce，这个由 Hadoop 自动完成。

3、在 Reduce 阶段

把 list(value)解析出来，来自 A 的元素，单独放在一个数组中，来自 B 的元素，放在另一个数组中，然后，计算两个数组的点积，即可计算出c_{ik} 的值。

示例如下：



三、程序实现

3.1 数据结构

```
public class MatSize {
public static int m = 10000;
public static int l = 10000;
public static int n = 10000;
}
```

3.2 矩阵预处理

```
public static class PrepareMapper extends Mapper<LongWritable, Text, Text, Text> {
// 行索引
private int rowIndex = 1;
// 8 个方向
final int[][] dirStep = {{-1, -1}, {1, 1}, {-1, 1}, {1, -1}, {1, 0}, {-1, 0}, {0, 1}, {0, -1}};

protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
// 获取一行数据
String[] tokens = value.toString().split(",");
// 枚举元素
for (int j = 0; j < tokens.length; j++) {
int dirRow = -1;
int dirColumn = -1;
// 处理斜方向的位置
for (int v = 0; v < 4; v++) {
dirRow = rowIndex + dirStep[v][0];
dirColumn = j + 1 + dirStep[v][1];
// 在矩阵范围内就添加权值为1 的值
if (1 <= dirRow && dirRow <= MatSize.m && 1 <= dirColumn && dirColumn <= MatSize.n) {
context.write(new Text(dirRow + "," + dirColumn), new Text("1," + tokens[j]));
}
}
// 处理其余位置
for (int v = 4; v < 8; v++) {
dirRow = rowIndex + dirStep[v][0];
dirColumn = j + 1 + dirStep[v][1];
```

```
27.      // 在矩阵范围内就添加权值为2的值
28.      if (1 <= dirRow && dirRow <= MatSize.m && 1 <= dirColumn &&
   dirColumn <= MatSize.n) {
29.       context.write(new Text(dirRow + "," + dirColumn), new Text(
   "2," + tokens[j]));
30.      }
31.     }
32.     context.write(new Text(rowIndex + "," + (j + 1)), new Text("-
   1," + tokens[j]));
33.    }
34.    ++rowIndex;
35.   }
36. }
37.
38. /**
39.  * 执行 Reduce 任务的类
40.  * 继承 Reducer
41.  */
42. public static class PrepareReducer extends Reducer<Text, Text, T
   ext, LongWritable> {
43.
44.  protected void reduce(Text key, Iterable<Text> values, Context
   context) throws IOException, InterruptedException {
45.   long tot = 0;
46.   long weight = 0;
47.   long initialValue = 0;
48.   for (Text value : values) {
49.    String[] val = value.toString().split(",");
50.    // -1 表示初始值
51.    // 1、2 表示权值
52.    if ("-1".equals(val[0])) {
53.     initialValue = Long.parseLong(val[1]);
54.    } else {
55.     weight += Long.parseLong(val[0]);
56.     tot += Long.parseLong(val[1]) * Long.parseLong(val[0]);
57.    }
58.   }
59.   double worthValue = (double) tot / (double) weight;
60.   // 误差估计
61.   if (Math.abs(worthValue - initialValue) / (double) Math.abs(in
   itialValue) >= 0.5) {
62.    context.write(new Text("(" + key.toString() + ")"), new LongW
   ritable(Math.round(worthValue)));
```

```
63.    } else {
64.     context.write(new Text("(" + key.toString() + ")"), new LongW
    ritable(initialValue));
65.    }
66.   }
67.   }
```

3. 3 计算矩阵相乘

```
1.   public static class MatrixMapper extends Mapper<LongWrita
ble, Text, Text, Text> {
2.   private String matrixName = null;
3.   private int rowIndexA = 1;
4.   private int rowIndexB = 1;
5.   protected void setup(Context context) {
6.    matrixName = ((FileSplit) context.getInputSplit()).getPath().g
    etName();
7.   }
8.
9.   protected void map(LongWritable key, Text value, Context contex
    t) throws IOException, InterruptedException {
10.   // 获取一行元素
11.   String[] tokens = value.toString().split(",");
12.   if ("MatrixA".equals(matrixName)) {
13.    // 遍历每个元素
14.    for (int j = 0; j < tokens.length; j++) {
15.     // 忽略0元素
16.     if ("0".equals(tokens[j])) {
17.      continue;
18.     }
19.     // 枚举C的列
20.     // A[row][j]都可用于计算C[row][i]
21.     for (int i = 1; i <= MatSize.n; i++) {
22.      context.write(new Text(rowIndexA + "," + i), new Text("A,"
    + (j + 1) + "," + tokens[j]));
23.     }
24.    }
25.    ++rowIndexA;
26.   } else if ("MatrixB".equals(matrixName)) {
27.    // 遍历每个元素
28.    for (int j = 0; j < tokens.length; j++) {
29.     // 忽略0元素
30.     if ("0".equals(tokens[j])) {
```

```
31.       continue;
32.      }
33.      // 枚举C的行
34.      // B[row][j]都可用于计算C[i][j]
35.      for (int i = 1; i <= MatSize.m; i++) {
36.       context.write(new Text(i + "," + (j + 1)), new Text("B," +
   rowIndexB + "," + tokens[j]));
37.      }
38.     }
39.     ++rowIndexB;
40.    }
41.   }
42. }
43.
44. /**
45.  * 执行Reduce任务的类
46.  * 继承Reducer
47.  */
48. public static class MatrixReducer extends Reducer<Text, Text, Te
   xt, LongWritable> {
49.   protected void reduce(Text key, Iterable<Text> values, Context
   context) throws IOException, InterruptedException {
50.    /*
51.     C[i][j]=sum(A[i][k]*B[k][j])
52.     mapA[k]=A[i][k]
53.     mapB[k]=B[k][j]
54.    */
55.    HashMap<String, String> mapA = new HashMap<>();
56.    HashMap<String, String> mapB = new HashMap<>();
57.    for (Text value : values) {
58.     String[] val = value.toString().split(",");
59.     if ("A".equals(val[0])) {
60.      mapA.put(val[1], val[2]);
61.     } else if ("B".equals(val[0])) {
62.      mapB.put(val[1], val[2]);
63.     }
64.    }
65.    long res = 0;
66.    // 相应元素相乘再累加
67.    for (String mkey : mapA.keySet()) {
68.     if (mapB.get(mkey) == null) {
69.      continue;
70.     }
```

```
71.     res += Long.parseLong(mapA.get(mkey)) * Long.parseLong(mapB.g
   et(mkey));
72.    }
73.    key.set("(" + key + ")");
74.    context.write(key, new LongWritable(res));
75.   }
76. }
    77.  }
```

四、运行结果

4.1 小矩阵

矩阵 A：

```
2 0 2 1 1 0 0 3 6 8 6
9 3 3 6 6 6 5 7 3 1 8
4 6 5 2 4 4 9 1 1 7 5
4 5 2 6 5 6 9 9 5 5 4
1 2 6 1 2 1 9 6 3 6 8
3 4 5 6 5 8 1 5 9 7 3
8 6 3 1 5 3 5 7 2 2 8
4 1 6 3 4 6 9 3 8 7 2
1 8 5 5 8 6 7 7 1 6 4
6 7 3 2 7 8 1 4 6 1 3
5 8 9 3 3 8 7 9 2 2 5
```

矩阵 B:

```
2 0 2 1 1 0 0 3 6 8 6
3 8 8 9 3 4 4 4 6 1 2
1 3 9 1 2 5 2 6 5 8 7
3 9 6 3 7 1 3 5 3 3 5
8 3 9 8 7 4 2 7 2 8 4
2 1 9 4 4 6 1 7 4 1 8
8 9 1 4 1 2 8 3 4 1 9
1 8 6 7 6 1 3 1 2 9 7
5 8 5 5 3 9 7 6 4 1 2
5 4 3 1 4 3 3 7 1 3 8
9 5 5 5 7 6 4 8 9 9 8
```

运行结果：

```
"C:\Program Files\Java\jdk-20\bin\java.exe"
212 217 245 230 180 178 176 186 221 233 231
203 209 232 214 171 167 168 181 214 225 224
221 228 254 238 186 182 181 195 230 237 236
212 213 239 220 177 172 173 183 215 226 226
223 222 248 232 185 182 183 189 222 235 239
214 223 248 236 182 180 178 189 226 233 231
223 224 251 236 185 181 181 189 220 229 231
234 243 269 255 197 191 191 206 242 244 245
230 241 271 255 195 191 186 205 244 246 243
232 250 283 270 202 199 195 212 257 251 245
249 267 303 283 216 211 208 228 277 269 262
Process finished with exit code 0
```