

实验报告——MapReduce实现矩阵乘法

薛飞跃 1700017831

朱政烨 1700017760

2020 年 4 月 12 日

1 实验步骤

1.1 继承mapper类，重写setup方法如下：

```
public static class MatrixMapper extends
    Mapper<LongWritable, Text, Text, Text> {
    private String flag = null; // 数据集名称
    private int rowNum = 10; // 矩阵A的行数
    private int colNum = 10; // 矩阵B的列数
    private int rowIndexA = 1; // 矩阵A, 当前在第几行
    private int rowIndexB = 1; // 矩阵B, 当前在第几行

    @Override
    protected void setup(Context context) throws IOException,
        InterruptedException {
        flag = ((FileSplit) context.getInputSplit()).getPath().getName(); // 获取文件名称
    }

    @Override
```

- 1.2 重写map方法。首先使用flag判断调用map方法的矩阵。遍历获取当前矩阵的当前行所有数据。在context中，key=(i,j)代表结果矩阵的第i行和第j列，value值分为三部分(p,q,r)，代表p矩阵的第i行第q列的数值为r。

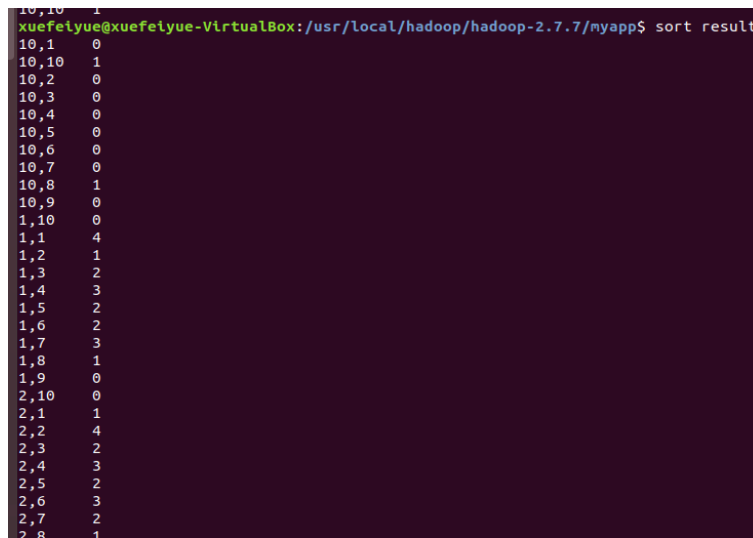
```
protected void map(LongWritable key, Text value, Context context)
    throws IOException, InterruptedException {
    String[] tokens = value.toString().split(" ");
    if ("ma".equals(flag)) {
        for (int i = 1; i <= colNum; i++) {
            Text k = new Text(rowIndexA + "," + i);
            for (int j = 0; j < tokens.length; j++) {
                Text v = new Text("a," + (j + 1) + "," + tokens[j]);
                context.write(k, v);
            }
        }
        rowIndexA++; // 每执行一次map方法，矩阵向下移动一行
    } else if ("mb".equals(flag)) {
        for (int i = 1; i <= rowNum; i++) {
            for (int j = 0; j < tokens.length; j++) {
                Text k = new Text(i + "," + (j + 1));
                Text v = new Text("b," + rowIndexB + "," + tokens[j]);
                context.write(k, v);
            }
        }
        rowIndexB++; // 每执行一次map方法，矩阵向下移动一行
    }
}
```

- 1.3 重写reduce方法。先解析来自map的结果，根据value(p,q,r)的格式，根据不同的p分别建立q到r的map。mapA和mapB的key值相同的value相乘后累加，即得结果矩阵第i行第j列的值。

```
int result = 0;
Iterator<String> mKeys = mapA.keySet().iterator();
while (mKeys.hasNext()) {
    String mkey = mKeys.next();
    if (mapB.get(mkey) == null) { // 因为mkey取的是mapA的key集合，所以只需要判断mapB是否存在即可。
        continue;
    }
    result += Integer.parseInt(mapA.get(mkey))
        * Integer.parseInt(mapB.get(mkey));
}
context.write(key, new IntWritable(result));
```

2 实验结果

输出结果以key: value的格式保存，key代表结果矩阵的行号和列号，value代表数值。



```
xuefelyue@xuefelyue-VirtualBox: /usr/local/hadoop/hadoop-2.7.7/myapp$ sort result
10,10 1
10,1 0
10,10 1
10,2 0
10,3 0
10,4 0
10,5 0
10,6 0
10,7 0
10,8 1
10,9 0
1,10 0
1,1 4
1,2 1
1,2 2
1,3 2
1,4 3
1,5 2
1,6 2
1,7 3
1,8 1
1,9 0
2,10 0
2,1 1
2,2 4
2,3 2
2,4 3
2,5 2
2,6 3
2,7 2
2,8 1
```

和matlab计算的结果进行比对，计算结果无误。

```

ans =
  4  1  2  3  2  2  3  1  0  0
  1  4  2  3  2  3  2  1  0  0
  2  2  3  2  1  2  2  1  0  0
  3  3  2  6  2  3  3  2  0  0
  2  2  1  2  3  2  2  1  0  0
  2  3  2  3  2  5  2  1  1  0
  3  2  2  3  2  2  5  1  1  0
  1  1  1  2  1  1  1  3  0  1
  0  0  0  0  0  1  1  0  2  0
  0  0  0  0  0  0  0  1  0  1
  ``

```

3 实验心得

1. 确定好以什么作为key值很关键。reduce阶段需要来自A矩阵的一行和B矩阵的一列，Hadoop会自动把map阶段key值相同的value分到一个iterable中，那么key值应该是(i, j)代表矩阵的第i行第j列，map阶段的value则必须存储来自哪个矩阵这一信息。
2. 最终的结果是以(行列号: value)的形式存储，只需稍加改动(去掉为0的键值对)即可方便的存储为稀疏矩阵的形式，使用Hadoop进行矩阵乘法，实际中应该会更多的遇到非常大的稀疏矩阵，相比原始的矩阵存储方式作为输入，以稀疏矩阵的形式输入有其优势：①稀疏矩阵有大量的0，传统矩阵的形式可能占更大的空间②方便分批读入分批处理，一般的矩阵形式必须一次读入，只读取一部分时无法知道相应的行号和列号，而稀疏矩阵的存储方式无此问题。