暨南大学本科实验报告专用纸

课程名称 云计算实验 成绩评定

实验项目名称 基于 MapReduce 框架实现电影个性化推荐

指导教师 梁倬骞、魏林锋

实验项目编号 0806030812 实验项目类型 验证型 实验地点 N116

学生姓名 陈宇 学号 2020101642

学院 信息科学技术 系 计算机 专业 计算机科学与技术

实验时间 2022 年 12 月 14 日上午～12 月 14 日中午 温度 ℃湿度

12.1 实验目的

1）理解分布式离线计算框架 MapRecue 的工作原理。

2）通过实验掌握分布式离线计算框架 MapRecue 的编程。

3）使用 MapReuce 实现电影的个性化推荐。

12.2 实验内容

使用 MapReuce 实现电影的个性化推荐。

12.3 案例引入

- 互联网某电影点评网站，主要产品包括：
  - 电影介绍
  - 电影排行
  - **网友对电影打分**
  - 网友影评
  - 影讯&购票
  - 用户在看|想看|看过的电影
  - **猜你喜欢（推荐）**
- 利用用户对电影的打分表来给用户推荐电影，用户打分表包括以下字段：
  - userID--用户 ID 号

- itemID--电影 ID 号
- score--评分

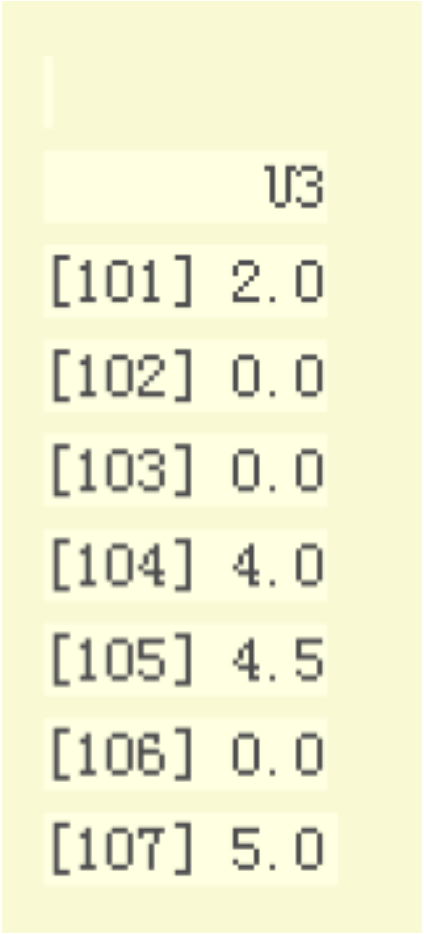
12.4 实验原理

12.4.1 基于物品的协同过滤算法

- 1、建立物品的同现矩阵

	[101]	[102]	[103]	[104]	[105]	[106]	[107]
[101]	5	3	4	4	2	2	1
[102]	3	3	3	2	1	1	0
[103]	4	3	4	3	1	2	0
[104]	4	2	3	4	2	2	1
[105]	2	1	1	2	2	1	1
[106]	2	1	2	2	1	2	0
[107]	1	0	0	1	1	0	1

- 2、建立用户对物品的评分矩阵



- 3、矩阵计算推荐结果

	101	102	103	104	105	106	107		U3		R
101	5	3	4	4	2	2	1		2.0		40.0
102	3	3	3	2	1	1	0		0.0		18.5
103	**4**	**3**	**4**	**3**	**1**	**2**	**0**	**X**	**0.0**	**=**	**24.5**
104	4	2	3	4	2	2	1		4.0		40.0
105	2	1	1	2	2	1	1		4.5		26.0
106	2	1	2	2	1	2	0		0.0		16.5
107	1	0	0	1	1	0	1		5.0		15.5

12.4.2 MapReduce 实现

- Java 类说明：
  - Recommend.java--主任务启动程序

- Step1.java--按用户分组，计算所有物品出现的组合列表，得到用户对物品的评分矩阵
- Step2.java--对 itemID 组合列表进行计数，建立其同现矩阵
- Step3.java--对同现矩阵和评分矩阵进行转型，便于后续处理
- Step4_Update.java--矩阵相乘乘法部分
- Step4_Update2.java--矩阵相乘加法部分
- Step5.java--对结果进行过滤和排序
- HDFSFile.java--HDFS 路径文件操作类
- SortHashMap.java--HashMap 排序类

12.5 大数据平台环境

已经配置完成的 Hadoop 伪分布式或完全分布式环境。环境配置如下：

Hadoop01：192.168.24.91 Hadoop02：192.168.24.92 Hadoop03：192.168.24.93 管理员用户：root / admin@1 Hadoop 用户：hadoop / hadoop

12.6 大数据编程环境配置

该部分以前已经做过，如果已经忘记怎么做可以参考本文档。

1、安装 Java 环境，下载在 windows 系统中下载 jdk1.8.0_131，图形化界面安装。

2、配置环境变量，右键点击“此电脑”，选择“属性”，点击“高级系统设置”，点击“环境变量”，在“系统变量”中按以下要求新增或修改环境变量。环境变量配置如下：

变量名	变量值	备注
JAVA_HOME	C:\Program Files\Java\jdk1.8.0_131	新增，填写的值为 windows 系统下 Java 环境的目录
HADOOP_HOME	D:\hadoop	新增，填写的值为 windows 系统下 hadoop 环境的目录
HADOOP_USER_HOME	jiahui	新增，填写的值为当前登录的 windows 用户名（不能使用中

		文)
CLASSPATH	.;%JAVA_HOME%\lib;%JAVA_HOME%\lib\dt.jar;%JAVA_HOME%\lib\tools.jar	新增
PATH	%JAVA_HOME%\bin; %JAVA_HOME%\jre\bin;	在原有 PATH 中添加此两项（切记不是覆盖)。

3、使用 XFTP，登录到 192.168.24.91，下载/usr/hadoop 目录到 D 盘中。

4、将 hadoop-eclipse-plugin-2.7.0.jar 拷贝到“D:\Program Files (x86)\sts-bundle\sts-3.9.1.RELEASE\plugins”目录中。

5、将 winutils.exe 拷贝到 D:\hadoop\bin 目录中。

6、将 hadoop.dll 拷贝到 C:\Windows\System32 目录中。

7、打开 STS.exe，点击 Window->Preferences->Hadoop Map/Reduce,设置 Hadoop installation directory 为：D:\hadoop，点击 Apply and Close。

8、创建项目，选择 File->New->Project->Map/Reduce Project,点击“NEXT”，输入项目名称：MapReduce，点击“NEXT”，点击“Finish”。

9、在下方窗口找到 Map/Reduce Locations，右键点击“New Hadoop Location”。设置如下：Location name：192.168.24.91；Host：192.168.24.91；左边 Port：9001；右边 Port：9000；勾选“Use M/R Master host”；User name：jiahui，点击 Finish。配置完成后，可间左方窗口的 DFS Locations 下有 192.168.24.91 的信息。

10、右键点击左边窗口的 192.168.24.91,选择 Refresh,即可将 HDFS 的目录刷新出来。

11、使用 xshell 连接到 192.168.24.91，使用管理员用户登录。

12、创建用户 jiahui，命令如下：

```
[root@master ~]# useradd jiahui
```

13、切换到 hadoop 用户，命令如下：

```
[root@master ~]# su hadoop
```

14、打开 hadoop 集群，命令如下：

```
[hadoop@master ~]$ start-all.sh
```

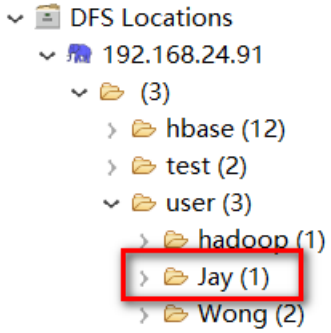
15、在 hdfs 上创建 jiahui 文件夹，并将该文件夹的权限赋予给 jiahui 用户，命令如下：

```
[hadoop@master ~]$ hdfs dfs -mkdir /user/jiahui
```

```
[hadoop@master ~]$ hdfs dfs -chown -R jiahui:jiahui /user/jiahui
```

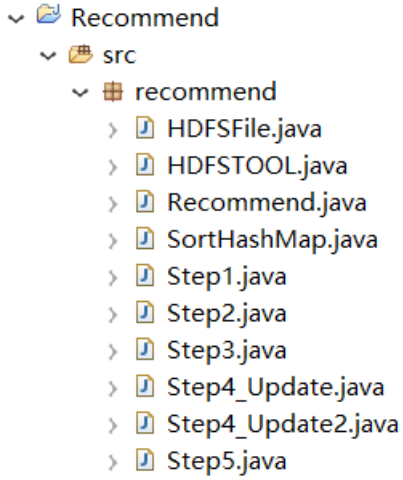
```
[hadoop@master ~]$ hdfs dfs -chown -R Jay:Jay /user/Jay
[hadoop@master ~]$ hdfs dfs -mkdir /user/Jay
mkdir: `/user/Jay': File exists
```

16、右键点击左边窗口的 192.168.24.91,选择 Refresh，即可将 HDFS 的/user/jiahui 目录刷新出来。



12.7 实验步骤与结果

1、新建一个名为 Recommend 的 MapReuduce Project。



2、编写 HDFSTOOL.java 文件，配置 Hadoop 的主机 IP 和访问端口号。代码如下：

```
package recommend;

public class HDFSTOOL {
    public static String MASTERNAME = "192.168.24.91";
    public static String DFSPORT = "9000";
}
```

3、编写 HDFSFile.java 文件，实现 HDFS 的文件操作。代码如下：

```
package recommend;
```

```java
import java.io.IOException;
import java.text.SimpleDateFormat;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FSDataInputStream;
import org.apache.hadoop.fs.FSDataOutputStream;
import org.apache.hadoop.fs.FileStatus;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IOUtils;

/**
 * HDFS文件操作
 *
 */
public class HDFSFile {
    Configuration conf = new Configuration();
    private FileSystem hdfs;

    /**
     * 构造方法
     *
     * @param hdfsPath
     * @throws IOException
     */
    public HDFSFile(Path hdfsPath) throws IOException {
        hdfs = hdfsPath.getFileSystem(conf);
    }

    /**
     * 创建目录
     *
     * @param path
     * @throws IOException
     */
    public void mkDir(Path path) throws IOException {
        hdfs.mkdirs(path);
    }

    /**
     * 上传文件
     *
```

```
     * @param src
     * @param dst
     * @throws IOException
     */
    public void copyLocalToHdfs(Path src, Path dst) throws IOException {
        hdfs.copyFromLocalFile(src, dst);
    }

    /**
     * 删除文件
     *
     * @param path
     * @throws IOException
     */
    @SuppressWarnings("deprecation")
    public void delFile(Path path) throws IOException {
        hdfs.delete(path);
    }

    /**
     * 读取文件内容
     *
     * @param path
     * @throws IOException
     */
    public void readFile(Path path) throws IOException {
        // 获取文件信息
        FileStatus filestatus = hdfs.getFileStatus(path);
        // FS的输入流
        FSDataInputStream in = hdfs.open(path);
        // 用Hadoop的IOUtils工具方法来让这个文件的指定字节复制到标准输出流上
        IOUtils.copyBytes(in, System.out, (int) filestatus.getLen(),
false);
        System.out.println();
    }

    /**
     * 得到文件的修改时间
     *
     * @param path
     * @throws IOException
     */
    public void getModifyTime(Path path) throws IOException {
```

```
        FileStatus files[] = hdfs.listStatus(path);
        for (FileStatus file : files) {
            // time = file.getModificationTime().
            SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd
hh:mm:ss");
            String date = sdf.format(file.getModificationTime());
            System.out.println(file.getPath() + "\t" + date);
        }
    }

    /**
     * 在hdfs上创建文件并写入内容
     *
     * @param path
     * @param content
     * @throws IOException
     */
    public void writeFile(Path path, String content) throws IOException {
        FSDataOutputStream os = hdfs.create(path);
        // 以utf-8的格式写入文件
        os.write(content.getBytes("UTF-8"));
        os.close();
    }

    /**
     * 列出某一路径下所有的文件
     *
     * @param path
     * @throws IOException
     */
    @SuppressWarnings("deprecation")
    public void listFiles(Path path) throws IOException {
        hdfs = path.getFileSystem(conf);
        FileStatus files[] = hdfs.listStatus(path);
        int listlength = files.length;
        for (int i = 0; i < listlength; i++) {
            if (files[i].isDir() == false) {
                System.out.println("filename:" + files[i].getPath() +
"\tsize:" + files[i].getLen());
            } else {
                Path newpath = new Path(files[i].getPath().toString());
                listFiles(newpath);
            }
```

```
        }
    }
}
```

4、编写 Step1.java 文件，获取用户评分向量。代码如下：

```
package recommend;
import java.io.IOException;
import java.util.Map;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.Reducer;




/**
 * 得到用户评分向量
 *
 */
public class Step1{

    public static class Step1_ToItemPreMapper extends Mapper<Object, Text, IntWritable,
Text> {
        private final static IntWritable k = new IntWritable();
        private final static Text v = new Text();

        @Override
        public void map(Object key, Text value, Context context) throws IOException,
InterruptedException {
            String[] tokens = Recommend.DELIMITER.split(value.toString());
            int userID = Integer.parseInt(tokens[0]);
            String itemID = tokens[1];
            String pref = tokens[2];
            k.set(userID);
            v.set(itemID + ":" + pref);
            context.write(k, v);
```

```
        }
    }

    public static class Step1_ToUserVectorReducer extends Reducer <IntWritable, Text, IntWritable, Text> {
        private final static Text v = new Text();

        @Override
        protected void reduce(IntWritable key, Iterable<Text> values,
                Reducer<IntWritable, Text, IntWritable, Text>.Context context)
                throws IOException, InterruptedException {
            // TODO Auto-generated method stub
            StringBuilder sb = new StringBuilder();
            for (Text value:values) {
                sb.append("," + value.toString());
            }
            v.set(sb.toString().replaceFirst(",", ""));
            context.write(key, v);
        }

    }

    public static void run(Map<String, String> path) throws IOException, ClassNotFoundException, InterruptedException {
        //获得配置信息
        Configuration conf = Recommend.config();
        //得到输入输出路径
        Path input = new Path(path.get("Step1Input"));
        Path output = new Path(path.get("Step1Output"));
        //将本地文件上传到集群
        HDFSFile hdfs = new HDFSFile(new Path(Recommend.HDFS));
        hdfs.delFile(input);
        hdfs.mkDir(input);
        hdfs.copyLocalToHdfs(new Path(path.get("data")), input);
        //设置作业参数
        @SuppressWarnings("deprecation")
        Job job = new Job(conf,"Step1");
        job.setJarByClass(Step1.class);

        job.setMapperClass(Step1_ToItemPreMapper.class);
        job.setCombinerClass(Step1_ToUserVectorReducer.class);
        job.setReducerClass(Step1_ToUserVectorReducer.class);
```

```
        job.setOutputKeyClass(IntWritable.class);
        job.setOutputValueClass(Text.class);

        FileInputFormat.addInputPath(job,input);
        FileOutputFormat.setOutputPath(job,output);
        //运行作业
        job.waitForCompletion(true);
    }

}
```

5、编写 Step2.java 文件，由用户评分向量得到共现矩阵。代码如下：

```
package recommend;

import java.io.IOException;
import java.util.Map;

import org.apache.hadoop.conf.Configuration;
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.Job;
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.output.FileOutputFormat;

/**
 * 由用户评分向量得到共现矩阵
 *
 */
public class Step2 {

    public static class Step2_UserVectorToCooccurrenceMapper extends
Mapper<LongWritable, Text, Text, IntWritable> {
        private final static Text k = new Text();
        private final static IntWritable v = new IntWritable(1);

        @Override
        public void map(LongWritable key, Text values, Context context) throws
IOException, InterruptedException {
            String[] tokens = Recommend.DELIMITER.split(values.toString());
```

```
                for (int i = 1; i < tokens.length; i++) {
                    String itemID = tokens[i].split(":")[0];
                    for (int j = 1; j < tokens.length; j++) {
                        String itemID2 = tokens[j].split(":")[0];
                        k.set(itemID + ":" + itemID2);
                        context.write(k, v);
                    }
                }
            }
        }

        public static class Step2_UserVectorToConoccurrenceReducer extends Reducer <Text,
IntWritable, Text, IntWritable> {
            private IntWritable result = new IntWritable();

            @Override
            protected void reduce(Text key, Iterable<IntWritable> values,Context context)
                    throws IOException, InterruptedException {
                int sum = 0;
                for (IntWritable value:values) {
                    sum += value.get();
                }
                result.set(sum);
                context.write(key, result);
            }

        }
        @SuppressWarnings("deprecation")
        public static void run(Map<String, String> path) throws IOException,
ClassNotFoundException, InterruptedException {
            //获得配置信息
            Configuration conf = Recommend.config();
            //得到输入输出路径
            Path input = new Path(path.get("Step2Input"));
            Path output = new Path(path.get("Step2Output"));
            //删掉上次输出结果
            HDFSFile hdfs = new HDFSFile(new Path(Recommend.HDFS));
            hdfs.delFile(output);
            //设置作业参数
            Job job = new Job(conf,"Step2");
            job.setJarByClass(Step2.class);

            job.setMapperClass(Step2_UserVectorToCooccurrenceMapper.class);
```

```
        job.setCombinerClass(Step2_UserVectorToConoccurrenceReducer.class);
        job.setReducerClass(Step2_UserVectorToConoccurrenceReducer.class);

        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        FileInputFormat.addInputPath(job,input);
        FileOutputFormat.setOutputPath(job,output);
        //运行作业
        job.waitForCompletion(true);
    }
}
```

6、编写 Step3.java 文件，对评分向量和共现矩阵进行整理。代码如下：

```
package recommend;

import java.io.IOException;
import java.util.Map;

import org.apache.hadoop.conf.Configuration;
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.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

/**
 * 对评分向量和共现矩阵进行整理
 *
 */
public class Step3 {
    public static class Step31_UserVectorSplitterMapper extends Mapper<LongWritable,
Text, IntWritable, Text> {
        private final static IntWritable k = new IntWritable();
        private final static Text v = new Text();

        @Override
        public void map(LongWritable key, Text values, Context context)
                throws IOException, InterruptedException {
            String[] tokens = Recommend.DELIMITER.split(values.toString());
```

```
                for (int i = 1; i < tokens.length; i++) {
                    String[] vector = tokens[i].split(":");
                    int itemID = Integer.parseInt(vector[0]);
                    String pref = vector[1];

                    k.set(itemID);
                    v.set(tokens[0] + ":" + pref);
                    context.write(k, v);
                }
            }
        }
        @SuppressWarnings("deprecation")
        public static void run1(Map<String, String> path) throws IOException, ClassNotFoundException, InterruptedException {
            //获得配置信息
            Configuration conf = Recommend.config();
            //得到输入输出路径
            Path input = new Path(path.get("Step3Input1"));
            Path output = new Path(path.get("Step3Output1"));
            //删除上一次的输出
            HDFSFile hdfs = new HDFSFile(new Path(Recommend.HDFS));
            hdfs.delFile(output);
            //设置作业参数
            Job job = new Job(conf,"Step3_1");
            job.setJarByClass(Step3.class);

            job.setMapperClass(Step31_UserVectorSplitterMapper.class);

            job.setOutputKeyClass(IntWritable.class);
            job.setOutputValueClass(Text.class);

            FileInputFormat.addInputPath(job,input);
            FileOutputFormat.setOutputPath(job,output);
            //运行作业
            job.waitForCompletion(true);
        }

        public static class Step32_CooccurrenceColumnWrapperMapper extends Mapper<LongWritable, Text, Text, IntWritable> {
            private final static Text k = new Text();
            private final static IntWritable v = new IntWritable();

            @Override
```

```
            public void map(LongWritable key, Text values,Context context)
                    throws IOException, InterruptedException {
                String[] tokens = Recommend.DELIMITER.split(values.toString());
                k.set(tokens[0]);
                v.set(Integer.parseInt(tokens[1]));
                context.write(k, v);
            }
        }

        @SuppressWarnings("deprecation")
        public static void run2(Map<String, String> path) throws IOException,
            ClassNotFoundException, InterruptedException {
            // 获得配置信息
            Configuration conf = Recommend.config();
            // 得到输入输出路径
            Path input = new Path(path.get("Step3Input2"));
            Path output = new Path(path.get("Step3Output2"));
            // 删除上一次的输出
            HDFSFile hdfs = new HDFSFile(new Path(Recommend.HDFS));
            hdfs.delFile(output);
            // 设置作业参数
            Job job = new Job(conf, "Step3_2");
            job.setJarByClass(Step3.class);

            job.setMapperClass(Step32_CooccurrenceColumnWrapperMapper.class);

            job.setOutputKeyClass(Text.class);
            job.setOutputValueClass(IntWritable.class);

            FileInputFormat.addInputPath(job, input);
            FileOutputFormat.setOutputPath(job, output);
            // 运行作业
            job.waitForCompletion(true);
        }
}
```

7、编写 Step4_Update.java 文件，实现共现矩阵乘评分向量的乘法部分。代码如下：

```
package recommend;

import java.io.IOException;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
```

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
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.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.FileSplit;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

/**
 * 共现矩阵乘评分向量的乘法部分
 *
 */
public class Step4_Update {
	public static class Step4_PartialMultiplyMapper extends Mapper<LongWritable, Text,
Text, Text> {

		private String flag;// A 同现矩阵/左矩阵  or B 评分矩阵/右矩阵

		@Override
		protected void setup(Context context) throws IOException, InterruptedException
{
			FileSplit split = (FileSplit) context.getInputSplit();
			flag = split.getPath().getParent().getName();//  判断读的数据集
		}

		@Override
		public void map(LongWritable key, Text values, Context context) throws
IOException, InterruptedException {
			String[] tokens = Recommend.DELIMITER.split(values.toString());

			if (flag.equals("step3_2")) {//  同现矩阵
				String[] v1 = tokens[0].split(":");
				String itemID1 = v1[0];
				String itemID2 = v1[1];
				String num = tokens[1];

				Text k = new Text(itemID1);
```

```
                Text v = new Text("A:" + itemID2 + "," + num);

                context.write(k, v);
                // System.out.println(k.toString() + "    " + v.toString());

            } else if (flag.equals("step3_1")) {//  评分矩阵
                String[] v2 = tokens[1].split(":");
                String itemID = tokens[0];
                String userID = v2[0];
                String pref = v2[1];

                Text k = new Text(itemID);
                Text v = new Text("B:" + userID + "," + pref);

                context.write(k, v);
                // System.out.println(k.toString() + "    " + v.toString());
            }
        }

    }

    public static class Step4_AggregateReducer extends Reducer<Text, Text, Text, Text> {

        @Override
        public void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException {
            //System.out.println(key.toString() + ":");

            Map<String, String> mapA = new HashMap<String, String>();
            Map<String, String> mapB = new HashMap<String, String>();

            for (Text line : values) {
                String val = line.toString();
                //System.out.println(val);

                if (val.startsWith("A:")) {
                    String[] kv = Recommend.DELIMITER.split(val.substring(2));
                    mapA.put(kv[0], kv[1]);

                } else if (val.startsWith("B:")) {
                    String[] kv = Recommend.DELIMITER.split(val.substring(2));
                    mapB.put(kv[0], kv[1]);
```

```
                }
            }

            double result = 0;
            Iterator<String> iter = mapA.keySet().iterator();
            while (iter.hasNext()) {
                String mapk = iter.next();// itemID

                int num = Integer.parseInt(mapA.get(mapk));
                Iterator<String> iterb = mapB.keySet().iterator();
                while (iterb.hasNext()) {
                    String mapkb = iterb.next();// userID
                    double pref = Double.parseDouble(mapB.get(mapkb));
                    result = num * pref;// 矩阵乘法相乘计算

                    Text k = new Text(mapkb.toString());
                    Text v = new Text(mapk + "," + result);

                    context.write(k, v);
                    //System.out.println(k.toString() + "    " + v.toString());
                }
            }
        }
    }

    @SuppressWarnings("deprecation")
    public static void run(Map<String, String> path) throws IOException,
InterruptedException, ClassNotFoundException {
        // 获得配置信息
        Configuration conf = Recommend.config();
        // 得到输入输出路径
        Path input1 = new Path(path.get("Step4_1Input1"));
        Path input2 = new Path(path.get("Step4_1Input2"));
        Path output = new Path(path.get("Step4_1Output"));
        // 删除上一次的输出
        HDFSFile hdfs = new HDFSFile(new Path(Recommend.HDFS));
        hdfs.delFile(output);
        // 设置作业参数
        Job job = new Job(conf);
        job.setJarByClass(Step4_Update.class);

        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);
```

```
        job.setMapperClass(Step4_Update.Step4_PartialMultiplyMapper.class);
        job.setReducerClass(Step4_Update.Step4_AggregateReducer.class);

        job.setInputFormatClass(TextInputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);

        FileInputFormat.setInputPaths(job, input1, input2);
        FileOutputFormat.setOutputPath(job, output);

        job.waitForCompletion(true);
    }
}
```

8、编写 Step4_Update2.java 文件，实现共现矩阵乘评分向量的加法部分。代码如下：

```
package recommend;

import java.io.IOException;
import java.util.Map;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
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.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;



/**
 * 共现矩阵乘评分向量的加法部分
 * 优化了原始代码
 *
 */
public class Step4_Update2 {

    public static class Step4_RecommendMapper extends Mapper<LongWritable, Text,
Text, Text> {
```

```
        @Override
        public void map(LongWritable key, Text values, Context context) throws IOException, InterruptedException {
            //原来的，修改之前
            //String[] tokens = Recommend.DELIMITER.split(values.toString());
            //Text k = new Text(tokens[0]);
            //Text v = new Text(tokens[1]+","+tokens[2]);
            //context.write(k, v);
            //修改后
            String[] tokens = Recommend.DELIMITER.split(values.toString());
            Text k = new Text(tokens[0]+","+tokens[1]);
            Text v = new Text(tokens[2]);
            context.write(k, v);
        }
    }

    public static class Step4_RecommendReducer extends Reducer<Text, Text, Text, Text> {

        @Override
        public void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException {
            //修改前
            /*System.out.println(key.toString() + ":");
            Map<String, Double> map = new HashMap<String, Double>();// 结果

            for (Text line : values) {
                System.out.println(line.toString());
                String[] tokens = Recommend.DELIMITER.split(line.toString());
                String itemID = tokens[0];
                Double score = Double.parseDouble(tokens[1]);

                if (map.containsKey(itemID)) {
                    map.put(itemID, map.get(itemID) + score);// 矩阵乘法求和计算
                } else {
                    map.put(itemID, score);
                }
            }

            Iterator<String> iter = map.keySet().iterator();
            while (iter.hasNext()) {
                String itemID = iter.next();
                double score = map.get(itemID);
```

```
                Text v = new Text(itemID + "," + score);
                context.write(key, v);
            }*/
            //修改后
            System.out.println(key+"---");
            double score=0.0;
            for(Text line : values){
                System.out.println(line);
                score += Double.valueOf(line.toString());
            }
            String[] tokens = Recommend.DELIMITER.split(key.toString());
            String userID = tokens[0];
            String itemID = tokens[1];
            Text k = new Text(userID);
            Text v = new Text(itemID + "," + String.valueOf(score));
            context.write(k, v);
        }
    }

    @SuppressWarnings("deprecation")
    public static void run(Map<String, String> path) throws IOException,
InterruptedException, ClassNotFoundException {
        // 获得配置信息
        Configuration conf = Recommend.config();
        // 得到输入输出路径
        Path input = new Path(path.get("Step4_2Input"));
        Path output = new Path(path.get("Step4_2Output"));
        // 删除上一次的输出
        HDFSFile hdfs = new HDFSFile(new Path(Recommend.HDFS));
        hdfs.delFile(output);
        // 设置作业参数
        Job job = new Job(conf);
        job.setJarByClass(Step4_Update2.class);

        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);

        job.setMapperClass(Step4_Update2.Step4_RecommendMapper.class);
        job.setReducerClass(Step4_Update2.Step4_RecommendReducer.class);

        job.setInputFormatClass(TextInputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);
```

```
            FileInputFormat.setInputPaths(job, input);
            FileOutputFormat.setOutputPath(job, output);

            job.waitForCompletion(true);
        }

}
```

9、编写 SortHashMap.java 文件，实现 HashMap 排序。代码如下：

```
package recommend;
import java.util.Collections;
import java.util.Comparator;
import java.util.HashMap;
import java.util.LinkedList;
import java.util.List;
import java.util.Map.Entry;



/**
 * HashMap 排序
 *
 */
public class SortHashMap {
    private List<Entry<String,Float>> list = new LinkedList<Entry<String,Float>>();
    /**
     * 倒序
     * @param map
     * @return
     */
    public static   List<Entry<String,Float>> sortHashMap(HashMap<String,Float> map){
        SortHashMap sorthashmap = new SortHashMap();

        sorthashmap.list.addAll(map.entrySet());

        Collections.sort(sorthashmap.list,new Comparator<Entry<String,Float>>(){
            public int compare(Entry<String,Float> obj1,Entry<String,Float> obj2){

    if(Float.parseFloat(obj1.getValue().toString())<Float.parseFloat(obj2.getValue().toStrin
g()))
                    return 1;
                else
if(Float.parseFloat(obj1.getValue().toString())==Float.parseFloat(obj2.getValue().toString())
```

```
)
                    return 0;
                else
                    return -1;
            }
        });
        /*Iterator<Entry<String,Float>> ite = list.iterator();
        while(ite.hasNext()){
            Entry<String,Float> tmp = ite.next();
            System.out.println(tmp.getKey()+"\t"+tmp.getValue());
            sorthashmap.map.put(tmp.getKey(),tmp.getValue());
        }*/
        return sorthashmap.list;
    }
    public static void main(String[]args){
        HashMap<String, Float> omap = new HashMap<String, Float> ();
        omap.put("a", (float)(1.0));
        omap.put("b", (float)(3.0));
        omap.put("c", (float)(2.0));
        List<Entry<String,Float>> list = new LinkedList<Entry<String,Float>>();
        list=SortHashMap.sortHashMap(omap);
        for(Entry<String,Float> ilist : list){
            System.out.println(ilist.getKey()+"\t"+ilist.getValue());
        }
    }
}
```

10、编写 Step5.java 文件，实现对电影推荐结果进行过滤和排序。代码如下：

```
package recommend;

import java.io.IOException;
import java.util.HashMap;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;
import java.util.Map.Entry;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
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.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.FileSplit;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

/**
 * 对结果进行过滤和排序
 * 1 过滤掉用户已经打过分的
 * 2 按推荐权重倒序排列
 */
public class Step5 {
    public static class Step5_FilterSortMapper extends Mapper<LongWritable, Text, Text,
Text> {
        private String flag;// 判断输入的文件
        private Text k;
        private Text v;
        @Override
        protected void setup(
                Mapper<LongWritable, Text, Text, Text>.Context context)
                throws IOException, InterruptedException {
            FileSplit split = (FileSplit) context.getInputSplit();
            flag = split.getPath().getParent().getName();// 判断读的数据集
        }

        @Override
        public void map(LongWritable key, Text values, Context context) throws
IOException, InterruptedException {

            if(flag.equals("step4_2")){//values like 1    101,44.0
                String[] tokens = Recommend.DELIMITER.split(values.toString());
                k = new Text(tokens[0]);
                v = new Text("W:" + tokens[1] + "," + tokens[2]);//为推荐权重
            }else{
                String[] tokens = Recommend.DELIMITER.split(values.toString());
                k = new Text(tokens[0]);
                v = new Text("S:" + tokens[1] + "," + tokens[2]);//为分数
            }
            context.write(k,v);
        }
    }
```

```
    public static class Step5_FilterSortReducer extends Reducer<Text, Text, Text, Text> {
        private Text k;
        private Text v;
        @Override
        public void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException {
            HashMap<String,String> wMap = new HashMap<String,String>();
            HashMap<String,String> sMap = new HashMap<String,String>();

            System.out.println(key+"-----");
            for(Text line:values){
                System.out.println(line);
                String[] tokens = Recommend.DELIMITER.split(line.toString());
                String flag = tokens[0].split(":")[0];
                String itemID = tokens[0].split(":")[1];
                if(flag.equals("W")){
                    wMap.put(itemID, tokens[1]);
                }else{
                    sMap.put(itemID, tokens[1]);
                }
            }
            //过滤
            HashMap<String,Float> filterMap = new HashMap<String,Float>();
            Iterator<String> iter = wMap.keySet().iterator();
            while(iter.hasNext()){
                String k = iter.next();
                if(sMap.containsKey(k)==false)
                    filterMap.put(k, Float.valueOf(wMap.get(k)));
            }
            //排序
            List<Entry<String,Float>> list = new LinkedList<Entry<String,Float>>();
            list=SortHashMap.sortHashMap(filterMap);
            for(Entry<String,Float> l : list){
                k = key;
                v = new Text(l.getKey().toString() + "," + l.getValue().toString());
                context.write(k,v);
            }
        }
    }
    @SuppressWarnings("deprecation")
    public static void run(Map<String, String> path) throws IOException, InterruptedException, ClassNotFoundException {
        // 获得配置信息
```

```
        Configuration conf = Recommend.config();
        // 得到输入输出路径
        Path input1 = new Path(path.get("Step5Input1"));
        Path input2 = new Path(path.get("Step5Input2"));
        Path output = new Path(path.get("Step5Output"));
        // 删除上一次的输出
        HDFSFile hdfs = new HDFSFile(new Path(Recommend.HDFS));
        hdfs.delFile(output);
        // 设置作业参数
        Job job = new Job(conf);
        job.setJarByClass(Step5.class);

        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);

        job.setMapperClass(Step5_FilterSortMapper.class);
        job.setReducerClass(Step5_FilterSortReducer.class);

        job.setInputFormatClass(TextInputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);

        FileInputFormat.setInputPaths(job, input1,input2);
        FileOutputFormat.setOutputPath(job, output);

        job.waitForCompletion(true);
    }
}
```

11、编写 Recommend.java 文件，实现推荐系统入口程序的编写。注意需把 small2.csv 文件放入到工程中，并根据实际环境需要修改步骤 1 中的 HDFS 地址。代码如下：

```
package recommend;

import java.util.HashMap;
import java.util.Map;
import java.util.regex.Pattern;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;

/**
 * 基于物品的协同推荐系统
 */
```

```
public class Recommend {
    //HDFS 文件路径
    public static final String HDFS = "hdfs://"+HDFSTOOL.MASTERNAME+":"+HDFSTOOL.DFSPORT;
    //MapReduce 分割符为\t 和,
    public static final Pattern DELIMITER = Pattern.compile("[\t,]");
    //入口函数
    /**
     * @param args
     * @throws Exception
     */
    public static void main(String[] args) throws Exception {
        Map<String, String> path = new HashMap<String, String>();
        //本地数据位置
        String localData = "./small2.csv";
        //本地数据路径
        path.put("data", localData);
        //步骤 1 的输入输出路径
        path.put("Step1Input", HDFS + "/user/jiahui/recommend");
        path.put("Step1Output", path.get("Step1Input") + "/step1");
        //步骤 2 的输入输出路径
        path.put("Step2Input", path.get("Step1Output"));
        path.put("Step2Output", path.get("Step1Input") + "/step2");
        //步骤 3_1 的输入输出路径
        path.put("Step3Input1", path.get("Step1Output"));
        path.put("Step3Output1", path.get("Step1Input") + "/step3_1");
        //步骤 3_2 的输入输出路径
        path.put("Step3Input2", path.get("Step2Output"));
        path.put("Step3Output2", path.get("Step1Input") + "/step3_2");
        //步骤 4 的输入输出路径
        path.put("Step4_1Input1", path.get("Step3Output1"));
        path.put("Step4_1Input2", path.get("Step3Output2"));
        path.put("Step4_1Output", path.get("Step1Input") + "/step4_1");
        path.put("Step4_2Input", path.get("Step4_1Output"));
        path.put("Step4_2Output", path.get("Step1Input") + "/step4_2");
        //步骤 5 的输入输出路径
        path.put("Step5Input1", path.get("Step4_2Output"));
        path.put("Step5Input2", path.get("Step1Input")+"/small2.csv");
        path.put("Step5Output", path.get("Step1Input") + "/step5");



        Step1.run(path);
        Step2.run(path);
```

```
            Step3.run1(path);
            Step3.run2(path);
            Step4_Update.run(path);
            Step4_Update2.run(path);
            Step5.run(path);

            //输出结果到终端
            HDFSFile hdfs = new HDFSFile(new Path(HDFS));
            //Step1 的输出结果
            System.out.println(path.get("Step1Output")+"/part-r-00000");
            hdfs.readFile(new Path(path.get("Step1Output")+"/part-r-00000"));
            //Step2 的输出结果
            System.out.println(path.get("Step2Output")+"/part-r-00000");
            hdfs.readFile(new Path(path.get("Step2Output")+"/part-r-00000"));
            //Step3_1 的输出结果
            System.out.println(path.get("Step3Output1")+"/part-r-00000");
            hdfs.readFile(new Path(path.get("Step3Output1")+"/part-r-00000"));
            //Step3_2 的输出结果
            System.out.println(path.get("Step3Output2")+"/part-r-00000");
            hdfs.readFile(new Path(path.get("Step3Output2")+"/part-r-00000"));
            //Step4_1 的输出结果
            System.out.println(path.get("Step4_1Output")+"/part-r-00000");
            hdfs.readFile(new Path(path.get("Step4_1Output")+"/part-r-00000"));
            //System.exit(0);
            //Step4_2 的输出结果
            System.out.println(path.get("Step4_2Output")+"/part-r-00000");
            hdfs.readFile(new Path(path.get("Step4_2Output")+"/part-r-00000"));
            //Step5 的输出结果
            System.out.println(path.get("Step5Output")+"/part-r-00000");
            hdfs.readFile(new Path(path.get("Step5Output")+"/part-r-00000"));

            System.exit(0);

        }
        public static Configuration config() {
            Configuration conf = new Configuration();
            return conf;
        }
}
```

12、运行 Recommend.java 程序，实验最后可展示每个用户推荐哪部电影。实验结果如下：

1,101---
25.0
10.0
9.0
1,102---
7.5
15.0
9.0
1,103---
10.0
9.0
20.0
1,104---
20.0
6.0
7.5
1,105---
2.5
10.0
3.0
1,106---
5.0
10.0
3.0
1,107---
5.0
2,101---
10.0
8.0
7.5
20.0
2,102---
7.5
6.0
4.0
15.0
2,103---
6.0
20.0
7.5
8.0
2,104---
15.0

8.0
5.0
8.0
2,105---
4.0
5.0
4.0
2.5
2,106---
10.0
4.0
4.0
2.5
2,107---
2.0
2.0
3,101---
9.0
10.0
5.0
16.0
3,102---
4.5
6.0
8.0
3,103---
4.5
8.0
12.0
3,104---
8.0
9.0
5.0
16.0
3,105---
9.0
4.0
5.0
8.0
3,106---
8.0
4.0
4.5

3,107---
5.0
4.5
4.0
2.0
4,101---
25.0
8.0
12.0
18.0
4,102---
9.0
4.0
9.0
15.0
4,103---
8.0
12.0
13.5
20.0
4,104---
18.0
20.0
9.0
8.0
4,105---
4.0
10.0
3.0
9.0
4,106---
8.0
6.0
9.0
10.0
4,107---
4.5
5.0
5,101---
8.0
7.0
20.0
9.0

16.0
8.0
5,102---
9.0
4.0
3.5
6.0
12.0
8.0
5,103---
8.0
12.0
9.0
3.5
16.0
8.0
5,104---
16.0
6.0
16.0
7.0
8.0
6.0
5,105---
8.0
7.0
8.0
3.0
4.0
2.0
5,106---
8.0
3.5
3.0
8.0
4.0
8.0
5,107---
4.0
3.5
4.0
1-----
S:101,5.0

S:102,3.0
S:103,2.5
W:102,31.5
W:103,39.0
W:104,33.5
W:101,44.0
W:105,15.5
W:106,18.0
W:107,5.0
2-----
W:101,45.5
W:102,32.5
W:103,41.5
W:104,36.0
W:105,15.5
W:106,20.5
W:107,4.0
S:101,2.0
S:102,2.5
S:103,5.0
S:104,2.0
3-----
S:107,5.0
S:105,4.5
S:104,4.0
S:101,2.0
W:104,38.0
W:107,15.5
W:106,16.5
W:105,26.0
W:103,24.5
W:102,18.5
W:101,40.0
4-----
W:101,63.0
W:102,37.0
W:103,53.5
W:104,55.0
W:105,26.0
W:106,33.0
W:107,9.5
S:106,4.0
S:103,3.0

S:101,5.0
S:104,4.5
5-----
S:104,4.0
S:105,3.5
S:106,4.0
S:101,4.0
S:102,3.0
S:103,2.0
W:101,68.0
W:102,42.5
W:103,56.5
W:104,59.0
W:105,32.0
W:106,34.5
W:107,11.5
hdfs://192.168.24.91:9000/user/jiahui/recommend/step1/part-r-00000
1 101:5.0,102:3.0,103:2.5
2 101:2.0,102:2.5,103:5.0,104:2.0
3 107:5.0,105:4.5,104:4.0,101:2.0
4 106:4.0,103:3.0,101:5.0,104:4.5
5 104:4.0,105:3.5,106:4.0,101:4.0,102:3.0,103:2.0

hdfs://192.168.24.91:9000/user/jiahui/recommend/step2/part-r-00000
101:101 5
101:102 3
101:103 4
101:104 4
101:105 2
101:106 2
101:107 1
102:101 3
102:102 3
102:103 3
102:104 2
102:105 1
102:106 1
103:101 4
103:102 3
103:103 4
103:104 3
103:105 1
103:106 2

104:101 4
104:102 2
104:103 3
104:104 4
104:105 2
104:106 2
104:107 1
105:101 2
105:102 1
105:103 1
105:104 2
105:105 2
105:106 1
105:107 1
106:101 2
106:102 1
106:103 2
106:104 2
106:105 1
106:106 2
107:101 1
107:104 1
107:105 1
107:107 1

hdfs://192.168.24.91:9000/user/jiahui/recommend/step3_1/part-r-00000
101 1:5.0
101 5:4.0
101 4:5.0
101 3:2.0
101 2:2.0
102 2:2.5
102 5:3.0
102 1:3.0
103 1:2.5
103 4:3.0
103 5:2.0
103 2:5.0
104 3:4.0
104 4:4.5
104 5:4.0
104 2:2.0
105 5:3.5

105 3:4.5
106 4:4.0
106 5:4.0
107 3:5.0

hdfs://192.168.24.91:9000/user/jiahui/recommend/step3_2/part-r-00000
101:101 5
101:102 3
101:103 4
101:104 4
101:105 2
101:106 2
101:107 1
102:101 3
102:102 3
102:103 3
102:104 2
102:105 1
102:106 1
103:101 4
103:102 3
103:103 4
103:104 3
103:105 1
103:106 2
104:101 4
104:102 2
104:103 3
104:104 4
104:105 2
104:106 2
104:107 1
105:101 2
105:102 1
105:103 1
105:104 2
105:105 2
105:106 1
105:107 1
106:101 2
106:102 1
106:103 2
106:104 2

106:105 1
106:106 2
107:101 1
107:104 1
107:105 1
107:107 1

hdfs://192.168.24.91:9000/user/jiahui/recommend/step4_1/part-r-00000
1 101,25.0
2 101,10.0
3 101,10.0
4 101,25.0
5 101,20.0
1 102,15.0
2 102,6.0
3 102,6.0
4 102,15.0
5 102,12.0
1 103,20.0
2 103,8.0
3 103,8.0
4 103,20.0
5 103,16.0
1 104,20.0
2 104,8.0
3 104,8.0
4 104,20.0
5 104,16.0
1 105,10.0
2 105,4.0
3 105,4.0
4 105,10.0
5 105,8.0
1 106,10.0
2 106,4.0
3 106,4.0
4 106,10.0
5 106,8.0
1 107,5.0
2 107,2.0
3 107,2.0
4 107,5.0
5 107,4.0

1	101,9.0
2	101,7.5
5	101,9.0
1	102,9.0
2	102,7.5
5	102,9.0
1	103,9.0
2	103,7.5
5	103,9.0
1	104,6.0
2	104,5.0
5	104,6.0
1	105,3.0
2	105,2.5
5	105,3.0
1	106,3.0
2	106,2.5
5	106,3.0
1	101,10.0
2	101,20.0
4	101,12.0
5	101,8.0
1	102,7.5
2	102,15.0
4	102,9.0
5	102,6.0
1	103,10.0
2	103,20.0
4	103,12.0
5	103,8.0
1	104,7.5
2	104,15.0
4	104,9.0
5	104,6.0
1	105,2.5
2	105,5.0
4	105,3.0
5	105,2.0
1	106,5.0
2	106,10.0
4	106,6.0
5	106,4.0
2	101,8.0

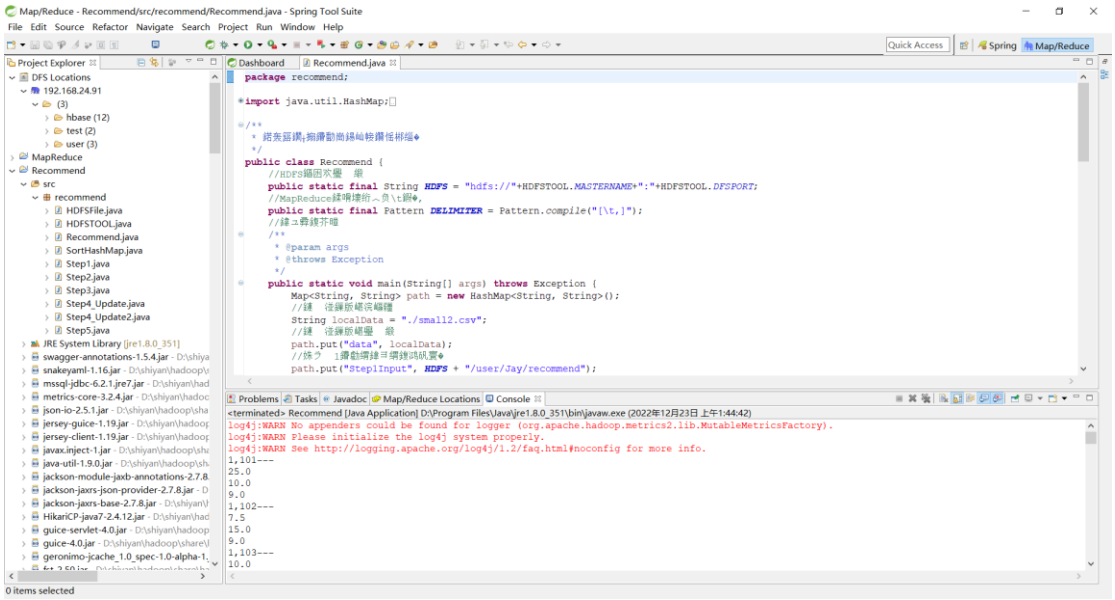
3	101,16.0
4	101,18.0
5	101,16.0
2	102,4.0
3	102,8.0
4	102,9.0
5	102,8.0
2	103,6.0
3	103,12.0
4	103,13.5
5	103,12.0
2	104,8.0
3	104,16.0
4	104,18.0
5	104,16.0
2	105,4.0
3	105,8.0
4	105,9.0
5	105,8.0
2	106,4.0
3	106,8.0
4	106,9.0
5	106,8.0
2	107,2.0
3	107,4.0
4	107,4.5
5	107,4.0
3	101,9.0
5	101,7.0
3	102,4.5
5	102,3.5
3	103,4.5
5	103,3.5
3	104,9.0
5	104,7.0
3	105,9.0
5	105,7.0
3	106,4.5
5	106,3.5
3	107,4.5
5	107,3.5
4	101,8.0
5	101,8.0

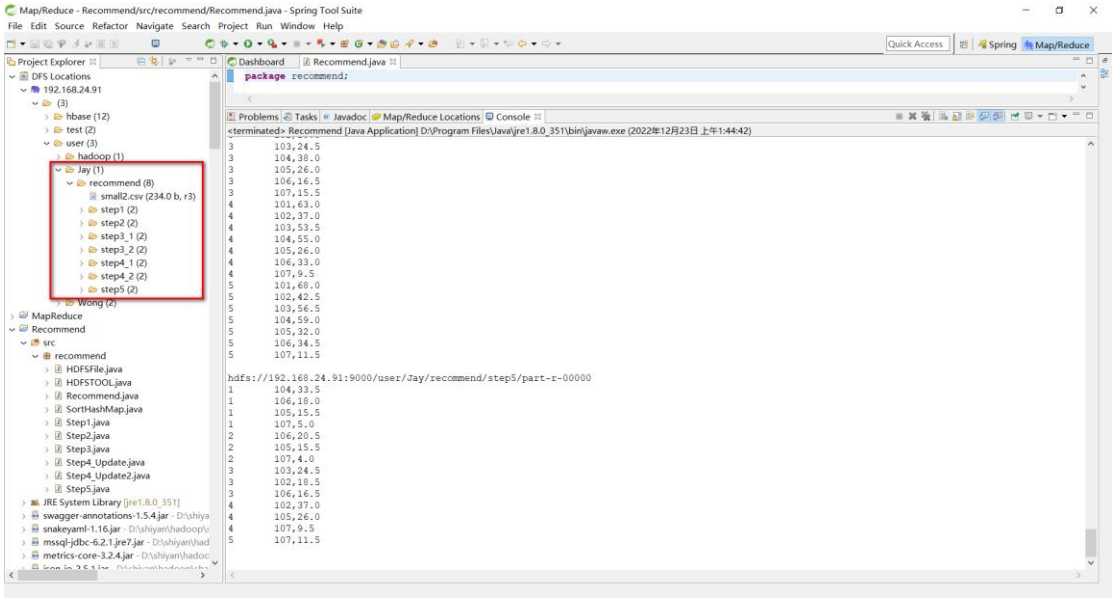
4 102,4.0
5 102,4.0
4 103,8.0
5 103,8.0
4 104,8.0
5 104,8.0
4 105,4.0
5 105,4.0
4 106,8.0
5 106,8.0
3 101,5.0
3 104,5.0
3 105,5.0
3 107,5.0

hdfs://192.168.24.91:9000/user/jiahui/recommend/step4_2/part-r-00000
1 101,44.0
1 102,31.5
1 103,39.0
1 104,33.5
1 105,15.5
1 106,18.0
1 107,5.0
2 101,45.5
2 102,32.5
2 103,41.5
2 104,36.0
2 105,15.5
2 106,20.5
2 107,4.0
3 101,40.0
3 102,18.5
3 103,24.5
3 104,38.0
3 105,26.0
3 106,16.5
3 107,15.5
4 101,63.0
4 102,37.0
4 103,53.5
4 104,55.0
4 105,26.0
4 106,33.0

4 107,9.5
5 101,68.0
5 102,42.5
5 103,56.5
5 104,59.0
5 105,32.0
5 106,34.5
5 107,11.5

hdfs://192.168.24.91:9000/user/jiahui/recommend/step5/part-r-00000
1 104,33.5
1 106,18.0
1 105,15.5
1 107,5.0
2 106,20.5
2 105,15.5
2 107,4.0
3 103,24.5
3 102,18.5
3 106,16.5
4 102,37.0
4 105,26.0
4 107,9.5
5 107,11.5





运行结果与所给实验结果一致。