读

Java MapRduce

推荐系统

```
public class StartRun {
    public static void main(String[] args) {
        Configuration config = new Configuration();

        Map<String, String> paths = new HashMap<String, String>();
        paths.put("Step1Input", "F:/code/hadoop/data/tuijian");
        paths.put("Step1Output", "F:/code/hadoop/data/tuijian/output1");
        paths.put("Step2Input", paths.get("Step1Output"));
        paths.put("Step2Output", "F:/code/hadoop/data/tuijian/output2");
        paths.put("Step3Input", paths.get("Step2Output"));
        paths.put("Step3Output", "F:/code/hadoop/data/tuijian/output3");
        paths.put("Step4Input1", paths.get("Step2Output"));
        paths.put("Step4Input2", paths.get("Step3Output"));
        paths.put("Step4Output", "F:/code/hadoop/data/tuijian/output4");
        paths.put("Step5Input", paths.get("Step4Output"));
        paths.put("Step5Output", "F:/code/hadoop/data/tuijian/output5");
        paths.put("Step6Input", paths.get("Step5Output"));
        paths.put("Step6Output", "F:/code/hadoop/data/tuijian/output6");

        Step1.run(config, paths);
        Step2.run(config, paths);
        Step3.run(config, paths);
        Step4.run(config, paths);
        Step5.run(config, paths);
        Step6.run(config, paths);
    }

    public static Map<String, Integer> R = new HashMap<String, Integer>();
    static {
        R.put("click", 1);
        R.put("collect", 2);
        R.put("cart", 3);
        R.put("alipay", 4);
    }
}
public class Step1 {


    public static boolean run(Configuration config,Map<String, String> paths){
        try {
            FileSystem fs =FileSystem.get(config);
            Job job =Job.getInstance(config);
            job.setJobName("step1");
            job.setJarByClass(Step1.class);

            job.setMapperClass(Step1_Mapper.class);
            job.setMapOutputKeyClass(Text.class);
```

```
            job.setMapOutputValueClass(NullWritable.class);

            job.setReducerClass(Step1_Reducer.class);

            FileInputFormat.addInputPath(job, new
Path(paths.get("Step1Input")));
            Path outpath=new Path(paths.get("Step1Output"));
            if(fs.exists(outpath)){
                fs.delete(outpath,true);
            }
            FileOutputFormat.setOutputPath(job, outpath);

            boolean f= job.waitForCompletion(true);
            return f;
        } catch (Exception e) {
            e.printStackTrace();
        }
        return false;
    }

    static class Step1_Mapper extends Mapper<LongWritable, Text, Text,
NullWritable>{
        protected void map(LongWritable key, Text value, Context context)
                throws IOException, InterruptedException {
            if(key.get()!=0){
                context.write(value, NullWritable.get());
            }
        }
    }


     static class Step1_Reducer extends Reducer<Text, IntWritable, Text,
NullWritable>{
        protected void reduce(Text key, Iterable<IntWritable> i, Context
context)
                    throws IOException, InterruptedException {
            context.write(key,NullWritable.get());
        }
    }
}
public class Step2 {
    public static boolean run(Configuration config,Map<String, String> paths){
        try {
            config.set("mapred.jar",
"C:\\Users\\Administrator\\Desktop\\wc.jar");
            FileSystem fs =FileSystem.get(config);
            Job job =Job.getInstance(config);
            job.setJobName("step2");
            job.setJarByClass(StartRun.class);

            job.setMapperClass(Step2_Mapper.class);
            job.setReducerClass(Step2_Reducer.class);

            job.setMapOutputKeyClass(Text.class);
            job.setMapOutputValueClass(Text.class);

            FileInputFormat.addInputPath(job, new
Path(paths.get("Step2Input")));
```

```
            Path outpath=new Path(paths.get("Step2Output"));
            if(fs.exists(outpath)){
                fs.delete(outpath,true);
            }
            FileOutputFormat.setOutputPath(job, outpath);

            boolean f= job.waitForCompletion(true);
            return f;
        } catch (Exception e) {
            e.printStackTrace();
        }
        return false;
    }

     static class Step2_Mapper extends Mapper<LongWritable, Text, Text, Text>{
        protected void map(LongWritable key, Text value,
                Context context)
                throws IOException, InterruptedException {
            String[]  tokens=value.toString().split(",");
            String item=tokens[0];
            String user=tokens[1];
            String action =tokens[2];
            Text k= new Text(user);
            Integer rv =StartRun.R.get(action);
            if(rv!=null){
            Text v =new Text(item+":"+ rv.intValue());
            context.write(k, v);
        }
    }
     static class Step2_Reducer extends Reducer<Text, Text, Text, Text>{

            protected void reduce(Text key, Iterable<Text> i,
                    Context context)
                    throws IOException, InterruptedException {
                    //key:u1
                     //value: i1: 2
                Map<String, Integer> r =new HashMap<String, Integer>();

                for(Text value :i){
                    String[] vs =value.toString().split(":");//value: i1:2
                    String item=vs[0];//i1
                    Integer action=Integer.parseInt(vs[1]);//2

                    action = ((Integer) (r.get(item)==null?
0:r.get(item))).intValue() + action;
                    r.put(item,action);
                }
                StringBuffer sb =new StringBuffer();
                for(Entry<String, Integer> entry :r.entrySet() ){

sb.append(entry.getKey()+":"+entry.getValue().intValue()+",");
                }

                context.write(key,new
Text(sb.toString().substring(0,sb.toString().length()-1)));
            }
        }
}
```

```
public class Step3 {
     private final static Text K = new Text();
     private final static IntWritable V = new IntWritable(1);

    public static boolean run(Configuration config,Map<String, String> paths){
        try {
            FileSystem fs =FileSystem.get(config);
            Job job =Job.getInstance(config);
            job.setJobName("step3");
            job.setJarByClass(StartRun.class);
            job.setMapperClass(Step3_Mapper.class);
            job.setReducerClass(Step3_Reducer.class);
            job.setCombinerClass(Step3_Reducer.class);
            job.setMapOutputKeyClass(Text.class);
            job.setMapOutputValueClass(IntWritable.class);



            FileInputFormat.addInputPath(job, new
Path(paths.get("Step3Input")));
            Path outpath=new Path(paths.get("Step3Output"));
            if(fs.exists(outpath)){
                fs.delete(outpath,true);
            }
            FileOutputFormat.setOutputPath(job, outpath);

            boolean f= job.waitForCompletion(true);
            return f;
        } catch (Exception e) {
            e.printStackTrace();
        }
        return false;
    }

     static class Step3_Mapper extends Mapper<LongWritable, Text, Text,
IntWritable>{

        protected void map(LongWritable key, Text value,
                Context context)
                throws IOException, InterruptedException {
            //u2727 i468:2,i446:3
            String[]  tokens=value.toString().split("\t");
            String[] items =tokens[1].split(",");//i468:2   i446:3
            for (int i = 0; i < items.length; i++) {
                String itemA = items[i].split(":")[0];
                for (int j = 0; j < items.length; j++) {
                    String itemB = items[j].split(":")[0];//i468
                    K.set(itemA+":"+itemB);
                    context.write(K, V);
                }
            }

        }
    }


     static class Step3_Reducer extends Reducer<Text, IntWritable, Text,
IntWritable>{
```

```
            protected void reduce(Text key, Iterable<IntWritable> i,
                    Context context)
                    throws IOException, InterruptedException {
                int sum =0;
                for(IntWritable v :i ){
                    sum =sum+v.get();
                }
                V.set(sum); //i468:i446   4
                context.write(key, V);
            }
        }
}
public class Step4 {

    public static boolean run(Configuration config, Map<String, String> paths) {
        try {
            FileSystem fs = FileSystem.get(config);
            Job job = Job.getInstance(config);
            job.setJobName("step4");
            job.setJarByClass(StartRun.class);
            job.setMapperClass(Step4_Mapper.class);
            job.setReducerClass(Step4_Reducer.class);
            job.setMapOutputKeyClass(Text.class);
            job.setMapOutputValueClass(Text.class);

            // FileInputFormat.addInputPath(job, new
            // Path(paths.get("Step4Input")));
            FileInputFormat.setInputPaths(job,
                    new Path[] { new Path(paths.get("Step4Input1")),
                            new Path(paths.get("Step4Input2")) });
            Path outpath = new Path(paths.get("Step4Output"));
            if (fs.exists(outpath)) {
                fs.delete(outpath, true);
            }
            FileOutputFormat.setOutputPath(job, outpath);

            boolean f = job.waitForCompletion(true);
            return f;
        } catch (Exception e) {
            e.printStackTrace();
        }
        return false;
    }

    static class Step4_Mapper extends Mapper<LongWritable, Text, Text, Text> {
        private String flag;

        protected void setup(Context context) throws IOException,
                InterruptedException {
            FileSplit split = (FileSplit) context.getInputSplit();
            flag = split.getPath().getParent().getName();

            System.out.println(flag + "**********************");
        }

        protected void map(LongWritable key, Text value, Context context)
                throws IOException, InterruptedException {
```

```
            String[] tokens = Pattern.compile("[\t,]").split(value.toString());

            if (flag.equals("output3")) {//i100:i100    3     同现矩阵
                String[] v1 = tokens[0].split(":");
                String itemID1 = v1[0];
                String itemID2 = v1[1];
                String num = tokens[1]; //一同出现的次数

                Text k = new Text(itemID1);//i100
                Text v = new Text("A:" + itemID2 + "," + num);// A:i109,3
                context.write(k, v);// key:i100   value : A:i100,3

            } else if (flag.equals("output2")) {// u13  i160:1,i2332:3
                String userID = tokens[0];//u13
                for (int i = 1; i < tokens.length; i++) {//i468:2,i446:3
                    String[] vector = tokens[i].split(":");
                    String itemID = vector[0];// id i468
                    String pref = vector[1];// 2

                    Text k = new Text(itemID);
                    Text v = new Text("B:" + userID + "," + pref); // B:u401,2
                    context.write(k, v);//key:i468   value: B:u401,2
                }
            }
        }
    }

    static class Step4_Reducer extends Reducer<Text, Text, Text, Text> {
        protected void reduce(Text key, Iterable<Text> values, Context context)
                throws IOException, InterruptedException {
            Map<String, Integer> mapA = new HashMap<String, Integer>();
            Map<String, Integer> mapB = new HashMap<String, Integer>();

            for (Text line : values) {
                String val = line.toString();
                if (val.startsWith("A:")) {
                //  key:i100   value : A:i100,3
                    String[] kv = Pattern.compile("[\t,]").split(
                            val.substring(2));//从第二个位置之后的子串
                    try {
                        mapA.put(kv[0], Integer.parseInt(kv[1]));
                    } catch (Exception e) {
                        e.printStackTrace();
                    }

                } else if (val.startsWith("B:")) {
                    ////key:i468   value: B:u401,2
                    String[] kv = Pattern.compile("[\t,]").split(
                            val.substring(2));
                    try {
                        mapB.put(kv[0], Integer.parseInt(kv[1]));
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                }
            }

            double result = 0;
```

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

                int num = mapA.get(mapk).intValue();
                Iterator<String> iterb = mapB.keySet().iterator();
                while (iterb.hasNext()) {
                    String mapkb = iterb.next();// userID
                    int pref = mapB.get(mapkb).intValue();
                    result = num * pref;

                    Text k = new Text(mapkb);//userID
                    Text v = new Text(mapk + "," + result);  //itemID, result
                    context.write(k, v);
                }
            }
        }
    }
}
public class Step5 {
    private final static Text K = new Text();
    private final static Text V = new Text();

    public static boolean run(Configuration config, Map<String, String> paths) {
        try {
            FileSystem fs = FileSystem.get(config);
            Job job = Job.getInstance(config);
            job.setJobName("step5");
            job.setJarByClass(StartRun.class);
            job.setMapperClass(Step5_Mapper.class);
            job.setReducerClass(Step5_Reducer.class);
            job.setMapOutputKeyClass(Text.class);
            job.setMapOutputValueClass(Text.class);

            FileInputFormat
                    .addInputPath(job, new Path(paths.get("Step5Input")));
            Path outpath = new Path(paths.get("Step5Output"));
            if (fs.exists(outpath)) {
                fs.delete(outpath, true);
            }
            FileOutputFormat.setOutputPath(job, outpath);

            boolean f = job.waitForCompletion(true);
            return f;
        } catch (Exception e) {
            e.printStackTrace();
        }
        return false;
    }

    static class Step5_Mapper extends Mapper<LongWritable, Text, Text, Text> {

        protected void map(LongWritable key, Text value, Context context)
                throws IOException, InterruptedException {
            String[] tokens = Pattern.compile("[\t,]").split(value.toString());
            Text k = new Text(tokens[0]);// u2732   i405,2.0
            Text v = new Text(tokens[1] + "," + tokens[2]);
            context.write(k, v);
```

```
        }
    }

    static class Step5_Reducer extends Reducer<Text, Text, Text, Text> {
        protected void reduce(Text key, Iterable<Text> values, Context context)
                throws IOException, InterruptedException {
            Map<String, Double> map = new HashMap<String, Double>();//

            for (Text line : values) {// i9,4.0
                String[] tokens = line.toString().split(",");
                String itemID = tokens[0];
                Double score = Double.parseDouble(tokens[1]);

                if (map.containsKey(itemID)) {
                    map.put(itemID, map.get(itemID) + score);//
                } 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);
            }

        }

    }
}
public class Step6 {
    private final static Text K = new Text();
    private final static Text V = new Text();

    public static boolean run(Configuration config, Map<String, String> paths) {
        try {
            FileSystem fs = FileSystem.get(config);
            Job job = Job.getInstance(config);
            job.setJobName("step6");
            job.setJarByClass(StartRun.class);
            job.setMapperClass(Step6_Mapper.class);
            job.setReducerClass(Step6_Reducer.class);
            job.setSortComparatorClass(NumSort.class);
            job.setGroupingComparatorClass(UserGroup.class);
            job.setMapOutputKeyClass(PairWritable.class);
            job.setMapOutputValueClass(Text.class);

            FileInputFormat
                    .addInputPath(job, new Path(paths.get("Step6Input")));
            Path outpath = new Path(paths.get("Step6Output"));
            if (fs.exists(outpath)) {
                fs.delete(outpath, true);
            }
            FileOutputFormat.setOutputPath(job, outpath);

            boolean f = job.waitForCompletion(true);
```

```
            return f;
        } catch (Exception e) {
            e.printStackTrace();
        }
        return false;
    }

    static class Step6_Mapper extends Mapper<LongWritable, Text, PairWritable,
Text> {

        protected void map(LongWritable key, Text value, Context context)
                throws IOException, InterruptedException {
            String[] tokens = Pattern.compile("[\t,]").split(value.toString());
            String u = tokens[0];
            String item = tokens[1];
            String num = tokens[2];
            PairWritable k =new PairWritable();
            k.setUid(u);
            k.setNum(Double.parseDouble(num));
            V.set(item+":"+num);
            context.write(k, V);

        }
    }

    static class Step6_Reducer extends Reducer<PairWritable, Text, Text, Text> {
        protected void reduce(PairWritable key, Iterable<Text> values, Context
context)
                throws IOException, InterruptedException {
            int i=0;
            StringBuffer sb =new StringBuffer();
            for(Text v :values){
                if(i==3)
                    break;
                sb.append(v.toString()+",");
                i++;
            }
            K.set(key.getUid());
            V.set(sb.toString());
            context.write(K, V);
        }

    }

    static class PairWritable implements WritableComparable<PairWritable>{
        private String uid;
        private double num;
        public void write(DataOutput out) throws IOException {
            out.writeUTF(uid);
            out.writeDouble(num);
        }

        public void readFields(DataInput in) throws IOException {
            this.uid=in.readUTF();
            this.num=in.readDouble();
        }

        public int compareTo(PairWritable o) {
```

```
            int r =this.uid.compareTo(o.getUid());
            if(r==0){
                return Double.compare(this.num, o.getNum());
            }
            return r;
        }

        public String getUid() {
            return uid;
        }

        public void setUid(String uid) {
            this.uid = uid;
        }

        public double getNum() {
            return num;
        }

        public void setNum(double num) {
            this.num = num;
        }

    }

    static class NumSort extends WritableComparator{
        public NumSort(){
            super(PairWritable.class,true);
        }

        public int compare(WritableComparable a, WritableComparable b) {
            PairWritable o1 =(PairWritable) a;
            PairWritable o2 =(PairWritable) b;

            int r =o1.getUid().compareTo(o2.getUid());
            if(r==0){
                return -Double.compare(o1.getNum(), o2.getNum());
            }
            return r;
        }
    }

    static class UserGroup extends WritableComparator{
        public UserGroup(){
            super(PairWritable.class,true);
        }

        public int compare(WritableComparable a, WritableComparable b) {
            PairWritable o1 =(PairWritable) a;
            PairWritable o2 =(PairWritable) b;
            return o1.getUid().compareTo(o2.getUid());
        }
    }
}
```

Hive

HiveQL语句实现WordCount算法

```
create table docs(line string);
    load data inpath 'input' overwrite into table docs;
    create table word_count as
    select word, count(1) as count from
    (select explode(split(line,' '))as word from docs)
    group by word
    order by word;
```

创建Hive外部表

```
create external table sogoulogs(id string,datatime string,userid
string,searchname string,retorder string,cliorder string,cliurl string)  STORED
BY  'org.apache.hadoop.hive.hbase.HBaseStorageHandler' WITH
SERDEPROPERTIES("hbase.columns.mapping" =
":key,info:datatime,info:userid,info:searchname,info:retorder,info:cliorder,info
:cliurl") TBLPROPERTIES("hbase.table.name" = "sogoulogs");
```

基于Hive的用户行为数据离线分析

统计新闻话题总量

```
select count(distinct searchname) from sogoulogs;
```

统计新闻话题浏览量排行

```
select searchname,count(*) as rank from sogoulogs  group by searchname order by
rank  desc limit 10;
```

统计新闻浏览量不同时段排行

```
select substr(datatime,0,5),count(substr(datatime,0,5)) as counter from sogoulogs
group by substr(datatime,0,5) order by counter desc limit 10;
```

分析链接排名与用户点击的相关性

```
select page_rank,count(*) as num from sogoulogs group by page_rank
Having page_rank is not null
and page_rank <> 0
Order by page_rank
limit 10;
```

Spark RDD