CS 549 Assignment-5 Page Rank in Hadoop

Dhruvit Patel (CWID: 1040432)

Abstract

In this report, I have shown how I implemented Page-Ranking algorithm to find most popular pages in a set of Web Pages in a dump of Wikipedia pages.

In the process of finding most popular pages following sub operations need to be implemented:

- init
- iter
- diff
- finish
- composite

1) DiffMap1.java

```
/**
 *  TODO: read node-rank pair and emit: key:node, value:rank
 */
//added
String[] noderank = sections[0].split("\\+");// split node+rank;
context.write(new Text(noderank[0]),  new Text(noderank[1])); // emit node, rank
```

2) DiffMap2.java

```
/*
 * TODO: emit: key:"Difference" value:difference calculated in DiffRed1
 */
//added
String[] noderank = s.split("\t+");
context.write(new Text("Difference"), new Text(noderank[1])); // emit difference
```

3) DiffRed1.java

```
/*
 * TODO: The list of values should contain two ranks.  Compute and output their difference.
 */
//added
Iterator<Text> iterator = values.iterator();
double diff = 0; // default diff has max value
// caculate rank1
if(iterator.hasNext()) {
    ranks[0] = Double.valueOf(iterator.next().toString());
}
// caculate rank2
if(iterator.hasNext()) {
    ranks[1] = Double.valueOf(iterator.next().toString());
}
// caculate diff
diff = Math.abs(ranks[0] - ranks[1]);
System.out.println( key.toString() + "  " + diff);
context.write(key, new Text(String.valueOf(diff)));
```

4) DiffRed2.java

```
/*
 * TODO: Compute and emit the maximum of the differences
 */
//added
Iterator<Text> iterator = values.iterator();
// caculate the maxium difference and output the result
while(iterator.hasNext()) {
    double diff = Double.valueOf(iterator.next().toString());
    diff_max = diff_max > diff ? diff_max : diff;
}
context.write(new Text(""), new Text(String.valueOf(diff_max)));
```

5) FinMappper.java

```
/*
 * TODO output key:-rank, value: node
 * See IterMapper for hints on parsing the output of IterReducer.
 */
//added
String[] sections = line.split("\t"); // nodeId+nodeName | rank

if (sections.length > 2) // Checks if the data is in the incorrect format
{
    throw new IOException("Incorrect data format");
}
if (sections.length != 2) {
    return;
}
// to reverse shuffle the reducer, we need minus rank with 0
context.write(new DoubleWritable(0 - Double.valueOf(sections[1])), new Text(sections[0]));
```

6) FinReducer.java

```
/*
 * TODO: For each value, emit: key:value, value:-rank
 */
//added
Iterator<Text> iterator = values.iterator();
String node;
while(iterator.hasNext()) {
    node = iterator.next().toString();
    // convert -rank back to rank
    context.write(new Text(node), new Text(String.valueOf(0 - key.get())));
}
```

7) InitMapper.java

```
/*
 * TODO: Just echo the input, since it is already in adjacency list format.
 */
//added
/*
 * split the line by symbol ":", and output key, adjacent list to reducer
 */
String[] pair = line.split(":");
if(pair != null && pair.length == 2) {
    context.write(new Text(pair[0].trim()), new Text(pair[1]));
}
```

8) InitReducer.java

```
/*
 * TODO: Output key: node+rank, value: adjacency list
 */
//added
/*
 * Since default rank is 1, so we need only output node+rank and adjacency list
 */
int defualtrank = 1;
Iterator<Text> v = values.iterator();
while(v.hasNext()) {
    // emit node+rank, value
    context.write(new Text(key + "+" + defualtrank), v.next());
}
```

9) IterMapper.java

```
/*
 * TODO: emit key: adj vertex, value: computed weight.
 *
 * Remember to also emit the input adjacency list for this node!
 * Put a marker on the string value to indicate it is an adjacency list.
 */
//added
String[] noderank = sections[0].split("\\+"); // split node+rank
String node = String.valueOf(noderank[0]);
double rank = Double.valueOf(noderank[1]);
String ajacentlist = sections[1].toString().trim(); // keep ajacent list

String[] ajacentnodes = ajacentlist.split(" ");
int N = ajacentnodes.length; // outgoing links number
// 1/n * rank
double weightOfPage = (double)1/N * rank; // calculate current page weight if outgoing to other links
for(String ajacentnode : ajacentnodes) {
    context.write(new Text(ajacentnode), new Text(String.valueOf(weightOfPage)));
}
// at the same time, emit current node's ajacent list with marker "ADJ:"
context.write(new Text(node), new Text(PageRankDriver.MARKER + sections[1]));
```

10) IterReducer.java

```
/*
 * TODO: emit key:node+rank, value: adjacency list
 * Use PageRank algorithm to compute rank from weights contributed by incoming edges.
 * Remember that one of the values will be marked as the adjacency list for the node.
 */

//added
Iterator<Text> iterator = values.iterator();
double currentRank = 0; // default rank is 1 - d
String ajacentlist = "";
while(iterator.hasNext()) {
    String line = iterator.next().toString();
    if(!line.startsWith(PageRankDriver.MARKER)) {
        currentRank += Double.valueOf(line);
    } else {
        ajacentlist = line.replaceAll(PageRankDriver.MARKER, "");
    }
}
// (1-d) + d * sum(bac
currentRank = 1 - d + currentRank * d;
context.write(new Text(key + "+" + currentRank), new Text(ajacentlist));
```