。了，



**《基于平台的编程》实践报告**

|  |  |
| --- | --- |
| 姓 名： | 薛旭升 |
| 学 院： | 计算机科学与技术 |
| 专 业： | 计算机科学与技术 |
| 班 级： | CS1606 |
| 学 号： | U201614683 |
| 指导老师： | 刘海坤 |

2019年11月12日

**目 录**

[1.试验任务一 PageRank算法 2](#_Toc22918)

[1.1 实验环境 2](#_Toc14612)

[1.2 实验目的 2](#_Toc14283)

[1.3 设计思路 2](#_Toc14831)

[1.3.1 PageRank算法 2](#_Toc30043)

[1.3.2 MapReduce实现PageRank算法 3](#_Toc1916)

[1.4实验结果 3](#_Toc9760)

[附录 6](#_Toc9356)

[PageRank算法代码 6](#_Toc19950)

# 1.试验任务一 PageRank算法

## 1.1 实验环境

ubuntu 18.04

hadoop-3.2.1

## 1.2 实验目的

(1)了解PageRank算法并实现；

(2)深入理解Hadoop系统并使用MapReduce实现PageRank算法，巩固课堂知识。

## 1.3 设计思路

### 1.3.1 PageRank算法

PageRank算法主要用于评测网页的重要程度，PageRank主要通过网页中的链接来“投票”决定网页的重要程度。一个页面的“得票数”由所有链向它的页面的重要性来决定，到一个页面的超链接相当于对该页投一票。一个页面的PageRank是由所有链向它的页面（“链入页面”）的重要性经过递归算法得到的。一个有较多链入的页面会有较高的等级，相反如果一个页面没有任何链入页面，那么它没有等级。

假设一个由4个页面组成的小团体：A，B，C和D。如果所有页面都链向A，那么A的PR（PageRank）值将是B，C及D的Pagerank总和。继续假设B也有链接到C，并且D也有链接到包括A的3个页面。一个页面不能投票2次。所以B给每个页面半票。以同样的逻辑，D投出的票只有三分之一算到了A的PageRank上。



换句话说，根据链出总数平分一个页面的PR值。



最后，所有这些被换算为一个百分比再乘上一个系数。由于“没有向外链接的页面”传递出去的PageRank会是0，所以，Google通过数学系统给了每个页面一个最小值：



由于每次计算后所有页面的PR值会变动，所以计算得到的PR值并不精确，所以需要不断循环计算新的PR值，直至所有页面的PR值基本不再浮动，此时所有PR值组成的方程组都满足。

### 1.3.2 MapReduce实现PageRank算法

综上，PageRank算法可通过两步MapReduce实现。

第一步，统计每个页面的出度。根据实验提供材料，输入的键值对为<From nodeld，To nodeld>，所以输出键值对<From nodeld，1>，表示From nodeld出度加1，在Reduce阶段，将同一key值的键值对合并，即可得到key值对应页面的出度。

第二步，循环计算PR值。每个页面规定一个初始PR值，保存入文件中，然后根据1.3.1中给出的公式计算，每次计算后更新文件中的PR值，直到循环足够多的次数或两次的PR值差别在很小的范围内。这一步的输入文件和第一步相同，所以在Map阶段可以计算value值对应的页面的“投票数”，即PR(k)/L(k)，然后输出对应键值对<key,PR(k)>；在Reduce阶段，将相同key的键值对的value相加后乘以系数即可得到新的PR值，接着只需要存入文件等待下一轮计算即可。

## 1.4实验结果

在eclipse中运行程序，控制台输出结果如图1.1。

接着查看文件中rank值输出，本次试验中rank值循环计算了10次，对比第9次和第10次结果，如图1.2和图1.3，两次结果相近，可以使用。

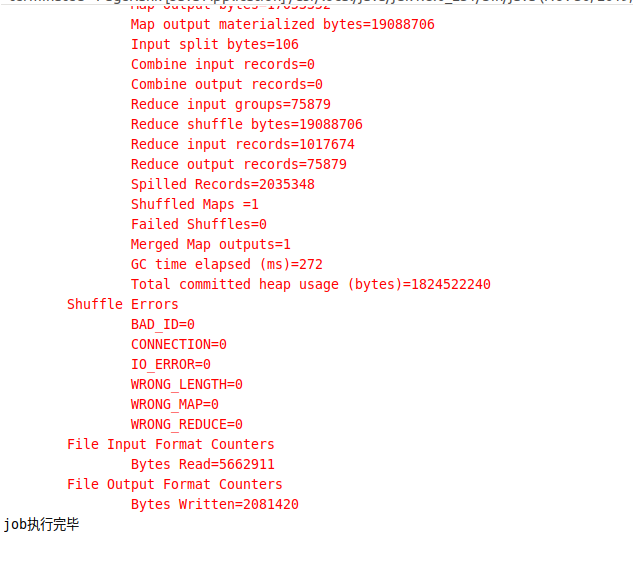


图1.1 控制台结果输出

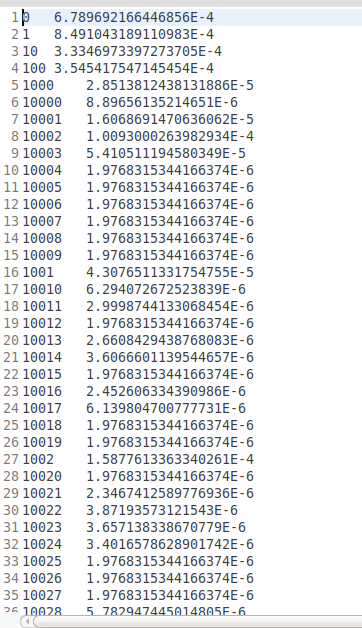


图1.2 rank计算阶段第9次输出结果

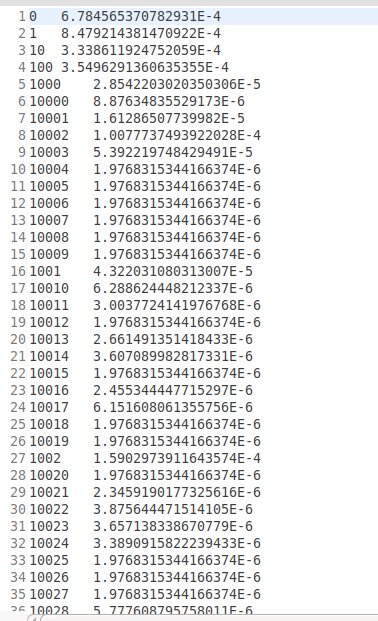


图1.3 rank计算阶段第9次输出结果

# 附录

## PageRank算法代码

package pagerank;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.FileSystem;

import org.apache.hadoop.fs.Path;

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.KeyValueTextInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import java.io.IOException;

import java.util.HashMap;

import java.util.Map;

public class PageRank {

private static String dataPath = "hdfs://master:9000/input/pagerankData.txt";

private static String acountLinePath = "hdfs://master:9000/output/AcountLine.txt";

private static String prPath = "hdfs://master:9000/output/Pr.txt";

// 统计所有链接的出链

private static Map<String,Integer> allOutLine = new HashMap<>();

// 统计所有链接的现有pagerank

private static Map<String,Double> allPageRank = new HashMap<>();

// 统计所有链接计算后的pagerank

private static Map<String ,Double> allNextPageRank = new HashMap<>();

// 计算rank值时的系数

private static double d = 0.85;

// 计算rank值时的循环次数

private static int cycle\_index = 10;

public static void main(String[] args) {

Configuration configuration = new Configuration();

configuration.set("fs.defaultFS", "hdfs://master:9000");

//第一个MapReduce为了统计出每个页面的出链数

//第二个MapReduce为了循环计算每个页面的PageRank值

//run1(configuration);

if (run1(configuration)){

run2(configuration);

}

}

/\*

输入数据格式：

FromNodeId ToNodeId

FromNodeId ToNodeId

\*/

/\*

输出数据格式：<key，1>

\*/

static class AcountOutMapper extends Mapper<Text,Text,Text,Text>{

// 计算每个节点的出链数

private final static String one = "1";

private final static String zero = "0";

@Override

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

//统计出链，<outLines[0]，1>

//输入文件格式<FromNodeId,ToNodeId>

context.write(key,new Text(one));

context.write(new Text(value.toString()),new Text(zero));

}

}

static class AcountOutReducer extends Reducer<Text,Text,Text,Text>{

@Override

protected void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException {

int sum = 0;

for (Text text : values){

sum += Integer.parseInt(text.toString());

}

// <key,出链数>

context.write(key,new Text(sum+""));

allOutLine.put(key.toString(),sum);

allPageRank.put(key.toString(),1.0);

}

}

public static boolean run1(Configuration configuration){

try {

Job job = Job.getInstance(configuration);

FileSystem hdfs = FileSystem.get(configuration);

job.setJobName("acountline");

job.setJarByClass(PageRank.class);

job.setMapperClass(AcountOutMapper.class);

job.setReducerClass(AcountOutReducer.class);

job.setMapOutputKeyClass(Text.class);

job.setMapOutputValueClass(Text.class);

job.setInputFormatClass(KeyValueTextInputFormat.class);

FileInputFormat.addInputPath(job,new Path(dataPath));

Path outPath = new Path(acountLinePath);

if (hdfs.exists(outPath)){

hdfs.delete(outPath,true);

}

FileOutputFormat.setOutputPath(job,outPath);

return job.waitForCompletion(true);

} catch (Exception e) {

e.printStackTrace();

}

return false;

}

static class PageRankMapper extends Mapper<Text,Text,Text,Text>{

@Override

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

String myUrl = key.toString();

int outLine = allOutLine.get(myUrl); // 计算出度

double nowPageRank = allPageRank.get(myUrl);

double newPageRank = nowPageRank/outLine;

context.write(key,new Text("0"));

context.write(new Text(value.toString()),new Text(""+newPageRank));

}

}

static class PageRankReducer extends Reducer<Text,Text,Text,Text>{

@Override

protected void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException {

// 后半段求和公式的和 （PR(1)/L(1)…………PR(i)/L(i)

double sum = 0.0;

for (Text text : values){

double pr = Double.valueOf(text.toString());

sum += pr;

}

// 算出该页面本次的PR结果

double nowPr = (1-d)/allPageRank.size()+d\*sum;

allNextPageRank.put(key.toString(),nowPr);

context.write(key,new Text(""+nowPr));

}

}

public static void run2(Configuration configuration){

for (String key : allPageRank.keySet()){

allPageRank.put(key,1.0/allPageRank.size());

allNextPageRank.put(key,1.0/allPageRank.size());

}

for(int count=0;count<cycle\_index;count++)

{

try{

configuration.setInt("count",count+1);

Job job = Job.getInstance(configuration);

FileSystem fileSystem = FileSystem.get(configuration);

job.setJobName("pagerank");

job.setJarByClass(PageRank.class);

job.setJobName("Pr"+count);

job.setMapperClass(PageRankMapper.class);

job.setReducerClass(PageRankReducer.class);

job.setMapOutputKeyClass(Text.class);

job.setMapOutputValueClass(Text.class);

job.setInputFormatClass(KeyValueTextInputFormat.class);

Path intPath = new Path(dataPath);

FileInputFormat.addInputPath(job,intPath);

Path outPath = new Path(prPath+count);

if (fileSystem.exists(outPath)){

fileSystem.delete(outPath,true);

}

FileOutputFormat.setOutputPath(job,outPath);

boolean f = job.waitForCompletion(true);

if (f){

System.out.println("job执行完毕");

// 提取本轮所有页面的PR值和上一轮作比较，

for (String key : allPageRank.keySet()){

allPageRank.put(key,allNextPageRank.get(key));

}

}

}catch(Exception e) {

e.printStackTrace();

}

}

}

}