# 《网络大数据管理理论和应用》大作业报告

姓名：..

学号：……….

日期：…..

题目：分别在MapReduce和Spark平台上实现微博用户重要度排序。

## 一.实验环境部署：

本次作业的环境是在我实验室自己的单机上面配置的，宿主机为ubuntu14.04（ip地址为：162.105.67.32），然后上面安装了虚拟机也是ubuntu14.04（ip为：162.105.67.31），通过桥接的方式链接的集群。

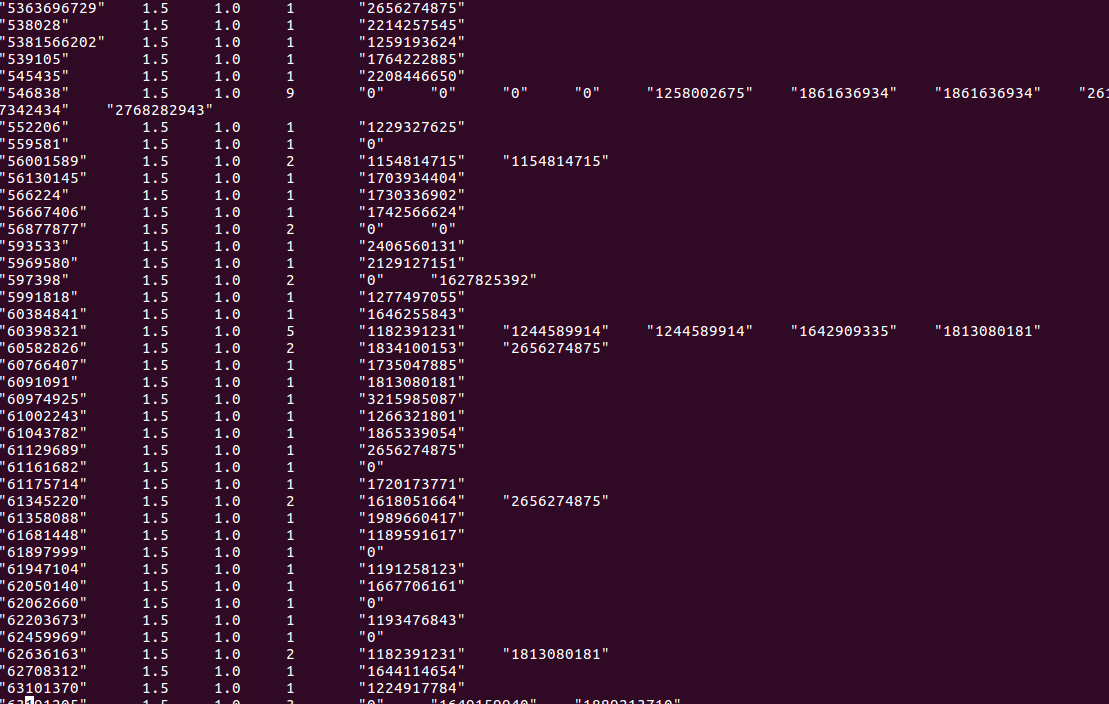
①在用hadoop上面的mapreduce实现pagerank的时候，我是在eclipse上面用java来实现的。

②在spark上实现pagerank的时候，我是通过python语言编写脚本，然后用spark-submit来提交运行的。

## 二.实验方法介绍

**①mapreduce实现**：

数据预处理：为了方便，先对数据做一下预处理，把数据统一成下面图A所示的格式：



图A-预处理后的数据示例

然后对每一行的输入抽象为Node节点的结构，示意图如图B所示：

老的pr值p

从属节点个数

新的pr值

从属节点

主节点id

图B-节点抽象图

初始的时候每个节点的newid值是1.0，oldid值一开始没用统一记为1.5。

这样的mapper每次读入一行数据的时候就把数据变成一个节点，这个Node节点的属性和主要的方法如下所示：

**public** **class** Node

{

String id;//当前节点

List destNodes //被指向的节点的列表

**public** **double** oldPR;// 前一轮迭代的值

**public** **double** newPR;// 新计算出来的值

**int** numDest;//指向的节点的个数

**static** String toTextWithoutID(Node node)//保留后四个部分

**static** Node InstanceFromString(String nodeTuple) //把一行字符串变为实例化节点

}//节点大概结构

为了防止某些节点的pr值太小，每次迭代的时候的pr值公式采用下面所示：

这样会pr值最小为1-PageRank.*d。因为总的网页数量很大，不这样处理的话pr值最后变为非常的小，没有实际意义。*

newPR = (1-PageRank.*d*) + PageRank.*d*\*inPR ;

（而不是采用.newPR = (1-PageRank.*d*)/PageRank.*numberofNodes* + PageRank.*d*\*inPR ;）

在Node结构中保留前一轮的pr值是为了方便判断什么时候停止迭代，方便计算每个节点在两次迭代中的误差。

**Mapreduce程序大体流程**：

do

{

long begin = System.currentTimeMillis();//第i轮起始时间

Job pageRankJob //在hadoop上面，计算pr值

Job chkjob //在hadoop上面计算当前轮与前一轮的均方误差

iterator ++; //迭代次数加1

long end = System.currentTimeMillis() - begin;//第i轮结束时间

System.out.println("第"+iterator+"轮耗时：" + end + "毫秒");//输出迭代时间

}while(!PageRank.canStopIteration(chkpath,PageRank.threshold,PageRank.maxiter,iterator));

说明：每一次while循环执行两轮mapreduce。通过canStopIteration函数判断是否结束迭代。pagerankJob是真正的计算每个节点当前的pr值，

chkJob是用来计算经过一轮迭代循环之后所有节点的均方差，当他小于我们预先设定好的值之后，或者超过最大迭代次数，就停止程序。在每一轮，迭代完成之后都输出当前一轮的时间，以毫秒为单位。

**代码较长，放在附件里面。**

**②spark：**

Spark计算pagerank我用的是python语言写的代码，为了方便使用Spark SQL

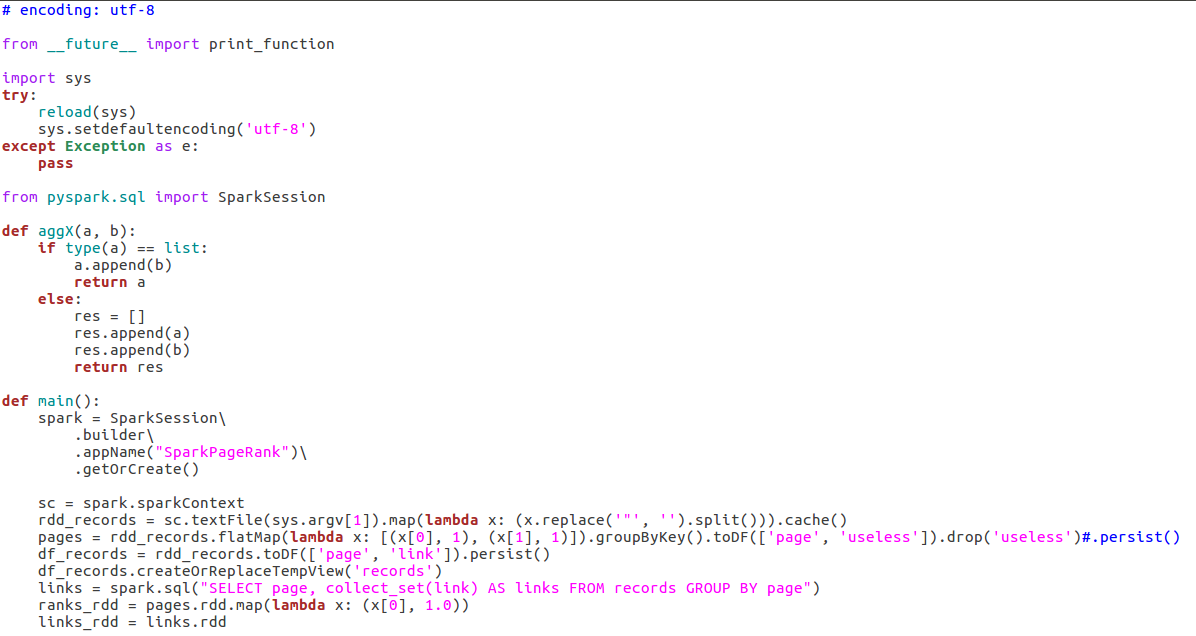
的一些功能和sparkContext,就引入了SparkSession，但是spark需要2.0版本以上才支持。

在spark目录下面直接提交python代码就可以了，

/bin/spark-submit PageRank.py hdfs://Master:9000/spark/input/\* hdfs://Master:9000/spark/output

代码图C中Argv【1】 Argv【2】分别指输入文件路径和输出文件路径

代码较短截图如下所示：





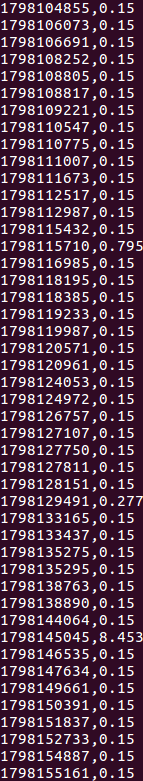
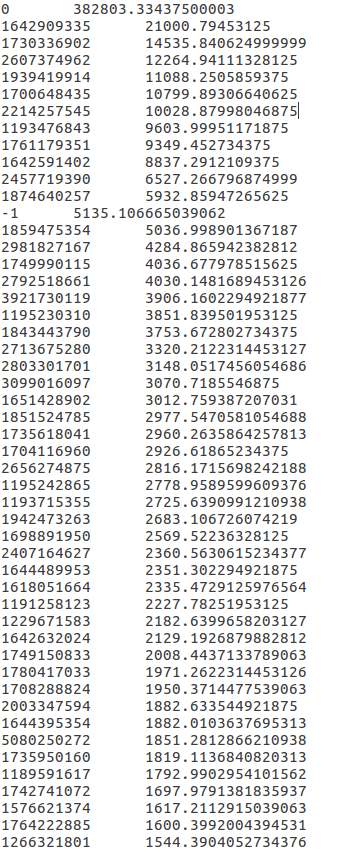
图C—spark代码

## 三.实验结果统计

**①mapreduce的结果**：

通过简单的统计发现0自己指向自己就有9827接近一万条记录，所以会产生0的pr值非常的高，为了简单，也没有单独做处理。

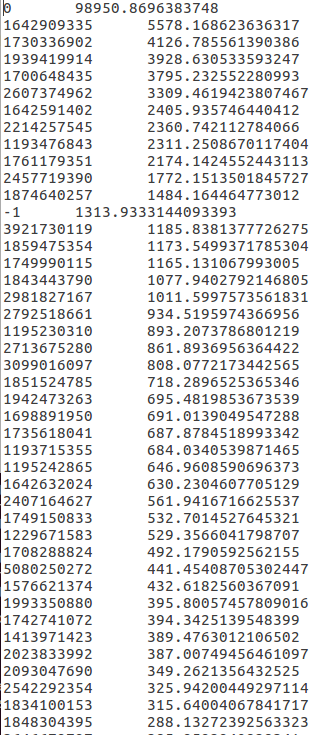
左边图D未排序的输出，右边图E是单独排序之后的汇总结果



图D-mr未排序 图E-mr排序后排名靠前节点

**②spark的结果：**

如下面图F所示，我们发现节点0都排在了第一而且节点0总的接近一万个点指向它，每个节点初始的pr=1，最终基本全部都分流到了0节点了接近10万：



图F-spark计算排名靠前的节点

四．对两个平台上实现方法的对比

（**包括：程序的对比、数据加载时间、第一轮迭代用时、每轮迭代的平均时间、总的算法执行时间等**）

**(1)程序的对比**：

①Mapreduce:程序较长，需要很多包的导入，和引用很多函数

②spark：不论是用scala还是python，所需要的代码都相对较短，程序比较只管

**(2)两个平台各种迭代用时的对比**：

①Mapreduce:

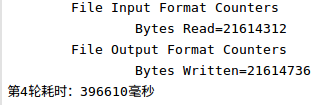
预处理耗时：13233毫秒

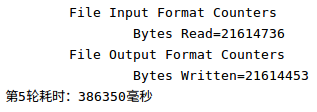
第1轮耗时：383724毫秒

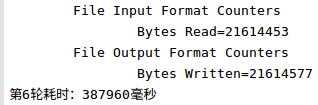
第2轮耗时：399233毫秒，

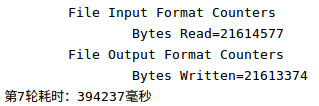
第3轮耗时：408315毫秒

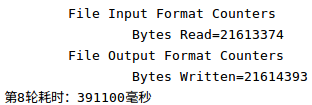
第4轮耗时：396610毫秒

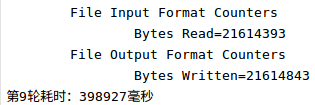


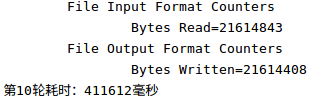












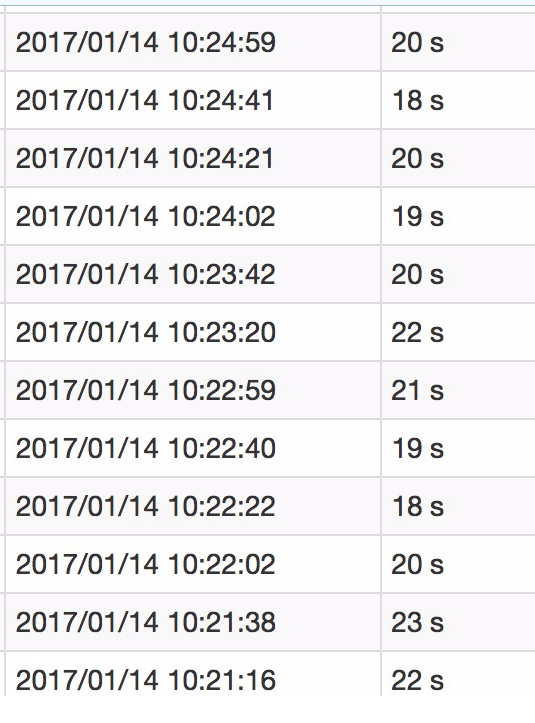
**每轮迭代的平均时间**：395423毫秒，差不多6.6分钟

**总的算法执行时间等**：4155秒==69分钟==差不多一个小时多一点

**②Spark耗时：**

Spark的数据加载时间在1-2秒就可以完成了。

每轮迭代的时间统计如图G所示：



图G-spark每轮迭代耗时

**每轮迭代的平均时间**：20秒，

**总的算法执行时间等**：241秒==大概4分钟就全部执行完了

## 五．你在这门课上学到了什么，以及对这门课的建议

**(1)实验环境：**

在实验室的自己单机电脑上，运行在ubuntu系统上面，

电脑配置：

内存：4GB

处理器：Intel® Core™ i5-4570 CPU @ 3.20GHz × 4

**(2)实验内容与完成情况：**

在基于Hadoop的平台上分别运行了pagerank的mapreduce方法和spark的方法，运行的数据集是完整的数据集page\_rank\_data.txt。

结果发现spark是hadoop自带的mapreduce的20-30倍的速度，所以在以后的数据处理中如果是任务对时间要求比较严格，那么我们最好选用spark，这样可以充分减少磁盘io加快处理速度。

**(3)学习心得：**

首先就是从对大数据完全从感性的认识到了一个比较真实的了解，学习了大数据大发展历史，起源，以及未来的趋势。在课堂老师介绍了主流的一些大数据平台和工具，也通过同学的课堂报告了解了一些比较前沿或者专业的一些数据处理工具，这都对以后从事专门的大数据处理工作有一个很重要的铺垫作用，同时也知道了大数据的一些常见的应用，和它能够处理数据的规模。最后还在课堂以及实验中了解了不同的平台处理数据的速度的差异，这些都让我们对大数据处理有了一个总体的认识。

**(4)课堂建议：**

感觉每次课的知识量有些大，如果不是有一定的基础或者没有提前准备预习的话，上课期间会比较吃力。希望一学期开学的时候提前把课件上传，不要每次上完课之后再上传课件，这样的话，先自己看一遍在听课，效率会高一些。