

1 PageRank介绍

- 1.1 算法来源
- 1.2 算法原理
- 1.3 终止点和陷阱

2 Hadoop 安装以及配置

- 2.1 安装SSH、配置SSH无密码登录
- 2.2 安装 Java 环境
- 2.3 安装Hadoop (国内源)
- 2.4 配置Hadoop
 - 2.4.1 编辑环境文件 .bashrc
 - 2.4.2 编辑hadoop-env.sh文件
 - 2.4.3 查看java和hadoop环境配置
 - 2.4.4 配置 core-site.xml文件
 - 2.4.5 配置hdfs-site.xml文件
 - 2.4.6 配置yarn-site.xml文件
- 2.5 启动Hadoop
- 2.6 在外部浏览器输入 master 的 IP 地址和 50070 端口的「Datanodes」查看 hdfs 上的节点

3 MapReduce实现

- 3.1 NodeCoverter.java
- 3.2 NodeCoverterMapper.java
- 3.3 NodeCoverterReduce.java
- 3.4 PageRank.java
- 3.5 PageRankMapper
- 3.6 PageRankReduce.java

4 PageRank实践

- 4.1 导入数据集
- 4.2 编译NodeCoverter
- 4.3 编译PageRank

5 实验心得

1 PageRank介绍

1.1 算法来源

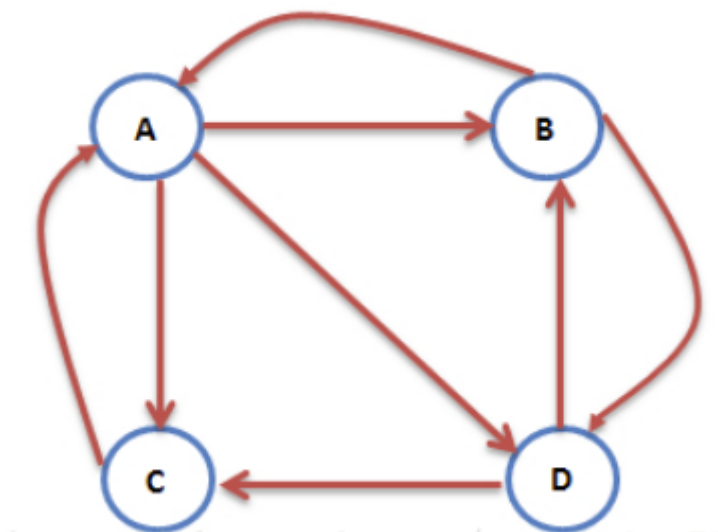
最早的搜索引擎采用的是**分类目录**的方法，即通过人工进行网页分类并整理出高质量的网站。后来网页越来越多，人工分类已经不现实了。搜索引擎进入了**文本检索**的时代，即计算用户查询关键词与网页内容的相关程度来返回搜索结果。这种方法突破了数量的限制，但是搜索结果不是很好。因为总有某些网页来回地倒腾某些关键词使自己的搜索排名靠前。

于是谷歌的两位创始人，当时还是美国斯坦福大学 (Stanford University) 研究生的佩奇 (Larry Page) 和布林 (Sergey Brin) 开始了对网页排序问题的研究。他们的借鉴了学术界评判学术论文重要性的通用方法，那就是看论文的引用次数。由此想到网页的重要性也可以根据这种方法来评价。于是PageRank的核心思想就诞生了：

- 如果一个网页被很多其他网页链接到的话说明这个网页比较重要，也就是PageRank值会相对较高
- 如果一个PageRank值很高的网页链接到一个其他的网页，那么被链接到的网页的PageRank值会相应地因此而提高

1.2 算法原理

如果说最终的目的是要为每个网页赋予一个得分的话，那么在初始化阶段，所有网页的得分应该是相等的，假设现在一共有NN个网页，那么初始阶段用户访问每个网页的概率就是 $1/N$ ，比如下图中的初始概率向量为 $(1/4, 1/4, 1/4, 1/4)$ ，而因为每个网页都可能存在着到其他任何网页的链接，所以，我们可以将这些链接以平均的概率表示成一个概率转移向量。比如下面这张图，网页A中有3个链接，分别指向B, C, D，则A的概率转移向量为 $(0, 1/3, 1/3, 1/3)$ ，向量的维度依次对应的是网页A指向A, B, C, D四个网页的概率。



显然，每个网页都有这样的概率转移向量，我们把这些向量转置后（按列放置）合起来就是随机过程中经典的概率转移矩阵了，如下：

$$\begin{pmatrix} 0 & 1/2 & 1 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{pmatrix}$$

不难理解，这种概率转移矩阵的一个显著的特征是每个元素都 ≥ 0 ，且列和为1，含义表示从当前网页跳转到其他网页的概率和为1。我们记这个矩阵为 T ， $T[i][j]$ 的含义是既可以表示由网页 i 跳转至网页 j 的概率，又可以表示由网页 j 跳转至网页 i 的概率。

根据这个矩阵，我们能够计算出经过一次网页跳转之后，用户访问到每个网页的概率分布，比如上图中，经过一次跳转之后，用户访问到各个网页的概率可以如下计算：

$$V_1 = T \cdot V_0 = \begin{pmatrix} 0 & 1/2 & 1 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{pmatrix} = \begin{pmatrix} 9/24 \\ 5/24 \\ 5/24 \\ 5/24 \end{pmatrix}$$

其中， T 为概率转移矩阵。同理，我们可以按照这个方法持续计算到用户第 n 次跳转之后的概率分布，公式如下

$$V_n = T \cdot V_{n-1} = T^n \cdot V_0$$

佩奇和布林发现，当 $n \rightarrow +\infty$ ，且概率转移矩阵 T 满足以下3个条件时， $\lim_{n \rightarrow +\infty} V_n$ 最终收敛，保持在一个稳定值附近。

- T 为随机矩阵。即所有 $T[i][j] \geq 0$ ，且的所有列向量的元素加和为1， $\sum_{i=1}^n T[i][j] = 1$
- T 是不可约的。所谓不可约是说 T 所对应的图示**强连通的**，即图中任何一个节点都可以达到其他任何一个节点，它要求概率转移矩阵不存在某些特殊情况，比如某个列向量都为0，即有一个网页没有链接到任何其他网页，我们把这类情况又称为终止点；或者是在概率转移矩阵的主对角线上，存在有一个元素为1的情况，即这个网页链接只链接它自己，我们把这类情况又称为陷阱。这两类特殊的情形在后面会详细说。
- T 是非周期的。所谓周期性，体现在Markov链的周期性上。即若 A 是周期性的，那么这个Markov链的状态就是周期性变化的。因为 A 是素矩阵（素矩阵指自身的某个次幂为正矩阵的矩阵），所以 A 是非周期的

比如上面这个例子中， $\lim_{n \rightarrow +\infty} V_n = \lim_{n \rightarrow +\infty} T^n \cdot V_0 = (3/9, 2/9, 2/9, 2/9)^T$ 这表明，经过足够多次的网页跳转，用户停留在网页A的概率要比停留在B, C, D的概率高，而后者基本是等概率的。经过这样的计算得到的每个网页的概率值也叫PR值，是评估网页质量的依据。也就是说，在我们使用搜索引擎时，在保持网页与查询一定相关度的基础上，PR值可以提供非常不错的排序依据。

在一般情况下，一个网页的PR值计算如下：

$$PR(p_i) = \alpha \sum_{p_j \in M_{pi}} \frac{PR(p_j)}{L(p_j)} + \frac{(1 - \alpha)}{N}$$

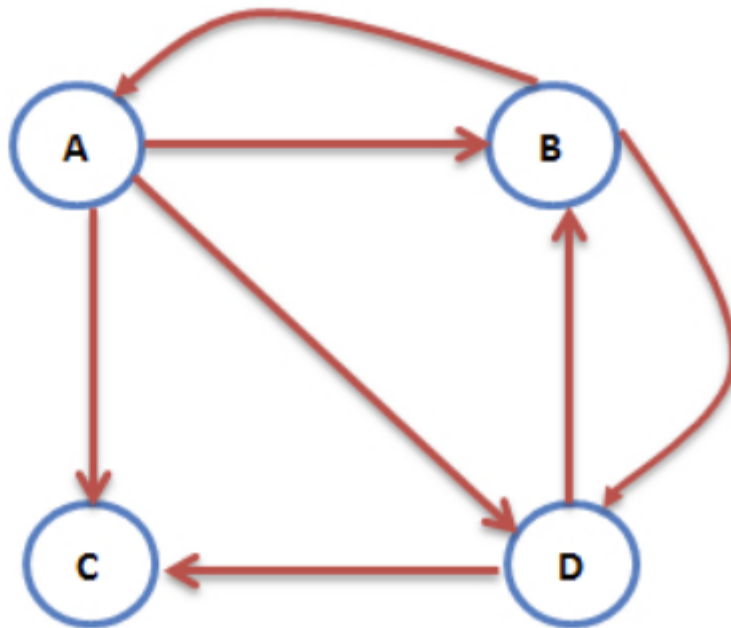
其中 M_{pi} 是所有对 p_i 网页有出链的网页集合， $L(p_j)$ 是网页 p_j 的出链数目， N 是网页总数， α 一般取0.85。

根据上面的公式，我们可以计算每个网页的PR值，在不断迭代趋于平稳的时候，即为最终结果。具体怎样算是趋于平稳，我们在下面的PR值计算方法部分再做解释。

1.3 终止点和陷阱

终止点

终止点指的是没有任何出链的网页。按照上面这种随机游走模型，用户访问到这个网页后，因为找不到链接，就会“四顾茫然”，不知下一步该怎么办了。比如下图中，网页C没有出链。这种情况下，概率转移矩阵在终止点对应的那一列，所有元素都是0，代入上面的PR值计算公式，会发现经过无穷多次跳转后，所有网页的PR值都是0：



上图对应的概率转移矩阵为：

$$T = \begin{pmatrix} 0 & 1/2 & 0 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{pmatrix}$$

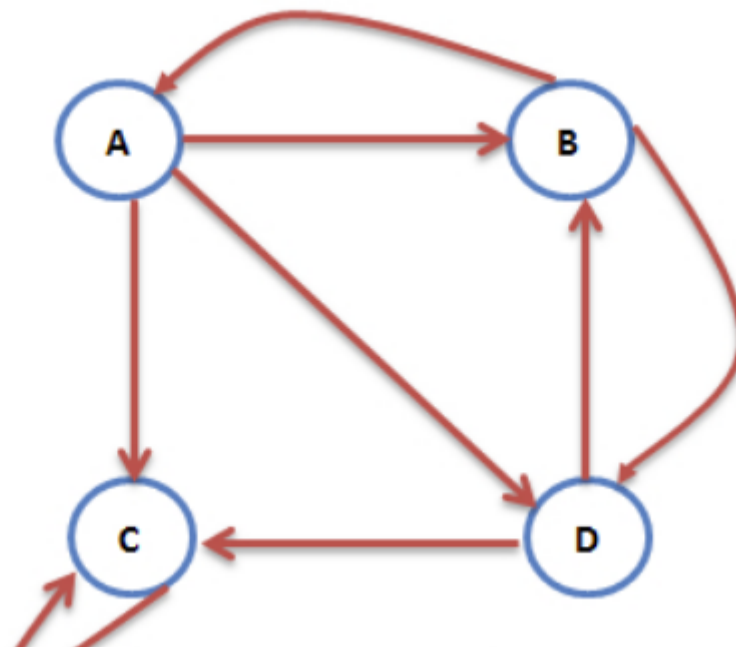
可见，对应C的第3列全为0，迭代计算PR值，最后的结果都是0：

$$\begin{pmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{pmatrix} \rightarrow \begin{pmatrix} 3/34 \\ 5/24 \\ 5/24 \\ 5/24 \end{pmatrix} \rightarrow \begin{pmatrix} 5/48 \\ 7/48 \\ 7/48 \\ 7/48 \end{pmatrix} \rightarrow \dots \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

那这样的计算结果当然是毫无意义的。

陷阱

陷阱指的是只有指向自身链接的网页。这种情况下，所有“随机游走”的用户到了这个网页后，就如同进了黑洞一般，一直在里面“打转”，出不来了。比如下图中，网页C只有一条出链，并且还是指向它自己的，这就把C变成了陷阱。



有陷阱的网络对应的概率转移矩阵中，主对角线上存在至少一个为1的元素（主对角线存在n个1，就代表整个网络中有n个陷阱）。

$$T = \begin{pmatrix} 0 & 1/2 & 0 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 1 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{pmatrix}$$

这种情况下，陷阱的PR值为1，而其他正常网页的PR值为0。迭代计算的结果大致如下：

$$\begin{pmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{pmatrix} \rightarrow \begin{pmatrix} 3/24 \\ 5/24 \\ 11/24 \\ 5/24 \end{pmatrix} \rightarrow \begin{pmatrix} 5/48 \\ 7/48 \\ 29/48 \\ 7/48 \end{pmatrix} \rightarrow \dots \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

当然，这种结果也是毫无参考意义的。

解决思路

当一个人遇到终止点或者陷阱的话，他不会不止所错，也不会无休止地自己打转，他会通过浏览器的地址栏输入新的地址，以逃离这个网页。也就是说，用户从一个网页转至另一个网页的过程中，会以一定的概率不点击当前网页中的链接，而是访问一个自己重新输入的新地址。我们可以依照这个原理修正之前的概率转移公式，如下：

$$V_n = \alpha TV_{n-1} + (1 - \alpha)V_0$$

其中， α 为用户继续点击当前网页中的链接的概率， $(1-\alpha)$ 为用户通过地址栏“逃离”的概率。关于这里的 α ，其实很讲究，首先 α 不能太大，因为 α 的大小与算法的收敛速度呈反比， α 太大会导致算法收敛慢而影响性能；其次， α 也不能太小，因为PageRank的精华就在于上面的公式中前一部分——由概率转移矩阵的多次迭代计算得到PR值。所以，最终两位博士将 α 值定为0.85。

我们来看经过这样处理之后，上面的终止点和陷阱问题能否得到解决：

当存在终止点时，迭代的结果为：

$$\begin{pmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{pmatrix} \rightarrow \begin{pmatrix} 0.14 \\ 0.21 \\ 0.21 \\ 0.21 \end{pmatrix} \rightarrow \begin{pmatrix} 0.1 \\ 0.1 \\ 0.1 \\ 0.1 \end{pmatrix}$$

当存在陷阱时，迭代的结果为：

$$\begin{pmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{pmatrix} \rightarrow \begin{pmatrix} 0.14 \\ 0.21 \\ 0.42 \\ 0.21 \end{pmatrix} \rightarrow \dots \begin{pmatrix} 0.1 \\ 0.1 \\ 0.7 \\ 0.1 \end{pmatrix}$$

这样就解决了终止点与陷阱的问题。

2 Hadoop 安装以及配置

这里的步骤有点类似于**实验 1**的操作（所以有些用户不同的话其实用了前面实验的图，但是这些不影响实验的结果，因为当时做的时候这不算是重点就忽略掉了没有截图。其中因为更换过了系统，所以重新安装一遍吧）

2.1 安装SSH、配置SSH无密码登录

SSH 协议可以为节点上的账户创建唯一的公私钥，然后利用这些公私钥实现无密码登录，从而让 Hadoop 直接绕开传统的账号密码登录过程，直接用公私钥访问节点。

```
# hadoop不管是集群、单机版都需要用到 SSH 登陆，系统默认已安装了 SSH client，此外还需要安装 SSH server:
$ sudo apt-get install openssh-server
# 安装后，尝试登陆本机，提示时输入yes,然后按提示输入密码
$ ssh localhost
# 在hadoop中，我们需要配置成SSH无密码登陆，首先退出输入exit，退出刚才的 ssh，回到我们原先的终端窗口，然后利用 ssh-keygen 生成密钥，并将密钥加入到授权中：
$ exit # 退出刚才的 ssh localhost
$ cd ~/.ssh/ # 若没有该目录，请先执行一次ssh localhost
$ ssh-keygen -t rsa # 会有提示，都按回车就可以
$ cat ./id_rsa.pub >> ./authorized_keys # 加入授权
# 此时再用 ssh localhost 命令，无需输入密码就可以直接登陆了
```

```
hadoop17341018@ubuntu:/home/eevhile$ sudo apt-get install openssh-server
正在读取软件包列表... 完成
正在分析软件包的依赖关系树
正在读取状态信息... 完成
openssh-server 已经是最新版 (1:8.0p1-6build1)。
下列软件包是自动安装的并且现在不需要了:
  libgdbm5 libmysqlclient20 libncurses5 libperl5.26 libpython3.6 libpython3.6-minimal libpython3.6-stdlib multiarch-support
使用'sudo apt autoremove'来卸载它(它们)。
升级了 0 个软件包，新安装了 0 个软件包，要卸载 0 个软件包，有 13 个软件包未被升级。
hadoop17341018@ubuntu:/home/eevhile$ ssh localhost
hadoop17341018@localhost's password:
Welcome to Ubuntu 19.10 (GNU/Linux 5.3.0-18-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

12 updates can be installed immediately.
0 of these updates are security updates.
To see these additional updates run: apt list --upgradable

Last login: Wed Mar  4 06:41:37 2020 from 127.0.0.1
hadoop17341018@ubuntu:~$
```

1、安装openssh-server

2、连接localhost

```
hadoop17341018@ubuntu:~$ exit
logout
Connection to localhost closed.
hadoop17341018@ubuntu:~/hadoop17341018$ cd ~/.ssh/
hadoop17341018@ubuntu:~/.ssh$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/hadoop17341018/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/hadoop17341018/.ssh/id_rsa.
Your public key has been saved in /home/hadoop17341018/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:8LXaTL8Xp7rkJs7MOYDNaIrNKsb162jfrACTGzCtCgQ hadoop17341018@ubuntu
The key's randomart image is:
+---[RSA 3072]-----+
|Eooo.                |
|..+o                 |
|.. .o.. .            |
|.. . .o.. .          |
|o  *S o              |
|.. . = + = . . . .   |
|..+.o...o o +        |
|..o..=.o +o+..o      |
|..ooo+.. .*+=+       |
+---[SHA256]-----+
hadoop17341018@ubuntu:~/.ssh$ cat ./id_rsa.pub >> ./authorized_keys
hadoop17341018@ubuntu:~/.ssh$

hadoop17341018@ubuntu:~/.ssh$ ssh localhost
Welcome to Ubuntu 19.10 (GNU/Linux 5.3.0-18-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

12 updates can be installed immediately.
0 of these updates are security updates.
To see these additional updates run: apt list --upgradable

Last login: Wed Mar  4 06:48:17 2020 from 172.17.0.1
hadoop17341018@ubuntu:~$ exit
logout
Connection to localhost closed.
hadoop17341018@ubuntu:~/.ssh$
```

- 1、退出
- 2、切换目录
- 3、根据提示回车
- 4、完成

- 1、不需要密码
- 2、退出

2.2 安装 Java 环境

```
# hadoop的运行依赖java环境，而且经验得知，如果是最新版Java的话总是会出现奇奇怪怪的问题，所以
采用的还是1.8经典版
$ wget https://repo.huaweicloud.com/java/jdk/8u152-b16/jdk-8u152-linux-
x64.tar.gz
# Untar
$ sudo tar -xzf jdk-8u152-linux-x64.tar.gz -C /usr/local
# Rename
$ sudo mv /usr/local/jdk1.8.0_152 /usr/local/java
```



```

eewhile@ubuntu:~$ wget https://repo.huaweicloud.com/java/jdk/8u152-b16/jdk-8u152-linux-x64.tar.gz
--2020-06-22 02:14:01-- https://repo.huaweicloud.com/java/jdk/8u152-b16/jdk-8u152-linux-x64.tar.gz
正在解析主机 repo.huaweicloud.com (repo.huaweicloud.com)... 111.29.28.254, 36.159.58.30
正在连接 repo.huaweicloud.com (repo.huaweicloud.com)[111.29.28.254]:443... 已连接。
已发出 HTTP 请求，正在等待回应... 200 OK
长度: 189784266 (181M) [application/octet-stream]
正在保存至: "jdk-8u152-linux-x64.tar.gz"

jdk-8u152-linux-x64.ta 96%[=====> ] 175.53M 3.72MB/s 剩余 1s ^
jdk-8u152-linux-x64.ta 100%[=====>] 180.99M 3.75MB/s 用时 48s

2020-06-22 02:14:49 (3.77 MB/s) - 已保存 "jdk-8u152-linux-x64.tar.gz" [189784266/189784266])

eewhile@ubuntu:~$
eewhile@ubuntu:~$ sudo tar -xzf jdk-8u152-linux-x64.tar.gz -C /usr/local
[sudo] eewhile 的密码:
jdk1.8.0_152/
jdk1.8.0_152/javafx-src.zip
jdk1.8.0_152/bin/
jdk1.8.0_152/bin/jmc
jdk1.8.0_152/bin/serialver
jdk1.8.0_152/bin/jmc.ini
jdk1.8.0_152/bin/jstack
jdk1.8.0_152/bin/rmiregistry
jdk1.8.0_152/bin/unpack200

eewhile@ubuntu:~$ sudo mv /usr/local/jdk1.8.0_152/ /usr/local/java
eewhile@ubuntu:~$ vim ~/.bashrc
eewhile@ubuntu:~$ source .bashrc
eewhile@ubuntu:~$ java -version
java version "1.8.0_152"
Java(TM) SE Runtime Environment (build 1.8.0_152-b16)
Java HotSpot(TM) 64-Bit Server VM (build 25.152-b16, mixed mode)

```

2.3 安装Hadoop（国内源）

```

# 清华园下载hadoop会比官网的速度快很多
$ sudo wget
https://mirrors.tuna.tsinghua.edu.cn/apache/hadoop/common/stable/hadoop-3.2.1.tar.gz
# 解压
$ sudo tar -xzf hadoop-3.2.1.tar.gz
# 改名为 hadoop 方便后面的操作
$ sudo mv ./hadoop-3.2.1/ ./hadoop
# 加入hadoop组(这里的eewhil就是对应自己的master用户名字)
$ sudo chown -R eewhile ./hadoop

```



```

hadoop17341018@ubuntu:/usr/local$ sudo wget https://mirrors.tuna.tsinghua.edu.cn/apache/hadoop/common/stable/hadoop-3.2.1.tar.gz
--2020-03-04 08:15:23-- https://mirrors.tuna.tsinghua.edu.cn/apache/hadoop/common/stable/hadoop-3.2.1.tar.gz
正在解析主机 mirrors.tuna.tsinghua.edu.cn (mirrors.tuna.tsinghua.edu.cn)... 101.6.8.193, 2402:f000:1:408:8100::1
正在连接 mirrors.tuna.tsinghua.edu.cn (mirrors.tuna.tsinghua.edu.cn)|101.6.8.193|:443... 已连接。
已发出 HTTP 请求，正在等待回应... 200 OK
长度: 359196911 (343M) [application/x-gzip]
正在保存至: "hadoop-3.2.1.tar.gz.1"

hadoop-3.2.1.tar.gz.1 100%[=====] 342.56M 6.40MB/s 用时 59s

2020-03-04 08:16:22 (5.84 MB/s) - 已保存 "hadoop-3.2.1.tar.gz.1" [359196911/359196911]

hadoop17341018@ubuntu:/usr/local$ sudo tar -zxvf hadoop-3.2.1.tar.gz
hadoop-3.2.1/
hadoop-3.2.1/README.txt
hadoop-3.2.1/libexec/
hadoop-3.2.1/libexec/hadoop-config.cmd
hadoop-3.2.1/libexec/hadoop-functions.sh
hadoop-3.2.1/libexec/hadoop-layout.sh.example
hadoop-3.2.1/libexec/shellprofile.d/
hadoop-3.2.1/libexec/shellprofile.d/hadoop-kafka.sh
hadoop-3.2.1/libexec/shellprofile.d/hadoop-allyun.sh
hadoop-3.2.1/libexec/shellprofile.d/hadoop-oozestack.sh

```

1、清华园下载hadoop

2、解压

```

hadoop17341018@ubuntu:/usr/local$ sudo mv ./hadoop-3.2.1/ ./hadoop
hadoop17341018@ubuntu:/usr/local$ sudo chown -R hadoop17341018 ./hadoop
hadoop17341018@ubuntu:/usr/local$

```

这里值得注意的是，因为我的新用户名字是eewhile，所以分组的名字应该是 eewhile 而不是原始的 hadoop17341018 了

2.4 配置Hadoop

2.4.1 编辑环境文件 .bashrc

加入如下内容

```
$ vim ~/.bashrc
```

```

export JAVA_HOME=/usr/local/java
export PATH=$PATH:$JAVA_HOME/bin
export HADOOP_HOME=/usr/local/hadoop
export PATH=$PATH:$HADOOP_HOME/bin
export PATH=$PATH:$HADOOP_HOME/sbin
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME
export YARN_HOME=$HADOOP_HOME
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native
export HADOOP_OPTS="-Djava.library.path=$HADOOP_HOME/lib"
export JAVA_LIBRARY_PATH=$HADOOP_HOME/lib/native:$JAVA_LIBRARY_PATH
export CLASSPATH=$(($HADOOP_HOME/bin/hadoop classpath):$CLASSPATH

```

```

export JAVA_HOME=/usr/local/java
export PATH=$PATH:$JAVA_HOME/bin
export HADOOP_HOME=/usr/local/hadoop
export PATH=$PATH:$HADOOP_HOME/bin
export PATH=$PATH:$HADOOP_HOME/sbin
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME
export YARN_HOME=$HADOOP_HOME
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native
export HADOOP_OPTS="-Djava.library.path=$HADOOP_HOME/lib"
export JAVA_LIBRARY_PATH=$HADOOP_HOME/lib/native:$JAVA_LIBRARY_PATH
export CLASSPATH=($(HADOOP_HOME/bin/hadoop classpath):$CLASSPATH)

# ~/.bashrc: executed by bash(1) for non-login shells.
# see /usr/share/doc/bash/examples/startup-files (in the package bash-doc)
# for examples

# If not running interactively, don't do anything
case $- in
    *i*) ;;
    *) return;;
esac

# don't put duplicate lines or lines starting with space in the history.
# See bash(1) for more options
HISTCONTROL=ignoreboth

# append to the history file, don't overwrite it
shopt -s histappend

# for setting history length see HISTSIZE and HISTFILESIZE in bash(1)
HISTSIZE=1000
HISTFILESIZE=2000

```

编辑完成后，还有重要的 source 一步

```
$ source .bashrc
```

2.4.2 编辑hadoop-env.sh文件

```
$ sudo vim /usr/local/hadoop/etc/hadoop/hadoop-env.sh
```

```

# 在hadoop-env.sh中export JAVA_HOME后面添加以下信息(JAVA_HOME路径改为实际路径):
#export JAVA_HOME=${JAVA_HOME}
export JAVA_HOME=/usr/local/java
export HADOOP_CONF_DIR=${HADOOP_CONF_DIR:-"/etc/hadoop"}
export HADOOP_PID_DIR=/usr/local/hadoop/tmp/pid
export YARN_HOME=$HADOOP_HOME

```

```
# The java implementation to use.
#export JAVA_HOME=${JAVA_HOME}
export JAVA_HOME=/usr/local/java

# The jsvc implementation to use. Jsvc is required to run secure datanodes
# that bind to privileged ports to provide authentication of data transfer
# protocol. Jsvc is not required if SASL is configured for authentication of
# data transfer protocol using non-privileged ports.
#export JSVC_HOME=${JSVC_HOME}

export HADOOP_CONF_DIR=${HADOOP_CONF_DIR:-"/etc/hadoop"}
export HADOOP_PID_DIR=/usr/local/hadoop/tmp/pid
export YARN_HOME=$HADOOP_HOME

# Extra Java CLASSPATH elements. Automatically insert capacity-scheduler.
for f in $HADOOP_HOME/contrib/capacity-scheduler/*.jar; do
    if [ "$HADOOP_CLASSPATH" ]; then
        export HADOOP_CLASSPATH=$HADOOP_CLASSPATH:$f
    else
```

编辑完成后，还有重要的 source 一步

```
$ source /usr/local/hadoop/etc/hadoop/hadoop-env.sh
```

2.4.3 查看java和hadoop环境配置

如图，输入命令查看java和hadoop版本信息，说明环境配置成功

```
$ java -version
$ JAVA_HOME/bin/java -version
$ hadoop version
```

```
eewhile@ubuntu:/usr/local/hadoop/bin$ vim ~/.bashrc
eewhile@ubuntu:/usr/local/hadoop/bin$ sudo vim /etc/profile
eewhile@ubuntu:/usr/local/hadoop/bin$ source .bashrc
bash: .bashrc: 没有那个文件或目录
eewhile@ubuntu:/usr/local/hadoop/bin$ source bashrc
bash: bashrc: 没有那个文件或目录
eewhile@ubuntu:/usr/local/hadoop/bin$ cd
eewhile@ubuntu:~$ source .bashrc
eewhile@ubuntu:~$ source /etc/profile
eewhile@ubuntu:~$ cd /usr/local/hadoop/
eewhile@ubuntu:/usr/local/hadoop$ java -version
java version "1.8.0_152"
Java(TM) SE Runtime Environment (build 1.8.0_152-b16)
Java HotSpot(TM) 64-Bit Server VM (build 25.152-b16, mixed mode)
eewhile@ubuntu:/usr/local/hadoop$ $JAVA_HOME/bin/java -version
java version "1.8.0_152"
Java(TM) SE Runtime Environment (build 1.8.0_152-b16)
Java HotSpot(TM) 64-Bit Server VM (build 25.152-b16, mixed mode)
eewhile@ubuntu:/usr/local/hadoop$ hadoop version
Hadoop 2.7.0
Subversion https://git-wip-us.apache.org/repos/asf/hadoop.git -r d4c8d4d4d203c934e8074b31
289a28724c0842cf
Compiled by jenkins on 2015-04-10T18:40Z
Compiled with protoc 2.5.0
From source with checksum a9e90912c37a35c3195d23951fd18f
This command was run using /usr/local/hadoop/share/hadoop/common/hadoop-common-2.7.0.jar
```

2.4.4 配置 core-site.xml文件

```
$ vim /usr/local/hadoop/etc/hadoop/core-site.xml
```

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!--
  Licensed under the Apache License, Version 2.0 (the "License");
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License. See accompanying LICENSE file.
-->

<!-- Put site-specific property overrides in this file. -->

<configuration>
  <property>
    <name>hadoop.tmp.dir</name>
    <value>/usr/local/hadoop/tmp</value>
  </property>
  <property>
    <name>fs.defaultFS</name>
    <value>hdfs://localhost:9000</value>
  </property>
</configuration>
~
```

如果再fs.defaultFS这里填写不当会出现如下问题

```
eewhile@ubuntu:/usr/local/hadoop$ sbin/start-dfs.sh
Starting namenodes on [localhost]
localhost: starting namenode, logging to /usr/local/hadoop/logs/hadoop-eewhile-namenode-ubuntu.out
localhost: starting datanode, logging to /usr/local/hadoop/logs/hadoop-eewhile-datanode-ubuntu.out
Starting secondary namenodes [0.0.0.0]
The authenticity of host '0.0.0.0 (0.0.0.0)' can't be established.
ECDSA key fingerprint is SHA256:7uZDtdCh4ZIOUlwHKZt4ZjBqmD8SiNR1mMqHBCspL7U.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
0.0.0.0: Warning: Permanently added '0.0.0.0' (ECDSA) to the list of known hosts
.
0.0.0.0: starting secondarynamenode, logging to /usr/local/hadoop/logs/hadoop-eewhile-secondarynamenode-ubuntu.out
```

2.4.5 配置hdfs-site.xml文件

```
$ vim /usr/local/hadoop/etc/hadoop/hdfs-site.xml
```

```

<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!--
  Licensed under the Apache License, Version 2.0 (the "License");
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License. See accompanying LICENSE file.
-->

<!-- Put site-specific property overrides in this file. -->

<configuration>
  <property>
    <name>dfs.replication</name>
    <value>1</value>
  </property>
</configuration>

```

只有一个节点

2.4.6 配置yarn-site.xml文件

```
$ vim /usr/local/hadoop/etc/hadoop/yarn-site.xml
```

```

<?xml version="1.0"?>
<!--
  Licensed under the Apache License, Version 2.0 (the "License");
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License. See accompanying LICENSE file.
-->
<configuration>

<!-- Site specific YARN configuration properties -->
  <property>
    <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle</value>
  </property>
</configuration>

```

2.5 启动Hadoop

如果是第一次启动的话，就不需要先停止所有服务。否则先执行一下操作保证hadoop是未启动状态

```
$ /usr/local/hadoop/sbin/stop-all.sh
```

如果是第一次启动则可以直接初始化，否则，则需要先删除 hadoop/tmp 的内容，然后再创建一个空的 tmp 文件夹。后续操作讲解为什么要这么做

```
$ rm -rf /usr/local/hadoop/tmp
$ mkdir /usr/local/hadoop/tmp
# 因为很可能会丢失掉DataNode节点，是由于多次初始话NameNode造成的，导致两个的版本号不一样，不
# 执行可能会出现以下情况
```

```
ee@ubuntu:~/hadoop$ jps
48388 NodeManager
48232 ResourceManager
47660 NameNode
49197 Jps
48894 SecondaryNameNode
ee@ubuntu:~/hadoop$ sbin/stop-all.sh
This script is deprecated. Instead use stop-dfs.sh and stop-yarn.sh
20/06/25 01:38:42 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Stopping namenodes on [localhost]
localhost: stopping namenode
localhost: no datanode to stop
Stopping secondary namenodes [0.0.0.0]
0.0.0.0: stopping secondarynamenode
20/06/25 01:38:57 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
stopping yarn daemons
stopping resourcemanager
localhost: stopping nodemanager
no proxyserver to stop
```

如果删除的话可以解决缺失问题

接下来就是执行初始化NameNode的问题

```
# 初始化有很多种方式，当然一遍初始化就行了，如果多次初始化会出现提示是否再初始化的问题(Y/N)
$ /usr/local/hadoop/bin/hdfs namenode -format
```

成功配置好，启动输入

```
$ /usr/local/hadoop/sbin/start-all.sh
# 关键看NodeManager,NameNode,DataNode,ResourceManager
$ jps
# 如果没开启hadoop就是只有一个Jps
```

```
ee@ubuntu:~/hadoop$ sbin/start-all.sh
This script is deprecated. Instead use start-dfs.sh and start-yarn.sh
20/06/25 01:49:17 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Starting namenodes on [localhost]
localhost: starting namenode, logging to /usr/local/hadoop/logs/hadoop-ee@ubuntu.out
localhost: starting datanode, logging to /usr/local/hadoop/logs/hadoop-ee@ubuntu.out
Starting secondary namenodes [0.0.0.0]
0.0.0.0: starting secondarynamenode, logging to /usr/local/hadoop/logs/hadoop-ee@ubuntu.out
20/06/25 01:49:33 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
starting yarn daemons
starting resourcemanager, logging to /usr/local/hadoop/logs/yarn-ee@ubuntu.out
localhost: starting nodemanager, logging to /usr/local/hadoop/logs/yarn-ee@ubuntu.out
ee@ubuntu:~/hadoop$ jps
59184 Jps
58690 NodeManager
58483 ResourceManager
52166 SecondaryNameNode
58074 DataNode
51724 NameNode
```

如果没有初始化的话会遇到很多问题，例如

```
root@ubuntu:~/hadoop/bin# hdfs dfs -mkdir /dataset
20/06/23 21:32:42 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
mkdir: Call From ubuntu/127.0.1.1 to localhost:9000 failed on connection exception: java.net.ConnectException: 拒绝连接; For more details see: http://wiki.apache.org/hadoop/ConnectionRefused
```

重复执行的话会提示节点占用资源，应该先stop所有资源


```
$ /usr/local/hadoop/sbin/stop-all.sh
```

```
localhost: nodemanager running as process 106296. Stop it first.  
eewhile@ubuntu:/usr/local/hadoop$ sbin/stop-all.sh
```

2.6 在外部浏览器输入 master 的 IP 地址和 50070 端口的「Datanodes」查看 hdfs 上的节点

The screenshot shows a web browser window with the address bar displaying 'localhost:50070/dfshealth.html#tab-datanode'. The browser's tab bar shows several tabs, including 'Gmail', 'YouTube', 'Maps', and several 'hadoop' related tabs. The page has a green header bar with navigation links: 'Hadoop', 'Overview', 'Datanodes', 'Datanode Volume Failures', 'Snapshot', 'Startup Progress', and 'Utilities'. The main content area is titled 'Datanode Information' and has a sub-header 'In operation'. Below this is a table with 11 columns: 'Node', 'Last contact', 'Admin State', 'Capacity', 'Used', 'Non DFS Used', 'Remaining', 'Blocks', 'Block pool used', 'Failed Volumes', and 'Version'. The table contains one row of data for 'ubuntu:50010 (127.0.0.1:50010)'. Below the table is a section titled 'Decommissioning' with a sub-header 'Under replicated blocks'. This section contains a table with 5 columns: 'Node', 'Last contact', 'Under replicated blocks', 'Blocks with no live replicas', and 'Under Replicated Blocks In files under construction'. At the bottom of the page, it says 'Hadoop, 2014.'

Node	Last contact	Admin State	Capacity	Used	Non DFS Used	Remaining	Blocks	Block pool used	Failed Volumes	Version
ubuntu:50010 (127.0.0.1:50010)	2	In Service	19.07 GB	24 KB	15.03 GB	4.04 GB	0	24 KB (0%)	0	2.7.0

Node	Last contact	Under replicated blocks	Blocks with no live replicas	Under Replicated Blocks In files under construction
------	--------------	-------------------------	------------------------------	---

3 MapReduce实现

一个简单的 MapReduce 程序一般由三部分组成。一个是继承了 Mapper 类的新Mapper，Mapper 的数据是以 key—value 对的形式，其中 Map 的业务逻辑写在 map 函数中，map 函数调用每个 key—value 对一次，然后输出 key—value 对。一个是继承了 Reducer 类的新Reducer，Reduce 的数据是以 key—value 对形式输入的，其中 Reduce 的业务逻辑写在 reduce 函数中，ReduceTask 进程对每一组相同 key 的 key—value 对进行一次 reduce 操作。还有一个main 函数，其中用 Job 管理该程序，比如指定 Mapper 和 Reducer 以及其他相关操作。

其中的切片机制，FileInputFormat 中的默认切片机制允许简单地按照文件的内容长度进行切片，切片的大小等于block 大小，不考虑整体数据集，只是针对每一个文件单独进行切片，切片的大小由 blocksize、maxsize、minsize 共同决定。

根据以上的原理实现PageRank算法

代码如下：

3.1 NodeCoverter.java

```
/*  
NodeCovert_17341018_陈景宇_FinalProj_2020_6_27  
*/
```


这部分代码主要是进行数据处理，进行格式上的转化，以方便后续的迭代工作。

数据格式被转化成：“起始节点 ID + PageRank值 + SPACE + 目标节点1 ID + 目标节点2 ID + 目标节点3 ID.....”。

```
*****/

import java.net.URI;
import java.util.Set;
import java.util.HashSet;
import java.io.IOException;
import java.util.ArrayList;
import java.util.StringTokenizer;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import java.net.URISyntaxException;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.Counters;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.mapred.Counters.Counter;
import org.apache.hadoop.mapreduce.Reducer.Context;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class NodeConverter {

    public static void main(String[] args) throws IOException,
        InterruptedException, ClassNotFoundException, URISyntaxException {
        Configuration conf = new Configuration();
        // 输入的文件路径，也就是soc-Epinions1.txt的内容
        String pathIn = "/dataset";
        // 输出文件路径，作为后面pagerank的输入文件内容
        String pathOut = "/dataconvert";

        // IP源
        // 这里要和$HADOOP_HOME/etc/hadoop/core-site.xml文件中的
        // fs.defaultFS NameNode URI      hdfs://host:port/ 对应好
        FileSystem.setDefaultUri(conf, new URI("hdfs://localhost:9000"));

        // 数据预处理
        Job job = new Job(conf, "MapReduce pretreatment");
        job.setJarByClass(NodeConverter.class);

        job.setMapperClass(NodeConvertMapper.class);
        job.setReducerClass(NodeConvertReducer.class);

        // output map类与reduce类相同
        job.setOutputKeyClass(IntWritable.class);
        job.setOutputValueClass(Text.class);

        FileInputFormat.addInputPath(job, new Path(pathIn));
        FileOutputFormat.setOutputPath(job, new Path(pathOut));
    }
}
```

```

        System.exit(job.waitForCompletion(true) ? 0:1);
    }
}

```

3.2 NodeCoverterMapper.java

```

/*****
NodeCovert_17341018_陈景宇_FinalProj_2020_6_27
*****/
/*****/

NodeConvertMapper 将 split 之后起始节点的目标节点们找到：
类中的私有变量 id 和 nextId 分别表示起始节点的 ID 和目标节点的 ID
起始节点的 ID 作为 key，目标节点的 ID 作为 value 组成 key - value 对
map() 方法利用 Token 进行读数，目标节点之前加上 '$' 符号，
便于区分起始节点和目标节点，写入 context。

*****/

import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.mapreduce.Mapper;

public class NodeConvertMapper extends Mapper<Object, Text, IntWritable, Text> {

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

        IntWritable id;
        IntWritable nextId;

        // 标记字符串
        StringTokenizer str = new StringTokenizer(value.toString());

        // 读入下一个标记
        if(str.hasMoreTokens()) {
            id = new IntWritable(Integer.parseInt(str.nextToken()));
        } else{
            return;
        }

        // &: address
        nextId = new IntWritable(Integer.parseInt(str.nextToken()));

        // 写入文档
        context.write(id, new Text("&" + nextId));

    }

}

```

3.3 NodeCoverterReduce.java

```
/*
NodeCovert_17341018_陈景宇_FinalProj_2020_6_27

把 shuffle 之后的结果聚合到一起：
私有成员 idList 采用 ArrayList 的存储结构
内存 String 类型的目标节点 ID; reduce() 方法把 Mapper 中 text 的数据转化成 String 形式
按照标示符，数据被整理成——“起始节点 ID + PageRank值 + SPACE + 目标节点1 ID + 目标节点2
ID + 目标节点3 ID.....”格式
迭代开始的时候，每个节点默认的 PageRank 值设为1。

import java.io.IOException;
import java.util.ArrayList;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.FloatWritable;
import org.apache.hadoop.mapreduce.Reducer;

public class NodeConvertReducer extends Reducer<IntWritable, Text, IntWritable,
Text> {

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

        // idList: address list
        ArrayList<String> idList = new ArrayList<String>();

        // idString: address
        for(Text id : values) {
            String idStr = id.toString();
            if (idStr.substring(0,1).equals("&")) {
                idList.add(idStr.substring(1));
            }
        }
        // departure pagerank
        float pr = 1.0f;
        // result string
        String result = String.valueOf(pr) + " ";
        for(int i = 0; i< idList.size();i++){
            result += idList.get(i) + " ";
        }

        // 写入
        context.write(key, new Text(result));
    }
}
```

3.4 PageRank.java

```
/******
```

```
PageRank_17341018_陈景宇_FinalProj_2020_6_27
```

```
*****/  
/******
```

起始节点的 pr 值对目标节点 pr 值进行贡献，并不停进行迭代。PageRank值通常需要迭代30-40次才能达到收敛。

数据格式为“起始节点 ID + PageRank值 + SPACE + 目标节点1 ID + 目标节点2 ID + 目标节点3 ID.....”。

```
*****/
```

```
import java.net.URI;  
import java.util.Set;  
import java.util.HashSet;  
import java.io.IOException;  
import java.util.ArrayList;  
import org.apache.hadoop.io.Text;  
import org.apache.hadoop.fs.Path;  
import java.util.StringTokenizer;  
import java.net.URISyntaxException;  
import org.apache.hadoop.fs.FileSystem;  
import org.apache.hadoop.mapreduce.Job;  
import org.apache.hadoop.io.IntWritable;  
import org.apache.hadoop.io.LongWritable;  
import org.apache.hadoop.mapreduce.Mapper;  
import org.apache.hadoop.mapreduce.Reducer;  
import org.apache.hadoop.mapreduce.Counters;  
import org.apache.hadoop.conf.Configuration;  
import org.apache.hadoop.mapred.Counters.Counter;  
import org.apache.hadoop.mapreduce.Reducer.Context;  
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;  
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;  
  
public class PageRank {  
    public static void main(String[] args) throws IOException,  
        InterruptedException, ClassNotFoundException, URISyntaxException {  
        Configuration conf = new Configuration();  
        // 输入的文件路径： 是NodeConvert数据格式转化的结果  
        String pathIn = "/dataconvert";  
        // 定义输出文件名  
        String output = "/dataoutput";  
  
        // 输出的文件路径： 这就是我们最后查看的，也是最重要的结果  
        String pathOut = "/dataoutput";  
  
        // IP源  
        // 这里要和$HADOOP_HOME/etc/hadoop/core-site.xml文件中的  
        // fs.defaultFS NameNode URI hdfs://host:port/ 对应好  
        FileSystem.setDefaultUri(conf, new URI("hdfs://localhost:9000"));
```

```

// 第一次输出文件的结果，以为后面需要进行迭代多次处理，所以记录序号方便查看
pathOut = output + "1";

// 多次迭代，也是pagerank mapreduce的和兴处理部分
// 其中的迭代次数可以是正无穷，但是到达一定的次数时候，就可以看出收敛的结果
// PageRank值通常需要迭代30-40次才能达到收敛
// 为了减少不必要的操作量，这里就执行了30次迭代最后再查看结果
for (int i = 1; i < 30; i++) {
    Job job = new Job(conf, "MapReduce pagerank");
    job.setJarByClass(PageRank.class);

    job.setMapperClass(PageRankMapper.class);
    job.setReducerClass(PageRankReducer.class);

    // output map类与reduce类相同
    job.setOutputKeyClass(IntWritable.class);
    job.setOutputValueClass(Text.class);

    FileInputFormat.addInputPath(job, new Path(pathIn));
    FileOutputFormat.setOutputPath(job, new Path(pathOut));

    // 新一轮的输入为上一轮的输出文件，这是迭代的典型代表
    pathIn = pathOut;
    pathOut = output + (i + 1);

    // 等待执行完成
    job.waitForCompletion(true);
}
}
}

```

3.5 PageRankMapper

```

/*****

PageRank_17341018_陈景宇_FinalProj_2020_6_27

*****/

对数据进行了平均加权：
私有成员 id 表示起始节点的 ID、pr 表示目标节点 ID、count 给目标节点记数、average_pr 计算平均贡献
起始节点的 ID 作为 key，其权值和目标节点的 ID 通过不同的符号标示作为 value，组成 key - value 对
map() 方法首先对起始节点的目标节点个数进行了记数；然后用起始节点的 PageRank 值除以目标节点个数，
得到了平均值——起始节点对每个目标节点 PageRank 值的平均贡献；最后，用 ‘$’ 标示对目标节点们的平均 PageRank 贡献、用 ‘&’ 标示目标节点们。

*****/

import java.io.IOException;
import java.util.ArrayList;

```

```

import org.apache.hadoop.io.Text;
import java.util.StringTokenizer;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.mapreduce.Mapper;

public class PageRankMapper extends Mapper<Object, Text, IntWritable, Text> {
    private IntWritable id;
    private String nextId;
    private int count;
    private float averagePr;

    public void map(Object key, Text value, Context context) throws IOException,
        InterruptedException{
        // 标记字符串
        StringTokenizer str = new StringTokenizer(value.toString());

        // 读入下一个标记字符串
        if (str.hasMoreTokens()) {
            id = new IntWritable(Integer.parseInt(str.nextToken()));
        } else {
            return;
        }

        nextId = str.nextToken();

        // 获取标记字符串的计数
        count = str.countTokens();

        // 各标记字符串平均奖励
        averagePr = Float.parseFloat(nextId) / count;

        // $: departure pagerank value
        // &: address
        while (str.hasMoreTokens()) {

            String nextId = str.nextToken();

            Text tmpText = new Text("$" + averagePr);
            context.write(new IntWritable(Integer.parseInt(nextId)), tmpText);

            tmpText = new Text("&" + nextId);
            context.write(id, tmpText);
        }
    }
}

```

3.6 PageRankReduce.java

```

/*****

PageRank_17341018_陈景宇_FinalProj_2020_6_27

*****/
/*****

```

```

*****/

import java.io.IOException;
import java.util.ArrayList;
import java.util.StringTokenizer;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.FloatWritable;
import org.apache.hadoop.mapreduce.Reducer;

public class PageRankReducer extends Reducer<IntWritable, Text, IntWritable,
Text> {

    // 自行设计阿尔法贝塔值
    private static float alpha = 0.85f; // alpha一般都是取0.85
    private static float bonus = 0.2f;

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

        // idList: address list
        ArrayList<String> idList = new ArrayList<String>();
        // 输出
        String output = " ";
        float pr = 0;

        // $: departure pagerank value
        // &: address
        for (Text id : values) {
            String idStr = id.toString();
            if (idStr.substring(0, 1).equals("$")) {
                pr += Float.parseFloat(idStr.substring(1));
            }
            else if (idStr.substring(0, 1).equals("&")) {
                String nextId = idStr.substring(1);
                idList.add(nextId);
            }
        }
        // pagerank 计算
        pr = pr * alpha + bonus;

        // 目标输出
        for (int i = 0; i < idList.size(); i++) {
            output = output + idList.get(i) + " ";
        }

        // 目标字符串
        String result = pr + output;

        // 写入
        context.write(key, new Text(result));
    }
}

```


4 PageRank实践

因为我的虚拟机和本地机的共享文件夹为 `/mnt/hgfs/linuxshare`, 故在这个目录下编译也没有问题的, 而且本地也有了编译的结果

4.1 导入数据集

建立 hdfs 的 `/dataset` 文件夹 (`/usr/local/hadoop/bin/hdfs dfs -mkdir /dataset`), 导入数据集 `soc-Epinions1.txt` 到 hdfs 的 `/dataset` 文件夹。数据集一共包含75888个节点, 以及508837条指向关系。

```
$ /usr/local/hadoop/bin/hdfs dfs -put /mnt/hgfs/linuxshare/NodeConverter/soc-Epinions1.txt /dataset
```

The screenshot shows the SNAP website interface. The main content area displays the 'Epinions 社交网络' dataset page. It includes a sidebar with navigation links, a main content area with dataset statistics, and a source section.

数据信息

这是一般消费者评论网站Epinions.com的在线信任社交网络。该站点的成员可以决定是否彼此“信任”。所有的信任关系相互作用并形成信任网络, 然后将其与评论等级组合以确定向用户显示哪些评论。

数据集统计

节点数	75879
边缘	508837
最大的WCC中的节点	75877 (1.000)
最大的WCC优势	508836 (1.000)
最大SCC中的节点	32223 (0.425)
最大SCC中的优势	443506 (0.872)
平均聚类系数	0.1378
三角形数	1624481
闭合三角形的分数	0.0229
直径 (最长的最短路径)	14
90%有效直径	5

来源 (引文)

- 理查森 (M. Richardson) 和 R. Agrawal 和 P. Domingos. 语义Web的信任管理. ISWC, 2003年。

档案

文件	描述
soc-Epinions1.txt.gz	定向Epinions社交网络

Browse Directory

/dataset								Go!
Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name	
-rw-r--r--	eewhile	supergroup	5.4 MB	2020/6/26 下午9:05:37	1	128 MB		

Hadoop, 2014.

4.2 编译NodeCoverter

```
# 首先来到/mnt/hgfs/linuxshare/NodeConverter目录下
$ cd /mnt/hgfs/linuxshare/NodeConverter
$ javac NodeCoverter.java
# 提示过时的API不影响编译的正确性

#将生成的 NodeConvertMapper.class、NodeConvertReducer.class、NodeConverter.class
包装成 jar 包
$ jar -cvf NodeConverter.jar *.class
```

```
eeehille@ubuntu:~$ cd /mnt/hgfs/linuxshare/
eeehille@ubuntu:~/mnt/hgfs/linuxshare$ cd NodeConverter/
eeehille@ubuntu:~/mnt/hgfs/linuxshare/NodeConverter$ javac NodeConverter.java
注: NodeConverter.java使用或覆盖了已过时的 API。
注: 有关详细信息, 请使用 -Xlint:deprecation 重新编译。
eeehille@ubuntu:~/mnt/hgfs/linuxshare/NodeConverter$ ls
NodeConvertMapper.class NodeConverter.class NodeConvertMapper.java NodeConvertReducer.class NodeConvertReducer.java soc-Epinions1.txt
eeehille@ubuntu:~/mnt/hgfs/linuxshare/NodeConverter$ jar -cvf NodeConverter.jar *.class
已添加清单
正在添加: NodeConverter.class(输入 = 1762) (输出 = 942)(压缩了 46%)
正在添加: NodeConvertMapper.class(输入 = 1840) (输出 = 784)(压缩了 57%)
正在添加: NodeConvertReducer.class(输入 = 2231) (输出 = 982)(压缩了 55%)
```

如果在 .bashrc 中没有加入classpath 将会出现下面问题

```
eeehille@ubuntu:~/mnt/hgfs/linuxshare/NodeConverter$ javac NodeConverter.java
NodeConverter.java:17: 错误: 程序包org.apache.hadoop.fs不存在
import org.apache.hadoop.fs.Path;
                        ^
NodeConverter.java:18: 错误: 程序包org.apache.hadoop.io不存在
import org.apache.hadoop.io.Text;
                        ^
NodeConverter.java:20: 错误: 程序包org.apache.hadoop.fs不存在
import org.apache.hadoop.fs.FileSystem;
                        ^
NodeConverter.java:21: 错误: 程序包org.apache.hadoop.mapreduce不存在
import org.apache.hadoop.mapreduce.Job;
                        ^
NodeConverter.java:22: 错误: 程序包org.apache.hadoop.io不存在
import org.apache.hadoop.io.IntWritable;
                        ^
NodeConverter.java:23: 错误: 程序包org.apache.hadoop.io不存在
import org.apache.hadoop.io.LongWritable;
                        ^
NodeConverter.java:24: 错误: 程序包org.apache.hadoop.mapreduce不存在
import org.apache.hadoop.mapreduce.Mapper;
                        ^
NodeConverter.java:25: 错误: 程序包org.apache.hadoop.mapreduce不存在
import org.apache.hadoop.mapreduce.Reducer;
                        ^
NodeConverter.java:26: 错误: 程序包org.apache.hadoop.mapreduce不存在
import org.apache.hadoop.mapreduce.Counters;
                        ^
NodeConverter.java:27: 错误: 程序包org.apache.hadoop.conf不存在
import org.apache.hadoop.conf.Configuration;
                        ^
NodeConverter.java:28: 错误: 程序包org.apache.hadoop.mapred.Counters不存在
import org.apache.hadoop.mapred.Counters.Counter;
                        ^
```

编译成果的过程图:

名称	修改日期	类型	大小
.DS_Store	2020/6/18 17:01	DS_STORE 文件	7 KB
NodeConverter.class	2020/6/27 11:50	CLASS 文件	2 KB
NodeConverter.jar	2020/6/27 11:53	Executable Jar File	4 KB
NodeConverter.java	2020/6/27 16:37	JAVA 文件	3 KB
NodeConvertMapper.class	2020/6/27 11:50	CLASS 文件	2 KB
NodeConvertMapper.java	2020/6/27 15:57	JAVA 文件	2 KB
NodeConvertReducer.class	2020/6/27 11:50	CLASS 文件	3 KB
NodeConvertReducer.java	2020/6/27 16:21	JAVA 文件	2 KB

```

20/06/26 21:05:47 INFO Configuration.deprecation: mapred.skip.on is deprecated. Instead, use mapreduce.job.skiprecords
20/06/26 21:05:47 INFO mapred.Task: Task:attempt_local1603655705_0001_r_000000_0 is done. And is in the process of committing
20/06/26 21:05:47 INFO mapred.LocalJobRunner: 1 / 1 copied.
20/06/26 21:05:47 INFO mapred.Task: Task attempt_local1603655705_0001_r_000000_0 is allowed to commit now
20/06/26 21:05:47 INFO output.FileOutputCommitter: Saved output of task 'attempt_local1603655705_0001_r_000000_0' to hdfs://localhost:9000/dataconvert/_tempor
705_0001_r_000000
20/06/26 21:05:47 INFO mapred.LocalJobRunner: reduce > reduce
20/06/26 21:05:47 INFO mapred.Task: Task 'attempt_local1603655705_0001_r_000000_0' done.
20/06/26 21:05:47 INFO mapred.LocalJobRunner: Finishing task: attempt_local1603655705_0001_r_000000_0
20/06/26 21:05:47 INFO mapred.LocalJobRunner: reduce task executor complete.
20/06/26 21:05:48 INFO mapreduce.Job: map 100% reduce 100%
20/06/26 21:05:48 INFO mapreduce.Job: Job job_local1603655705_0001 completed successfully
20/06/26 21:05:48 INFO mapreduce.Job: Counters: 35
File System Counters
  FILE: Number of bytes read=12212882
  FILE: Number of bytes written=18870297
  FILE: Number of read operations=0
  FILE: Number of large read operations=0
  FILE: Number of write operations=0
  HDFS: Number of bytes read=11325822
  HDFS: Number of bytes written=3193885
  HDFS: Number of read operations=13
  HDFS: Number of large read operations=0
  HDFS: Number of write operations=4
Map-Reduce Framework
  Map input records=508837
  Map output records=508837
  Map output bytes=5085143
  Map output materialized bytes=6102823
  Input split bytes=94
  Combine input records=0
  Combine output records=0
  Reduce input groups=60341
  Reduce shuffle bytes=6102823
  Reduce input records=508837
  Reduce output records=60341
  Spilled Records=1017674
  Shuffled Maps =1
  Failed Shuffles=0
  Merged Map outputs=1
  GC time elapsed (ms)=43
  Total committed heap usage (bytes)=606076928
Shuffle Errors
  BAD_ID=0
  CONNECTION=0
  IO_ERROR=0

```

查看 datacover 结果

```
# 内容太长只能节选了
/usr/local/hadoop/bin/hdfs dfs -cat /dataconvert/*
```

```

75861 1.0 1438
75862 1.0 1438
75863 1.0 1438
75864 1.0 75825
75865 1.0 75829 75872 75873
75866 1.0 75837
75867 1.0 75874 75837 75868
75869 1.0 75868 75837
75871 1.0 75842 75875 75876 75878 75877
75872 1.0 75865 75879 75880 75881
75873 1.0 75865
75876 1.0 75871 75877
75877 1.0 75871 75876
75878 1.0 75871
75880 1.0 75872
75881 1.0 75872
75882 1.0 75872
75883 1.0 75872
75884 1.0 1699
75885 1.0 7900 16086
75886 1.0 51414
75887 1.0 52098

```

根据这个结果我们得到的是，“**起始节点 ID + PageRank值 + SPACE + 目标节点1 ID + 目标节点2 ID + 目标节点3 ID.....**”为数据的格式

这里我们可以根据75873为例：

75872	为起始节点ID
1.0	为PageRank值
75865 75879 75880 75881	均为目标节点

4.3 编译PageRank

首先为了确保目标文件夹下没有文件，要删除目标文件夹和 /tmp 文件夹，不然会导致无法成功 MapReduce jar 包：

```

$ /usr/local/hadoop/bin/hdfs dfs -rm -r /dataoutput* /tmp
# 当然如果还没执行过的话肯定是没有如上文件的

```

```

eewhile@ubuntu:~$ /usr/local/hadoop/bin/hdfs dfs -rm -r /dataoutput* /tmp
rm: `/dataoutput*': No such file or directory
rm: `/tmp': No such file or directory

```

```

# 首先来到/mnt/hgfs/linuxshare/PageRank目录下
$ cd /mnt/hgfs/linuxshare/PageRank
$ javac PageRank.java
# 提示过时的API不影响编译的正确性

#将生成的 NodeConvertMapper.class、NodeConvertReducer.class、NodeConverter.class
包装成 jar 包
$ jar -cvf PageRank.jar *.class

```

```

eeewhile@ubuntu:~$ cd /mnt/hgfs/linuxshare/PageRank/
eeewhile@ubuntu:/mnt/hgfs/linuxshare/PageRank$ javac PageRank.java
注: PageRank.java使用或覆盖了已过时的 API。
注: 有关详细信息, 请使用 -Xlint:deprecation 重新编译。
eeewhile@ubuntu:/mnt/hgfs/linuxshare/PageRank$ jar -cvf PageRank.jar *.class
已添加清单
正在添加: PageRank.class(输入 = 1967) (输出 = 1048)(压缩了 46%)
正在添加: PageRankMapper.class(输入 = 2162) (输出 = 974)(压缩了 54%)
正在添加: PageRankReducer.class(输入 = 2539) (输出 = 1174)(压缩了 53%)

```

DATA (E:) > ubuntu > linuxshare > PageRank

名称	修改日期	类型	大小
 PageRank.class	2020/6/27 20:14	CLASS 文件	2 KB
 PageRank.jar	2020/6/27 20:15	Executable Jar File	4 KB
 PageRank.java	2020/6/27 16:37	JAVA 文件	4 KB
 PageRankMapper.class	2020/6/27 20:14	CLASS 文件	3 KB
 PageRankMapper.java	2020/6/27 16:42	JAVA 文件	3 KB
 PageRankReducer.class	2020/6/27 20:14	CLASS 文件	3 KB
 PageRankReducer.java	2020/6/27 17:39	JAVA 文件	2 KB

利用 NodeConverter 的结果输出作为输入, MapReduce 运行 jar 包, MapReduce NodeConverter.jar 程序:

```

$ /usr/local/hadoop/bin/hadoop jar PageRank.jar PageRank /dataconvert
/dataoutput
# 执行了30次的迭代

```

localhost:50070/explorer.html#/dataoutput1

mail YouTube Maps hadoop2.7.1... 解决问题:Inpu... Hadoop解决W... hadoop 关闭... Hadoop问题... Resources are... Hadoop配置... Lir

Hadoop Overview Datanodes Snapshot Startup Progress Utilities

Browse Directory

/dataoutput1

Go!

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	eeewhile	supergroup	0 B	2020/6/27 上午5:17:30	1	128 MB	._SUCCESS
-rw-r--r--	eeewhile	supergroup	4.06 MB	2020/6/27 上午5:17:30	1	128 MB	part-r-00000

Hadoop, 2014.

localhost:50070/explorer.html#/dataoutput29

mail

YouTube

Maps

hadoop2.7.1...

解决问题:Inpu...

Hadoop解决W...

hadoop 关闭d...

Hadoop问题...

Resources are...

Hadoop配置...

Linux

HadoopOverviewDatanodesSnapshotStartup ProgressUtilities

Browse Directory

/dataoutput29

Go!

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	eewhile	supergroup	0 B	2020/6/27 上午5:19:18	1	128 MB	_SUCCESS
-rw-r--r--	eewhile	supergroup	4.08 MB	2020/6/27 上午5:19:18	1	128 MB	part-r-00000

Hadoop, 2014.

利用 hdfs 的 cat 功能显示输出第一次迭代时候文件夹下的数据：

```
$ /usr/local/hadoop/bin/hdfs dfs -cat /dataoutput1/*
```

```

75843 0.2 74676
75844 0.2 74697
75845 0.625 75737 75414
75846 0.625
75847 0.20330739
75848 0.20330739
75849 0.20330739
75850 0.2 751
75851 0.2 751
75852 0.2 751
75853 0.2 751
75854 0.2 751
75855 0.2 751
75856 0.2 751
75857 0.2 751
75858 0.2 751
75859 0.2 751
75860 0.2 751
75861 0.2 1438
75862 0.2 1438
75863 0.2 1438
75864 0.2 75825
75865 1.2625 75873 75829 75872
75866 0.48333335 75837
75867 0.48333335 75874 75837 75868
75868 1.1916667
75869 0.2 75837 75868
75870 0.625
75871 2.325 75875 75842 75876 75878 75877
75872 3.8833332 75865 75879 75880 75881
75873 0.48333335 75865
75874 0.48333335
75875 0.37
75876 0.795 75877 75871
75877 0.795 75871 75876
75878 0.37 75871
75879 0.41250002
75880 0.41250002 75872
75881 0.41250002 75872
75882 0.2 75872
75883 0.2 75872
75884 0.2 1699
75885 0.2 7900 16086
75886 0.2 51414
75887 0.2 52098

```

根据这个结果我们得到的是，“**起始节点 ID + PageRank值 + SPACE + 目标节点1 ID + 目标节点2 ID + 目标节点3 ID.....**”为数据的格式

这里我们可以根据75876为例：（内容太多没必要全部截出来）

75876	为起始节点ID
0.795	为PageRank值
75877 75871	均为目标节点

#试看第五次迭代的结果（节选）

\$ /usr/local/hadoop/bin/hdfs dfs -cat /dataoutput5/*

```
65960 0.4146185
65961 0.68623817
65962 0.2 40344
65963 0.2 40344
65964 0.2 40344
65965 0.2 40352
65966 0.2 40359
65967 0.2 40363
65968 0.2 40363
65969 0.419643
65970 0.5542866
65971 0.2 40378
65972 0.625 40386
65973 0.2 40386
65974 0.32355696 40388
65975 0.32355696 40388
65976 0.31947464 40390 66742
65977 0.2 40394
65978 0.285
65979 0.29975957 40417
65980 0.2 40420
65981 0.2914724
65982 0.33814842 40434
65983 0.33814842 40434
65984 0.2 40436
65985 0.44005322 40447
65986 0.2 40448
65987 0.46640974 40451
65988 0.2 40462
65989 0.26441777
65990 0.2 40485
65991 0.7063616 40495 46269
65992 0.4368865 40495
65993 0.4368865 40495
65994 0.2 40496
65995 0.2 40499
65996 0.22779216
65997 0.22779216
65998 0.22779216
65999 0.22779216 854
66000 0.25666666
66001 0.35854405
66002 0.2 40523
66003 0.2 40526
66004 0.44001824 40527
66005 1.2437683 40532
```

66113	0.24372953	40940			
66114	0.29994106				
66115	0.2	40946			
66116	0.2	40946			
66117	0.24481039				
66118	0.2912085	40951			
66119	0.2912085				
66120	0.46951973	40959			
66121	0.25666666				
66122	0.30483335				
66123	0.2	40970			
66124	0.234				
66125	0.234				
66126	0.2	40992			
66127	0.2	40993			
66128	0.32136536	1719	66129	790	40993
66129	0.26886678	66128	40993		
66130	0.2	40999			
66131	0.38247472				
66132	0.2	41009			
66133	0.56513554				
66134	0.4002706	41025			
66135	0.4002706	41025			
66136	0.2	41045			
66137	0.4043184	41051			
66138	0.2	41051			
66139	0.35038102	41055			
66140	0.8736803				
66141	0.38318378				
66142	0.58295715	52193	41060	42157	66143
66143	0.32571948	41060			
66144	0.2	41060			
66145	0.4513482	52193	66142	41060	
66146	0.2	388	41061		
66147	0.37	41066			
66148	0.2	41067			
66149	0.41278607				
66150	2.3278608	70607	70608	41069	70606
66151	0.2	41072			
66152	0.2	41072			
66153	0.2	41074			

66897	0.2	44238							
66898	0.8414394	18	70723						
66899	0.7160257								
66900	0.2	44247							
66901	0.2	44253							
66902	0.2	44253							
66903	0.2	44255							
66904	2.037591	44259	66905						
66905	1.5871804	66904							
66906	0.5584881	44265							
66907	1.5118337	44268							
66908	0.2	44268							
66909	0.280831								
66910	0.2	44280	70725						
66911	0.4237858	44294							
66912	0.4237858	44294							
66913	0.4237858	44294							
66914	0.234								
66915	0.2	44303							
66916	0.49488556	44313							
66917	0.29042622								
66918	0.46951973	44323							
66919	0.52782726	44333							
66920	0.52782726	44333							
66921	0.32616743								
66922	0.25666666								
66923	0.2952578	1815	18	1719	4415	790	1621		
66924	0.38545454								
66925	0.25666666								
66926	0.80507976								
66927	1.214894	44367							
66928	0.2	44371							
66929	0.2	44371							
66930	0.5591578	44377	70730	66931	70731				
66931	2.1538825	70731	44377						
66932	0.401631	44407							
66933	1.2274991	44424							
66934	0.2	44436							
66935	0.28711915								
66936	0.27339625								
66937	0.39643443	44459							

75843	0.2	74676				
75844	0.2	74697				
75845	0.35686898	75737	75414			
75846	0.40395287					
75847	0.28503066					
75848	0.28503066					
75849	0.28503066					
75850	0.2	751				
75851	0.2	751				
75852	0.2	751				
75853	0.2	751				
75854	0.2	751				
75855	0.2	751				
75856	0.2	751				
75857	0.2	751				
75858	0.2	751				
75859	0.2	751				
75860	0.2	751				
75861	0.2	1438				
75862	0.2	1438				
75863	0.2	1438				
75864	0.2	75825				
75865	0.9849504	75872	75829	75873		
75866	0.42731547	75837				
75867	0.42731547	75837	75868	75874		
75868	0.6568078					
75869	0.2	75837	75868			
75870	0.41428977					
75871	1.5599312	75876	75842	75878	75875	75877
75872	2.2164338	75879	75881	75880	75865	
75873	0.56476766	75865				
75874	0.34449232					
75875	0.44680876					
75876	0.8077908	75871	75877			
75877	0.8077908	75871	75876			
75878	0.44680876	75871				
75879	0.5707862					
75880	0.5707862	75872				
75881	0.5707862	75872				
75882	0.2	75872				
75883	0.2	75872				
75884	0.2	1699				
75885	0.2	16086	7900			
75886	0.2	51414				
75887	0.2	52098				

这里我们继续以75876为例：（内容太多没必要全部截出来）

75876	为起始节点ID
0.8077908	为PageRank值（发生了迭代出现的结果变化）
75877 75871	均为目标节点

接下来都是对这个节点查看第 9、15、19、25、29的迭代结果观察，查看迭代多次的结果出现的变化

```
75876    0.7810285  75877  75871
```

```
75876    0.7690137  75877  75871
```

```
75876    0.76723295 75871  75877
```

```
75876    0.7665813  75871  75877
```

```
75876    0.7664908  75871  75877
```

可以看出75876这个节点逐渐收敛到一个比较固定的值.....次数足够多下会更收敛精准（这里迭代次数为30）

4.4 导出 dataconvert 和 dataoutput 文件内容

```
$ /usr/local/hadoop/bin/hdfs dfs -get /dataconvert/*  
/mnt/hgfs/linuxshare/dataconvert/  
$ /usr/local/hadoop/bin/hdfs dfs -get /dataoutput1/*  
/mnt/hgfs/linuxshare/dataoutput/dataoutput1
```



```
eewhile@ubuntu:/mnt/hgfs/linuxshare/PageRank$ /usr/local/hadoop/bin/hdfs dfs -get /user/hadoop/dataoutput1/* /mnt/hgfs/linuxshare/  
get: '/user/hadoop/dataoutput1/*': No such file or directory  
eewhile@ubuntu:/mnt/hgfs/linuxshare/PageRank$ /usr/local/hadoop/bin/hdfs dfs -get dataoutput1/* /mnt/hgfs/linuxshare/  
get: 'dataoutput1/*': No such file or directory  
eewhile@ubuntu:/mnt/hgfs/linuxshare/PageRank$ /usr/local/hadoop/bin/hdfs dfs -get /dataoutput1/* /mnt/hgfs/linuxshare/  
20/06/27 05:40:11 WARN hdfs.DFSClient: DFSInputStream has been closed already  
20/06/27 05:40:11 WARN hdfs.DFSClient: DFSInputStream has been closed already  
eewhile@ubuntu:/mnt/hgfs/linuxshare/PageRank$ /usr/local/hadoop/bin/hdfs dfs -get /dataconvert/* /mnt/hgfs/linuxshare/dataconvert/  
get: '/dataconvert/*': No such file or directory  
eewhile@ubuntu:/mnt/hgfs/linuxshare/PageRank$ /usr/local/hadoop/bin/hdfs dfs -get /dataconvert/* /mnt/hgfs/linuxshare/dataconvert/  
20/06/27 05:43:24 WARN hdfs.DFSClient: DFSInputStream has been closed already  
20/06/27 05:43:24 WARN hdfs.DFSClient: DFSInputStream has been closed already  
eewhile@ubuntu:/mnt/hgfs/linuxshare/PageRank$ /usr/local/hadoop/bin/hdfs dfs -get /dataoutput1/* /mnt/hgfs/linuxshare/dataoutput/dataoutput1  
20/06/27 05:50:58 WARN hdfs.DFSClient: DFSInputStream has been closed already  
20/06/27 05:50:58 WARN hdfs.DFSClient: DFSInputStream has been closed already
```

```
eewhile@ubuntu:/mnt/hgfs/linuxshare/PageRank$ /usr/local/hadoop/bin/hdfs dfs -get /dataoutput2/* /mnt/hgfs/linuxshare/dataoutput/dataoutput2  
20/06/27 06:00:10 WARN hdfs.DFSClient: DFSInputStream has been closed already  
20/06/27 06:00:11 WARN hdfs.DFSClient: DFSInputStream has been closed already  
eewhile@ubuntu:/mnt/hgfs/linuxshare/PageRank$ /usr/local/hadoop/bin/hdfs dfs -get /dataoutput3/* /mnt/hgfs/linuxshare/dataoutput/dataoutput3  
20/06/27 06:00:21 WARN hdfs.DFSClient: DFSInputStream has been closed already  
20/06/27 06:00:21 WARN hdfs.DFSClient: DFSInputStream has been closed already  
eewhile@ubuntu:/mnt/hgfs/linuxshare/PageRank$ /usr/local/hadoop/bin/hdfs dfs -get /dataoutput4/* /mnt/hgfs/linuxshare/dataoutput/dataoutput4  
20/06/27 06:00:32 WARN hdfs.DFSClient: DFSInputStream has been closed already  
20/06/27 06:00:32 WARN hdfs.DFSClient: DFSInputStream has been closed already  
eewhile@ubuntu:/mnt/hgfs/linuxshare/PageRank$ /usr/local/hadoop/bin/hdfs dfs -get /dataoutput5/* /mnt/hgfs/linuxshare/dataoutput/dataoutput5  
20/06/27 06:00:44 WARN hdfs.DFSClient: DFSInputStream has been closed already  
20/06/27 06:00:44 WARN hdfs.DFSClient: DFSInputStream has been closed already  
eewhile@ubuntu:/mnt/hgfs/linuxshare/PageRank$ /usr/local/hadoop/bin/hdfs dfs -get /dataoutput6/* /mnt/hgfs/linuxshare/dataoutput/dataoutput6  
20/06/27 06:00:54 WARN hdfs.DFSClient: DFSInputStream has been closed already  
20/06/27 06:00:55 WARN hdfs.DFSClient: DFSInputStream has been closed already  
eewhile@ubuntu:/mnt/hgfs/linuxshare/PageRank$ /usr/local/hadoop/bin/hdfs dfs -get /dataoutput7/* /mnt/hgfs/linuxshare/dataoutput/dataoutput7  
20/06/27 06:01:06 WARN hdfs.DFSClient: DFSInputStream has been closed already  
20/06/27 06:01:06 WARN hdfs.DFSClient: DFSInputStream has been closed already  
eewhile@ubuntu:/mnt/hgfs/linuxshare/PageRank$ /usr/local/hadoop/bin/hdfs dfs -get /dataoutput8/* /mnt/hgfs/linuxshare/dataoutput/dataoutput8  
20/06/27 06:01:15 WARN hdfs.DFSClient: DFSInputStream has been closed already  
20/06/27 06:01:16 WARN hdfs.DFSClient: DFSInputStream has been closed already  
eewhile@ubuntu:/mnt/hgfs/linuxshare/PageRank$ /usr/local/hadoop/bin/hdfs dfs -get /dataoutput9/* /mnt/hgfs/linuxshare/dataoutput/dataoutput9  
20/06/27 06:01:31 WARN hdfs.DFSClient: DFSInputStream has been closed already  
20/06/27 06:01:32 WARN hdfs.DFSClient: DFSInputStream has been closed already  
eewhile@ubuntu:/mnt/hgfs/linuxshare/PageRank$ /usr/local/hadoop/bin/hdfs dfs -get /dataoutput10/* /mnt/hgfs/linuxshare/dataoutput/dataoutput10  
20/06/27 06:01:40 WARN hdfs.DFSClient: DFSInputStream has been closed already  
20/06/27 06:01:40 WARN hdfs.DFSClient: DFSInputStream has been closed already  
eewhile@ubuntu:/mnt/hgfs/linuxshare/PageRank$ /usr/local/hadoop/bin/hdfs dfs -get /dataoutput11/* /mnt/hgfs/linuxshare/dataoutput/dataoutput11  
20/06/27 06:01:48 WARN hdfs.DFSClient: DFSInputStream has been closed already  
20/06/27 06:01:49 WARN hdfs.DFSClient: DFSInputStream has been closed already  
eewhile@ubuntu:/mnt/hgfs/linuxshare/PageRank$ /usr/local/hadoop/bin/hdfs dfs -get /dataoutput12/* /mnt/hgfs/linuxshare/dataoutput/dataoutput12  
20/06/27 06:02:00 WARN hdfs.DFSClient: DFSInputStream has been closed already  
20/06/27 06:02:00 WARN hdfs.DFSClient: DFSInputStream has been closed already  
eewhile@ubuntu:/mnt/hgfs/linuxshare/PageRank$ /usr/local/hadoop/bin/hdfs dfs -get /dataoutput13/* /mnt/hgfs/linuxshare/dataoutput/dataoutput13  
20/06/27 06:02:10 WARN hdfs.DFSClient: DFSInputStream has been closed already  
20/06/27 06:02:10 WARN hdfs.DFSClient: DFSInputStream has been closed already  
eewhile@ubuntu:/mnt/hgfs/linuxshare/PageRank$ /usr/local/hadoop/bin/hdfs dfs -get /dataoutput14/* /mnt/hgfs/linuxshare/dataoutput/dataoutput14  
20/06/27 06:02:18 WARN hdfs.DFSClient: DFSInputStream has been closed already  
20/06/27 06:02:19 WARN hdfs.DFSClient: DFSInputStream has been closed already  
eewhile@ubuntu:/mnt/hgfs/linuxshare/PageRank$ /usr/local/hadoop/bin/hdfs dfs -get /dataoutput15/* /mnt/hgfs/linuxshare/dataoutput/dataoutput15  
20/06/27 06:02:27 WARN hdfs.DFSClient: DFSInputStream has been closed already  
20/06/27 06:02:27 WARN hdfs.DFSClient: DFSInputStream has been closed already  
eewhile@ubuntu:/mnt/hgfs/linuxshare/PageRank$ /usr/local/hadoop/bin/hdfs dfs -get /dataoutput16/* /mnt/hgfs/linuxshare/dataoutput/dataoutput16  
20/06/27 06:02:36 WARN hdfs.DFSClient: DFSInputStream has been closed already  
20/06/27 06:02:36 WARN hdfs.DFSClient: DFSInputStream has been closed already
```





```
eewhilegubuntu:/mnt/hgfs/linuxshare/PageRank$ /usr/local/hadoop/bin/hdfs dfs -get /dataoutput17/* /mnt/hgfs/linuxshare/dataoutput/dataoutput17
20/06/27 06:02:46 WARN hdfs.DFSClient: DFSInputStream has been closed already
20/06/27 06:02:47 WARN hdfs.DFSClient: DFSInputStream has been closed already
eewhilegubuntu:/mnt/hgfs/linuxshare/PageRank$ /usr/local/hadoop/bin/hdfs dfs -get /dataoutput18/* /mnt/hgfs/linuxshare/dataoutput/dataoutput18
20/06/27 06:02:54 WARN hdfs.DFSClient: DFSInputStream has been closed already
20/06/27 06:02:54 WARN hdfs.DFSClient: DFSInputStream has been closed already
eewhilegubuntu:/mnt/hgfs/linuxshare/PageRank$ /usr/local/hadoop/bin/hdfs dfs -get /dataoutput19/* /mnt/hgfs/linuxshare/dataoutput/dataoutput19
20/06/27 06:03:02 WARN hdfs.DFSClient: DFSInputStream has been closed already
20/06/27 06:03:02 WARN hdfs.DFSClient: DFSInputStream has been closed already
eewhilegubuntu:/mnt/hgfs/linuxshare/PageRank$ /usr/local/hadoop/bin/hdfs dfs -get /dataoutput20/* /mnt/hgfs/linuxshare/dataoutput/dataoutput20
20/06/27 06:03:11 WARN hdfs.DFSClient: DFSInputStream has been closed already
20/06/27 06:03:11 WARN hdfs.DFSClient: DFSInputStream has been closed already
eewhilegubuntu:/mnt/hgfs/linuxshare/PageRank$ /usr/local/hadoop/bin/hdfs dfs -get /dataoutput21/* /mnt/hgfs/linuxshare/dataoutput/dataoutput21
20/06/27 06:03:18 WARN hdfs.DFSClient: DFSInputStream has been closed already
20/06/27 06:03:19 WARN hdfs.DFSClient: DFSInputStream has been closed already
eewhilegubuntu:/mnt/hgfs/linuxshare/PageRank$ /usr/local/hadoop/bin/hdfs dfs -get /dataoutput22/* /mnt/hgfs/linuxshare/dataoutput/dataoutput22
20/06/27 06:03:27 WARN hdfs.DFSClient: DFSInputStream has been closed already
20/06/27 06:03:27 WARN hdfs.DFSClient: DFSInputStream has been closed already
eewhilegubuntu:/mnt/hgfs/linuxshare/PageRank$ /usr/local/hadoop/bin/hdfs dfs -get /dataoutput23/* /mnt/hgfs/linuxshare/dataoutput/dataoutput23
20/06/27 06:03:36 WARN hdfs.DFSClient: DFSInputStream has been closed already
20/06/27 06:03:36 WARN hdfs.DFSClient: DFSInputStream has been closed already
eewhilegubuntu:/mnt/hgfs/linuxshare/PageRank$ /usr/local/hadoop/bin/hdfs dfs -get /dataoutput24/* /mnt/hgfs/linuxshare/dataoutput/dataoutput24
20/06/27 06:03:45 WARN hdfs.DFSClient: DFSInputStream has been closed already
20/06/27 06:03:45 WARN hdfs.DFSClient: DFSInputStream has been closed already
eewhilegubuntu:/mnt/hgfs/linuxshare/PageRank$ /usr/local/hadoop/bin/hdfs dfs -get /dataoutput25/* /mnt/hgfs/linuxshare/dataoutput/dataoutput25
20/06/27 06:03:57 WARN hdfs.DFSClient: DFSInputStream has been closed already
20/06/27 06:03:57 WARN hdfs.DFSClient: DFSInputStream has been closed already
eewhilegubuntu:/mnt/hgfs/linuxshare/PageRank$ /usr/local/hadoop/bin/hdfs dfs -get /dataoutput26/* /mnt/hgfs/linuxshare/dataoutput/dataoutput26
20/06/27 06:04:07 WARN hdfs.DFSClient: DFSInputStream has been closed already
20/06/27 06:04:07 WARN hdfs.DFSClient: DFSInputStream has been closed already
eewhilegubuntu:/mnt/hgfs/linuxshare/PageRank$ /usr/local/hadoop/bin/hdfs dfs -get /dataoutput27/* /mnt/hgfs/linuxshare/dataoutput/dataoutput27
20/06/27 06:04:16 WARN hdfs.DFSClient: DFSInputStream has been closed already
20/06/27 06:04:16 WARN hdfs.DFSClient: DFSInputStream has been closed already
eewhilegubuntu:/mnt/hgfs/linuxshare/PageRank$ /usr/local/hadoop/bin/hdfs dfs -get /dataoutput28/* /mnt/hgfs/linuxshare/dataoutput/dataoutput28
20/06/27 06:04:24 WARN hdfs.DFSClient: DFSInputStream has been closed already
20/06/27 06:04:24 WARN hdfs.DFSClient: DFSInputStream has been closed already
eewhilegubuntu:/mnt/hgfs/linuxshare/PageRank$ /usr/local/hadoop/bin/hdfs dfs -get /dataoutput29/* /mnt/hgfs/linuxshare/dataoutput/dataoutput29
20/06/27 06:04:32 WARN hdfs.DFSClient: DFSInputStream has been closed already
20/06/27 06:04:32 WARN hdfs.DFSClient: DFSInputStream has been closed already
```


DATA (E:) > ubuntu > linuxshare > dataconvert


名称	修改日期	类型	大小
 _SUCCESS	2020/6/27 20:43	文件	0 KB
 part-r-00000	2020/6/27 20:43	文件	3,120 KB


DATA (E:) > ubuntu > linuxshare > dataoutput


 dataoutput1


 dataoutput2


 dataoutput3


 dataoutput4


 dataoutput5


 dataoutput6


 dataoutput7


 dataoutput8


 dataoutput9


 dataoutput10


 dataoutput11


 dataoutput12


 dataoutput13


 dataoutput14


 dataoutput15


 dataoutput16


 dataoutput17


 dataoutput18


 dataoutput19


 dataoutput20


 dataoutput21


 dataoutput22


 dataoutput23


 dataoutput24

 dataoutput25

 dataoutput26

 dataoutput27

 dataoutput28

 dataoutput29

5 实验心得

一开始选题的时候就纠结了，到底哪个比较适合。

Hadoop的话很早之前就做过了，但是忘得也差不多了哈哈，需要重新来过。

然后k8s的话是刚刚做的手正热，而且思路也很清晰。但是.....做过实验5和实验6后，感觉有点自己劝退自己的感觉，太多的bug卡到自己怀疑人生，单单是环境配置都有很多的问题。所以思前想后还是退出k8s的选题，来Hadoop看看。

可是换过了镜像源，以前的19.04版本ubuntu改成了20.04版本的ubuntu下的linux系统了，也就意味着之前的做的实验打下的基础作废了（可能会疑惑说那用回原来系统不久好了，就是因为19的出现了不可恢复的问题才选择了20阿.....）

所以相当于再做了一遍实验一，以及实验二和四的部分内容，然后测试一遍wordcount成功后，起始就可以验证hadoop是可以正常运行的了。

数据集一共包含75888个节点，以及508837条指向关系。这个数据集也是在数值计算这门课上做实验有用过的，非常符合这次的实验，对于海量数据的处理，使用hdfs储存和mapreduce思想实现。

总的来说还是，累并快乐着，debug让自己更强