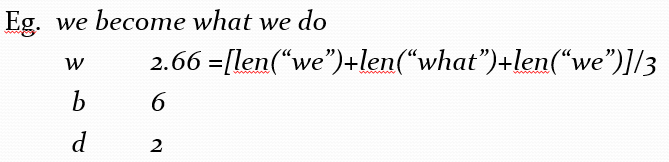
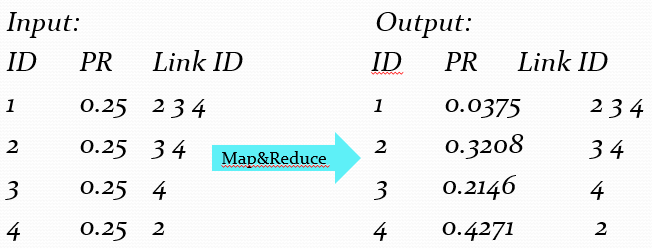
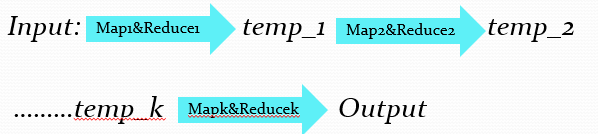
电类工程学导论C实验报告9

518030910406 郑思榕

1. 实验准备
2. 实验环境介绍
3. 环境：在windows系统中使用VirtualBox 5.2.18安装Ubuntu14.04虚拟机，从而在UNIX系统环境下进行本次实验。
4. 语言： python 2.7
5. 工具：本实验主要使用了分布式系统基础架构Hadoop和其提供的编程工具Hadoop Streaming
6. 实验目的
7. 有一篇英语论文，试着写一个mapper.py和一个reducer.py来计算每个单词从“a”到“Z”的平均长度。如下图：
8. 以下是一个关于PageRank的基本算法: https://en.wikipedia.org/wiki/PageRank#Algorithm

https://segmentfault.com/a/1190000000711128 需要几轮迭代才能得出最终的pagerank值。尝试编写自己的map .py和reduce .py来实现这个算法。让它更容易理解,我给下面的例子:α= 0.85，结果如下：

我们将通过下面这种方式得到最终的pagerank值:

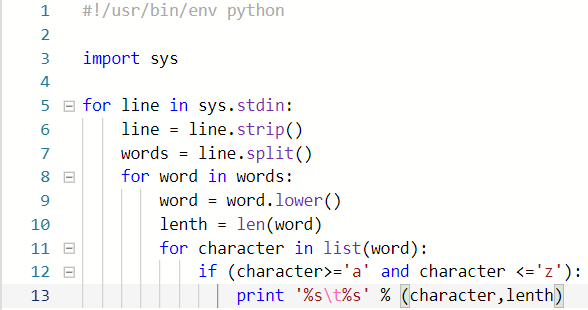
在这个过程中，前一个reducer的输出成为下一个mapper的输入。

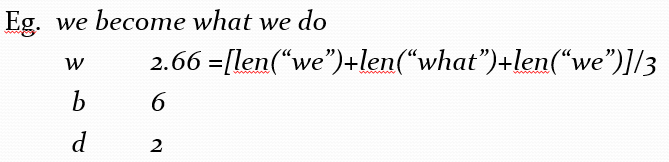
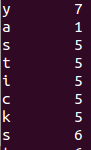
1. 实验原理

Hadoop Streaming是Hadoop提供的编程工具，它允许用户使用任何可执行文件或者脚本文件作为Mapper和Reducer。本实验通过设计mapper.py和reducer.py实现字母统计(ex1)。依据Pagerank的算法思路——即一个网页的排名等于所有链接到该网页的网页的加权排名之和——来设计mapper.py，reducer.py和批处理文件完成pagerank的算法实现。

1. 实验过程
2. ex1

涉及文件：ex1\_input.txt mapper\_ex1.py reducer\_ex1.py ex1\_result

* mapper-ex1.py:

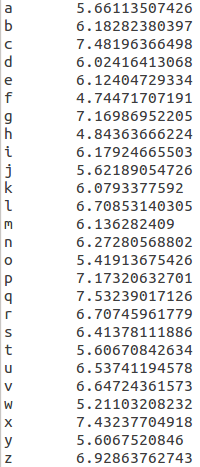
首先设计mapper-ex1.py文件。mapper和reducer会从输入中读取数据，一行一行处理后发送给输出端。所以for line in sys.stdin就是对每行数据的处理。line.strip()用于去除\n，每行里所有单词就存放在line.split()后得到的数组里。对于每个单词，应用word.lower()转为小写，方便后面统计字母数据。由于下图公式可知，需要计算出现某个字母的单词的长度。因此lenth=len(word)。对于word里的每一个字母character，如果是小写字母a~z就打印出来，后面带着该字母所在单词的单词长度。经过mapper-ex1.py后的输出如右图：

* reducer-ex1.py

reducer-ex1.py将mapper-ex1.py的输出作为自己的输入。根据上面的公式，只需分别统计出现a~z的单词总长度/单词个数，即是所求的答案。reducer-ex2.py的完整代码如下：



由于mapper-ex1.py的输出自动根据字母顺序就进行了排序，因此一旦current\_character != character，则代表着该字母统计结束，就可以进行print输出。current\_count指出现当前统计的字母的所有单词的单词长度总和，numofword指单词个数。最后输出时需用float类型转换来输出小数。

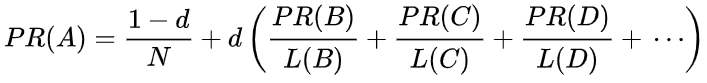
* 结果展示(文件夹ex1\_result)

1. ex2

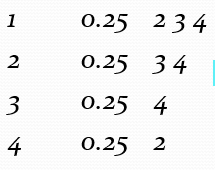
涉及文件：ex2\_input.txt mapper-ex2.py reducer-ex2.py ex2\_result

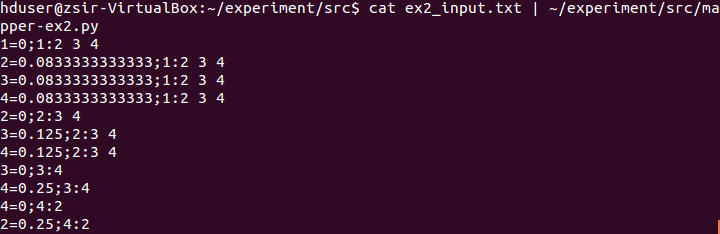
* mapper-ex2.py

根据Wiki中针对Pagerank的算法描述，网页A 的page可以表示为：



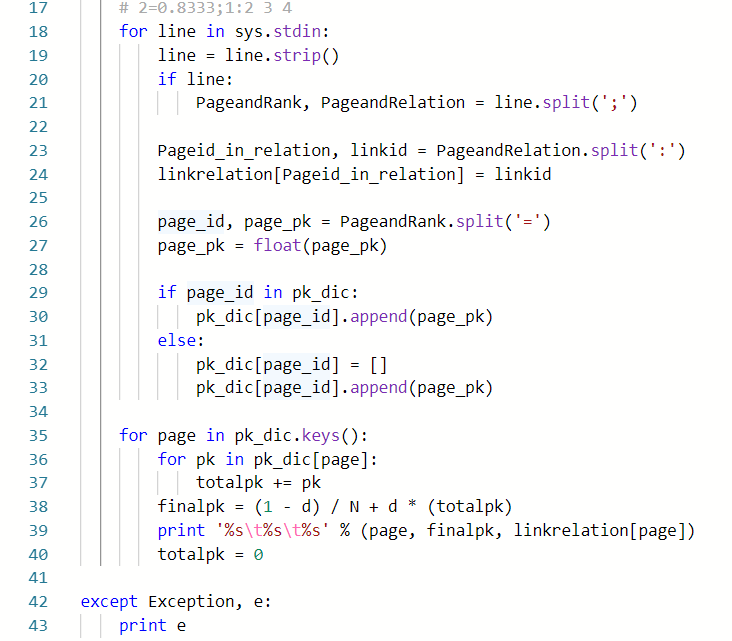
其中d是打开该网页上链接(而不是新开一个网页)的概率，一般设置为0.85。N是所有网页数，本次实验共有4个网页，所以N=4。PR(B)/L(B)是指，在网页B链接到网页A 的前提下，（B的pagerank）/（B中所有的链接数量），如本实验第一次迭代输入 ，则PR(2)/L(2) = 0.25/2=0.125。所以mapper的输出应是所有网页上所有链接的PR(B)/L(B)的值。

首先定义Page类型，包含Page.id和Page.pk便于存储，列表Pagelist用来存放所有的Page。然后进行数据的读取和处理。每行输入数据的格式为，所以line.split(None,2)以空格进行划分，划分两次，第一个元素是page\_id,第二个元素是page\_pk，剩下的是该网页上的所有链接strlinkid。然后进行数据类型转换，page\_pk转换成float类型，并生成Page类型节点存放到Pagelist中。而链接关系也存放到字典linkrelation里。

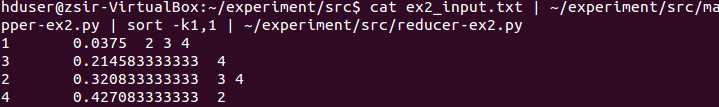
然后对于Pagelist中的网页进行PR(B)/L(B)的计算和输出。观察到输入数据(上图)中，没有网页链接到网页1，所以对于每个网页应先输出PR(B)/L(B)=0的情况，见代码第28行。然后将strlinkid转换成列表listlinkid，对于其中每个page，PR(B)/L(B)=page.pk/float(length)。注意应将链接关系也打印出来，因为reducer-ex2.py需要。最后输出格式类似为：2=0.8333;1:2 3 4，意为“网页2=网页2的PageRank；链接到网页2的父网页：父网页的所有链接”。在本地terminal只运行mapper-ex2.py后的输出如下图：

* reducer-ex2.py



首先对reducer.py的数据进行数据处理。对于每行输入信息进行split( ‘ ; ’ )，第一个元素是网页及由mapper-ex2.py计算得到的PR(B)/L(B)值，第二个元素是链接关系PageandRelaion。对PageandRelation进行split( ‘ : ‘ )拆分，加入到链接关系字典linkrelation，键值对关系为“网页id”=>“该网页上的所有链接”。对PageandRank进行split( ‘ = ‘ )拆分，加入到pageid和PageRank的字典pk\_dic中，键值对关系为“网页id”=>”由mapper得到的该网页所有PR(B)/L(B)值组成的列表”，注意字典pk\_dic中的值是一个列表。

然后对pk\_dic中的所有PR(B)/L(B)进行求和得到totalpk，最后根据PageRank的公式进行finalpk的计算，打印出结果即可。注意输出格式应与mapper-ex2.py的输入格式一致。

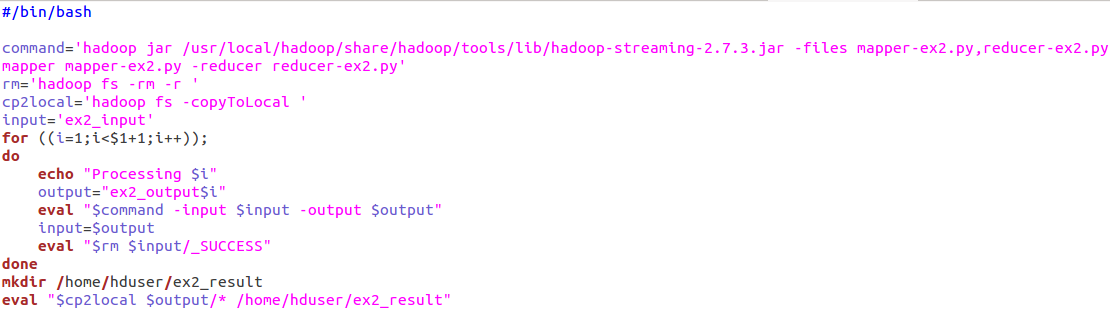
第一次运行mapper-ex2.py和reducer-ex2.py后的输出结果如下：

* batch\_ex2.sh

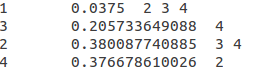
因为需要多次迭代，故创建脚本文件batch\_ex2.sh（如下图）

input = $output每次运行把上一次迭代的输出作为这次的输入，然后每次把输入文件$input删除

最后在本地创建文件夹ex2\_result，将最后一次的输出结果保留在本地



* 结果展示(ex2\_output文件夹)

在经过5次迭代后得到输出如下图，当迭代次数越多，输出趋于一个稳定值。

1. 实验总结
2. 实验概述

本实验通过设计mapper.py、reducer.py和批处理文件来实现字母统计和PageRank算法。

1. 实验心得

本实验我收获良多，主要有以下几点：

1. 学会了Hadoop的基本操作，如copyFromLocal，cat，运行Hadoop streaming等
2. 学会了如何设计mapper.py和reducer.py来实现Hadoop streaming
3. 学会了PageRank算法与如何通过Hadoop streaming实现
4. 学会了如何编写linux批处理文件
5. 遇到的困难及解决方案
6. ex2中在本地运行mapper.py正常输出，在Hadoop云端运行就报错

Solution：通过在mapper.py各个地方print出各种变量以及查看Hadoop stderr，得知是mapper.py自动将输入文件分成两部分，第一部分只包含page 1 和page 2 里的链接，而page 1中有指向page 3 的链接，因此按我以前的算法需要print 出page1的子网页中的链接，即page 3中的链接，但找不到信息因此报错。所以我修改算法，不print出子网页中的链接，反而print出父网页中的链接，即page 1的链接，在reducer.py中再统一收集所有网页的链接关系，因此不报错。

1. 实验创新点
2. 不采用给出的reducer.py的算法结构，不设置变量current\_page和current\_pk。创建自己的reducer.py结构，即创建pk\_dic字典，里面放page\_id=>[page\_pk1, page\_pk2,…]的键值对，最后对列表里的所有page\_pk进行求和。

最后，衷心感谢实验中老师和各位助教的帮助！