电类工程导论(C 类)实验 9 报告

贾萧松 516030910548

一、实验概述

Lab9 中我们学习了 Hadoop Streaming 的相关知识，在本实验中用 Python 来完成我们的 Map 和 Reduce 函数；除此之外，我们还学习如何利用多步 MapReduce（chaining jobs）来完成一项任务，在协调过程中，主要使用 Linux Shell 结合 Hadoop Streaming 完成工作。在 Ex2 中我们还学习了有关 PageRank 的相关知识。

二、实验环境

Ubuntu14.04+jdk1.7+Hadoop2.8.2

三、实验内容

在本次实验中，我们需要针对任务来设计自己的 Map 和 Reduce 函数。在写函数之前，我们首先要认识到：MR 是面向大数据，并行处理提高效率的计算框架，它的主要思想是分而治之。也就要求我们在写函数的时候要充分考虑到 Mapper 和 Reducer 都可能有多个（不同于我们往常的一个文件中的代码只在这个文件中

执行一次），互相之间要可以协同工作而不是导致错误（在本实验中都只需要保证 Mapper 和 Reducer 操作的对象是面向行，而不是面向整个输入文件）。搞清楚这一点，就可以避免很多 bug。

1. Ex1

题目：There is an English essay , try to write a mapper.py and a reducer.py to calculate the average length of each word starting from “A” to “Z”.

在这个实验中，我把任务分为两部分，Mapper 部分负责读取每一行的单词，输出的键值对是：单词的首字母--单词的长度。这样在 Reducer 的部分得到的按键排序的数据也就是字母序了，只需要像 wordcount 样例中那样做就可以了。

a. Mapper 中要做的第一步就是将行内的非字母去除（我用到了 Python 中的 translate 方法）：

```
delchar = string.punctuation+string.digits
identify = string.maketrans("","")

for line in sys.stdin:
    line = line.strip()
    line = line.translate(identify, delchar)
```

b. 然后是 Mapper 的输出：

```
for word in words:
    if(not word.isalpha()):
        continue
    print '%s\t%s' % (word[0].lower(), len(word))
```

c. Reducer 中的代码与 wordcount 类似，只不过需要多记录一个总长度，Reducer 的输出：

```
print '%s\t%s' % (current_char, float(total)/float(current_count))
```

d. 对于脚本文件，我做了一些便于本地调试的变动，这个将在实验拓展部分说明。下面是调用 ex1.sh 的方法，只需要将 mapper.py, reducer.py, 待处理的文本放在 ex1.sh 目录下，然后输入：

```
. ex1.sh pg4300.txt
```

pg4300.txt 为输入文件名

就可以在 ex1.sh 目录下生成 result 文件夹。

部分结果：

```
a   3.31357615894
b   4.97248977895
c   6.53125274725
d   5.4892103391
e   6.14421096003
f   5.05955358861
```

2. Ex2

题目：There is a basic algorithm about PageRank. It'll take several rounds of iterations to work out the final pagerank value. Try to write you own mapper.py and reducer.py to implement this algorithm.

在本实验中，首先要求我们学习 PageRank 算法，作为一种计算网页重要程度的算法，它曾经是 Google 成功的重要因素之一。其采用的是迭代的方式，每次都更新所有网页的 PageRank 值，更新的方式就是将每个网页的 PageRank 值平摊分给它指向的所有网页，每个网页累计所有指向它的网页平摊给它的值作为它该回合的 PageRank 值，直到全部网页的 PageRank 值收敛了或者满足一定的阈值条件就停止。我们要用 MapReduce chaining jobs 来实现这种算法。

2.1 总体思路

我将每次迭代分为两个过程：

第一个是 Mapper 中读取每个网页的 PageRank 和其指向的网页，输出键值对的是：被指向的网页-分配到的 PageRank 值。

第二个是 Reducer 获取排序后的 Mapper 的输出，将每个网页得到的 PageRank 值计算，并输出：网页-PageRank-其指向的网页，也就是 Mapper 需要的输入。这样就完成了一次迭代，此时考虑到 Reducer 需要知道每个网页指向的网页，可以让 Mapper 再输出一种键值对：网页-其指向的网页。也就是说，Mapper 每读取一个网页的信息（一行），输出两个键值对。

2.2 Mapper

a. 首先读取每一个网页的信息（一行）

```
node, pk, out_nodes = line.split('\t', 2)
```

分别对应该网页的 ID ,PageRank,指向的网页。

b. 输出键值对：网页-其指向的网页

```
print(node+'\t'+'n '+' '.join(str(nodes) for nodes in out_nodes))
```

字母 n 作为 Reducer 识别时的标志。

c. 输出键值对：被指向的网页-分配到的 PageRank 值

```
rank = p*pk/len(out_nodes)
for nodes in out_nodes:
        print '%s\t%s' % (nodes, rank)
```

其中 p 为用户选择当前网页上链接的概率（一般设为 0.85）

2.3 Reducer

a. 获取输入

```
node, data = line.split('\t', 1)
```

b. 判断数据类型

```
# This data is about outbound
if(data[0] == 'n'):
    out_nodes = data.split()[1:]
# This data is about pagerank
else:
    current_pk += float(data)
```

若有 n 标志，表示输入的是所指向的网页，否则为分配到的 PageRank。

c. 输出：网页-PageRank-其指向的网页

```
print(str(current_node)+'\t'+str(current_pk+(1-p)/float(N))+'\t' +
      ' '.join(str(nodes) for nodes in out_nodes))
```

其中 PageRank 的计算按照公式即可。

2.4 脚本文件

只需要将 mapper.py, reducer.py, 待处理的文本放在 ex2.sh 目录下，然后输入：

```
. ex2.sh test2.txt
```

test2.txt 为输入文件名

就可以在 ex2.sh 目录下生成 result 文件夹。

a. 考虑到程序的使用的方便性，我实现了脚本文件获取自动获取网页数目 N ：

```
N=$(sed -n '$=' ${basepath}/$1)
let N-=1
```

这里我用了 sed 获取文件行数，减去 1 后就是网页数目

接下来要将 N 传入 Reducer 中，这里 HadoopStreaming 提供了方便的 api：

在 command 中加入 `-cmdenv N=$N` 这个语句表示将向 MR 传入环境变量：键值对"N"-网页数目。

然后在 Reducer 中就可以获得：

```
import os
N = int(os.environ["N"])
```

b. 然后需要考虑 PageRank 迭代算法的收敛问题

这里我采取的方法是判断第二个 Page 的 PageRank 在迭代前后是否小于 0.001 来判断收敛（不选第一个是因为样例中的第一个网页比较特殊，没有其他网页指向它，它的值在第一次迭代后保持不变）。

下面是代码：

```
#last page_rank of page.id 1
last_pk=1
#now the page rank of page.id 1
current_pk=$(sed -n "3,1p" ${basepath}/$1 | awk '{print $2}')
#the difference between them
diff=$(abs $(echo "scale=10; $last_pk - $current_pk" | bc))
cnt=1

while bigger $diff 0.001
do
```

last_pk 储存的是上一次迭代的 PageRank

current_pk 从文件中获取本次迭代的 PageRank，我采用的是获取对应行和列的数据的方法（用到了 sed 和 awk 来获取文件第三行第二列的数）

diff 是上述二者之差。

c. 最后，在迭代结束后，将结果输出到目录下的 result 文件夹里

最终经过 11 次迭代，我的脚本停止，结果如下：

```
1 ID  PK  Link ID
2 1   0.25    2 3 4
3 2   0.25    3 4
4 3   0.25    4
5 4   0.25    2
```

四、问题与解决

1. bash 语言中的问题

本次作业中，我们要写好自己的脚本（主要用于提高效率，减少大量重复命令的输入），而 bash 作为一门脚本语言，它的特点和我们以往学过的语言不大一样，不管是语法上还是操作上，这让我在学习过程中遇到了许多奇怪的问题。

a. “‘`的区别

在我学过的 c++和 Python 中，”和’是没有区别的，而`则是没有用到过。于是我在学习过程中，把它们当作等价，发现 bug 特别多。

经过学习我了解到,在 bash 中：单引号’剥夺了所有字符的特殊含义，单引号"内就变成了单纯的字符；双引号"则对于双引号内的参数替换($)和命令替换(``)是个例外；反引号`是命令替换，命令替换是指 Shell 可以先执行``中的命令，将输出结果暂时保存，在适当的地方输出。

b. 变量的问题

bash 中的变量有个特点，没有变量类型，在运行的时候会被展开成其对应的值（字符串），也就是说变量的值要从输出中获取，这个思想和一般的函数获取值的思想还是不大一样的。

除此之外，变量通过=赋值，=两边不留空格（！！！）。

2. Linux Shell 中的数字处理

在 Shell 中具有最基本的数学计算能力，如可以使用 expr 、let 。但这些都只能处理整形数据。但是可以使用“bc”这个高精度的计算器工具来帮助，另外，也可以在 Bash 中调用“awk”脚本来处理浮点运算。

在 Ex2 中，由于要判断收敛，我学习并写了几个函数：

```
abs() { echo ${1#-};}
```

这个是学到的比较巧妙的求绝对值的实现。

```
$(echo "scale=10; $last_pk - $current_pk" | bc))
```

这个是浮点数作差的实现，代码意思是精度为 10 位使用 bc 计算器给二者作差。

```
bigger(){
    awk -v n1="$1" -v n2="$2" 'BEGIN {if (n1+0>n2+0) exit 0; exit 1}'
}
```

由于 Shell 中也没有浮点数比较函数，于是我使用 awk 实现了一个大于函数

从解决如此简单浮点数问题所需要的工作量，可以看出 Linux 内核对浮点数的支持是比较差的。至于原因，经过查询得知：

浮点的编码跟整数编码是不一样的，计算时需要专门的寄存器和浮点计算单元来处理，一个浮点运算指令使用的 CPU 周期也更长，因此对于内核来说就会想尽量回避浮点数运算，譬如说浮点数经过定点整数转换后进行运算，效率会高很多，即使 CPU 带有浮点数运算部件，一般内核还是要避免直接进行浮点数运算，因为这些部件有可能被用户进程占用了，内核要判断这些浮点数部件是否被占用，保护现场，然后用浮点运算部件计算结果，恢复现场，开销会很大。而对内核而言，效率是至关重要的。

五、实验拓展

1. hadoop 集群调试脚本的配置

从提供的样例脚本中，我学到了如何把需要输入的一系列命令集成的到一起，减少了每次调用都需要输一遍一系列相同代码的麻烦，下面是我的思路：

a. 首先，可以每次一打开脚本，自动完成 tempinput,tempoutput 文件夹的设置：

```
eval "hadoop fs -rm -r tempoutput"
eval "hadoop fs -rm -r tempinput"
eval "hadoop fs -mkdir tempinput"
eval "hadoop fs -copyFromLocal ${basepath}/$1 tempinput/test1.txt"
```

b. 这里注意到 basepath 变量，这代表.sh 脚本所在目录的绝对路径，只要将文件放在脚本所在目录，就可以只输入文件名就可以上传。

下面是 bashpath 的获取：

```
basepath=$(cd $(dirname ${BASH_SOURCE[0]}); pwd )
```

原理: BASH_SOURCE[0] 取得当前执行的 shell 文件所在的路径及文件名

dir_name 取给定路径的目录部分

pwd 查看”当前工作目录“的完整路径

这样就获取了当前执行脚本的绝对路径。

c. 然后，考虑到 mapper 和 reducer 是要频繁修改的，可不可以用本地文件呢？事实上，streaming 中有这个方法：

```
-file ${basepath}/mapper.py -file ${basepath}/reducer.py
```

在 command 中把-files 改成-file，就可以方便的使用目录下的文件了。

d. 最后是输出：

```
eval "$command -input tempinput/* -output tempoutput"
eval "rm -r ${basepath}/result"
eval "mkdir ${basepath}/result"
eval "$cp tempoutput/* ${basepath}/result"
```

经过这一番设计，每次不管是修改了 mapper reducer 还算是想更改输入文件，都只需要一行代码就可以完成在 hadoop 集群测试，并在本地得到结果，这个脚本大大提高了我调试的效率。

六、总结

在本实验中，我们使用到了 Hadoop Streaming 来完成对指定任务的 MR 工作，我体会到的优点有不少：可以用 Python 写 Map Reduce 函数；容易进行单机调试；含有丰富的参数设置来完成想要的功能而不用 java 语言修改，是一个比较便捷的 Hadoop 集群框架。

除此之外，我还学到了不少 bash 语言的知识，感受了脚本在程序调试过程中的好用之处。

最后，衷心感谢老师和助教们的精心准备和辛勤付出，你们整理完善的 ppt 和到位的答疑大大提高了我学习 Hadoop Streaming 和 bash 语言的效率，感谢~

贾萧松 516030910548

2017.11.19