

Report of Lab5 and Lab6

Yuxiang Chen 5110309783

December 12, 2012

Contents

1 The Purpose of This Experiment and My Preparation	1
2 The Main Part of the Experiment	1
2.1 The answer to Mini Problems	2
2.2 Calculate the Average Length of a Certain Initial Letter	2
2.3 Getting the PageRank	5
3 The Problems I Met and My Thoughts	9

1 The Purpose of This Experiment and My Preparation

Needless to say, this experiment is mainly based on the Linux, another currently-using operating system. And we have to use hadoop to finish the experiment and learn a method called mapreducing. This method consists of, as you can see from its name, two parts – map and reduce. To map is to set up an index of the inputs, and to reduce is just to make a synthesis of the output from the map method. And in this way, we can solve some certain problems.

The second part enables us to do some hadoop streaming jobs – which means we can process more files. Of course, this part also requires the mapreducing method.

2 The Main Part of the Experiment

Well, as an introduction of hadoop and probably Linux, this experiment is not very difficult, except for some mistakes we have engaged in when establishing the environment of hadoop. As I have met so many problems when trying to set up my own hadoop, so I decided to use the visual machine the TA had given us. Of course, there were still some mistakes even though I imitate the steps given in the demo ppt. But to make this report more based on the experiment itself, I will not list all these problems here. And the following are my result of the questions in these ppts.

2.1 The answer to Mini Problems

As the first ppt just intends to tell us some basic knowledges of hadoop, so the first part, in my opinion, doesn't need codes written by us. And the only thing we have to do is to get the result of the first problem, copying the command lines. The second mini exercise is just giving us a chance to try some sets of numbers to get the final pi we want. Here are the results of the problems:

2	10	3.80000000000000000000	34.587
5	10	3.28000000000000000000	45.487
10	10	3.20000000000000000000	54.551
2	100	3.12000000000000000000	35.311
10	100	3.14800000000000000000	56.49
10	100000	3.14155200000000000000	53.555
10	500000	3.14157280000000000000	55.463
100	5000	3.14159200000000000000	329.724

Figure 1: total result



Figure 2: the result of 2 and 10

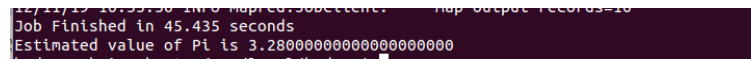


Figure 3: the result of 5 and 10

And the last three pictures are my results of the second Mini problem, and the last result attain the standard TA has given in the ppt files. I try these data just to learn more about the mapreducing method, and I find that the first number, which means the number of times of the experiment, is much more stronger than the second number, which means the number of experiments each time. And this means the more the times of experiments, the bigger the total time of calculating pi.

2.2 Calculate the Average Length of a Certain Initial Letter

This part is a little difficult because we are new at this operating system, so there are a lot of trouble at this experiment. However, I managed to solve it by trying so many times to understand what each command line means. And for this experiment, what we have to do is firstly make all the initial letters in their small form and then use the mapreduce method to get the total length of each initial letter, of course, then we can get the average length easily. Here are my mapper codes:

```
12/11/19 10:37:53 INFO MapredJobClient: Map Output Records=20  
Job Finished in 53.764 seconds  
Estimated value of Pi is 3.200000000000000000000000
```

Figure 4: the result of 10 and 10

```
Job Finished in 32.51 seconds  
Estimated value of Pi is 3.120000000000000000000000
```

Figure 5: the result of 2 and 100

```
Job Finished in 55.548 seconds  
Estimated value of Pi is 3.148000000000000000000000
```

Figure 6: the result of 10 and 100

```
12/11/19 10:41:42 INFO MapredJobClient: Map Output Records=20  
Job Finished in 53.45 seconds  
Estimated value of Pi is 3.141520000000000000000000
```

Figure 7: the result of 10 and 100000

```
12/11/19 10:45:40 INFO MapredJobClient: Map Output Records=20  
Job Finished in 55.463 seconds  
Estimated value of Pi is 3.141572800000000000000000
```

Figure 8: the result of 10 and 500000

```
12/11/19 10:50:02 INFO MapredJobClient: Map Output Records=200  
Job Finished in 331.074 seconds  
Estimated value of Pi is 3.141592000000000000000000
```

Figure 9: the result of 100 and 5000

```

1 |#!/usr/bin/env python
2 |
3 |import sys
4 |import string
5 |import re
6 |
7 |for line in sys.stdin:
8 |    line = line.strip()
9 |    line = re.sub(r'^a-zA-Z]', '_', line)
10 |    words = line.split()
11 |    for word in words:
12 |        print '%s\t%s' % (string.lower(word), len(word))

```

In this part, I only eliminate all the punctuation in the essay and then split them up. And the following are my reduce method:

```

1 |#!/usr/bin/env python
2 |
3 |from operator import itemgetter
4 |import sys
5 |
6 |current_word = None
7 |current_count = 0
8 |word = None
9 |word_num=0
10 |
11 |for line in sys.stdin:
12 |    line = line.strip()
13 |    word, count = line.split('\t', 1)
14 |    try:
15 |        count = int(count)
16 |    except ValueError:
17 |        continue
18 |
19 |    if current_word == word[0]:
20 |        current_count += count
21 |        word_num+=1
22 |    else:
23 |        if current_word:
24 |            average=float(current_count)/word_num
25 |            print '%s\t%f' % (current_word, average)
26 |            current_count = count
27 |            current_word = word[0]
28 |            word_num=1
29 |
30 |    if current_word == word[0]:
31 |        average=float(current_count)/word_num

```

32 | `print '%s\t%f' % (current_word, average)`

This part, of course, count the total length of each initial letter and also count the number of words initialed of each letter, then we can get the result we want easily.

Here are some pictures of my results:

```
hadoop@ieee-desktop:/usr/local/hadoop$ bin/hadoop jar contrib/streaming/hadoop-0.20.2-streaming.jar -file /home/experiments/src/mapper.py -mapper /home/experiments/src/mapper.py -file /home/experiments/src/reducer.py -reducer /home/experiments/src/reducer.py -input /user/hadoop/tempinput -output /user/hadoop/tempoutput packageJobJar: [/home/experiments/src/mapper.py, /home/experiments/src/reducer.py, /usr/local/hadoop/tmp/dir/hadoop-hadoop/hadoop-unjar3887532198548633564/] [] /tmp/streamjob2472336066592746128.jar tmpDir=null
12/11/17 19:23:12 INFO mapred.FileInputFormat: Total input paths to process : 1
12/11/17 19:23:13 INFO streaming.StreamJob: getLocalDirs(): [/usr/local/hadoop/tmp/dir/hadoop-hadoop/mapred/local]
12/11/17 19:23:13 INFO streaming.StreamJob: Running job: job_201211171900_0009
12/11/17 19:23:13 INFO streaming.StreamJob: To kill this job, run:
12/11/17 19:23:13 INFO streaming.StreamJob: /usr/local/hadoop/bin/./bin/hadoop job -Dmapred.job.tracker=localhost:9001 -kill job_201211171900_0009
12/11/17 19:23:13 INFO streaming.StreamJob: Tracking URL: http://localhost:50030/jobdetails.jsp?jobid=job_201211171900_0009
12/11/17 19:23:14 INFO streaming.StreamJob: map 0% reduce 0%
12/11/17 19:23:25 INFO streaming.StreamJob: map 100% reduce 0%
12/11/17 19:23:37 INFO streaming.StreamJob: map 100% reduce 100%
12/11/17 19:23:40 INFO streaming.StreamJob: Job complete: job_201211171900_0009
12/11/17 19:23:40 INFO streaming.StreamJob: Output: /user/hadoop/tempoutput
```

Figure 10: the process and command line

```
hadoop@ieee-desktop:/usr/local/hadoop$ bin/hadoop fs -ls /user/hadoop/tempoutput
Found 2 items
drwxr-xr-x  - hadoop supergroup          0 2012-11-17 19:23 /user/hadoop/tempoutput/logs
-rw-r--r--  2 hadoop supergroup        253 2012-11-17 19:23 /user/hadoop/tempoutput/part-00000
```

Figure 11: the catalog of the output fold

2.3 Getting the PageRank

This part requires us to get the pagerank of all nodes in a topological graph. And to accomplish this, we need to pass two pieces of information : the links between those nodes and the weight they get from their linking nodes. And what we need to pay attention to is we have to make sure the input and the output are in the same form, since we will have to make the output into a new input in the chaining jobs. And I have been trying for a long time to get a method, with which I can get the relationship(or link-relation) among those nodes in 'mapper' and pass them to 'reducer'. But after so many times of trying, I failed. So I just store their relations as a constant variable in 'reducer'. And we can modify the numbers of nodes, their relationships and alpha(represented with 'a' in my code) in 'reducer'. I decide to store them this way because I think if we have a large number times of experiments, say, 100,000, than if we write a code to

```
hadoop@ieee-desktop:/usr/local/hadoop$ bin/hadoop fs -cat /user/hadoop/tempoutpu
t/part-00000
a      3.215064
b      5.005002
c      6.336456
d      5.259408
e      5.896983
f      5.049560
g      5.429021
h      3.719976
i      2.701811
j      4.413710
k      5.212580
l      5.100660
m      4.728350
n      4.385548
o      2.773585
p      6.278085
q      6.062300
r      5.975724
s      4.923195
t      3.758199
u      4.702398
v      5.749630
w      4.513275
x      3.333333
y      3.615361
z      3.918919
```

Figure 12: the result of a part of essay 4300

```
hadoop@ieee-desktop:/usr/local/hadoop$ bin/hadoop fs -cat /user/hadoop/tempoutpu
t/part-00000
a      3.353702
b      4.193546
c      6.422313
d      5.554936
e      5.363234
f      5.189479
g      5.499076
h      4.171323
i      3.051732
j      5.587549
k      4.855204
l      5.177431
m      5.039734
n      4.054763
o      2.896752
p      6.443533
q      5.738971
r      6.481317
s      5.147164
t      3.539835
u      5.076759
v      5.699776
w      4.525913
x      3.571046
y      3.544376
z      4.606667
```

Figure 13: the result of a part of essay 5000

```

hadoop@ieee-desktop:/usr/local/hadoop$ bin/hadoop fs -cat /user/hadoop/tempoutpu
t/part-00000
a      3.605319
b      4.441879
c      7.075480
d      6.750173
e      6.933769
f      4.922384
g      5.701070
h      4.481759
i      3.818675
j      4.294944
k      5.181443
l      5.128402
m      5.439607
n      4.714019
o      2.793040
p      7.212383
q      6.931818
r      6.799242
s      5.469892
t      3.643482
u      5.399826
v      5.901840
w      4.237485
x      1.000000
y      4.386965
z      5.731343

```

Figure 14: the result of a part of essay 20417

get the relationship of the nodes in 'mapper', then it will be executed for also 100,000 times. So if I just store it in the reducer, it can be treated as a strategy of 'getting time in the price of space'.

And here are my codes of mapper:

```

1  #!/usr/bin/env python
2
3  import sys
4
5  for line in sys.stdin:
6      line = line.strip()
7      elements = line.split('\t')
8      linknodes=eval(elements[2])
9      length=len(linknodes)
10     node_id=elements[0].strip()
11     pr=elements[1]
12     link=list()
13     just_for_bug=list()
14     just_for_bug.append(node_id)
15     just_for_bug.append(0)
16     print node_id, '\t', just_for_bug
17     for l in linknodes:
18         average=float(pr)/length
19         assemble=list()
20         assemble.append(node_id)
21         assemble.append(average)
22     print l, '\t', assemble

```

In the former part, I just pass the information of every node into the reducer from the input, and the reducer is as follows:

```
1  #!/usr/bin/env python
2
3  from operator import itemgetter
4  import sys
5
6  current_node = None
7  current_pr = 0
8  node = None
9  a=0.85
10 num=4
11 flag=0
12 total_link=[[2,3,4],[3,4],[4],[2]]
13
14 # input comes from STDIN
15 for line in sys.stdin:
16     # remove leading and trailing whitespace
17     line = line.strip()
18
19     # parse the input we got from mapper.py
20     line= line.split('\t')
21     node=line[0]
22     add_list=line[1]
23     add_list=eval(add_list)
24     add_pr=add_list[1]
25
26     try:
27         add_pr = float(add_pr)
28     except ValueError:
29         continue
30
31     if current_node == node:
32         current_pr += add_pr
33     else:
34         if current_node:
35             final_pr=a*current_pr+(1-a)/num
36             final_pr=float('%4f' % final_pr)
37             print current_node, '\t', final_pr, '\t', total_link[flag]
38             flag+=1
39         current_node = node
40         current_pr = add_pr
41
42 # do not forget to output the last word if needed!
43 if current_node == node:
```



```

44 |         final_pr=a*current_pr+(1-a)/num
45 |         final_pr=float( '%.4f' % final_pr )
46 |         print current_node , '\t' , final_pr , '\t' , total_link [ flag ]

```

In this part, I just calculate the rank of each node and then given them the same form as the input file and then output them. And to make the form tidier, I make the result shown with four numbers after the point in the decimal. Meanwhile, to get the result we want being output in a file, I change the last sentence of the 'batch for pageRank.sh', which is as follows:

```

1 | #!/bin/bash
2 |
3 | command='bin/hadoop_jar_ contrib/streaming/hadoop-0.20.2-streaming.jar_ file_/hom
4 | mv='bin/hadoop_fs_ mv_'
5 | rm='bin/hadoop_fs_ rmr_'
6 | cp='bin/hadoop_fs_ copyToLocal_'
7 | ct='bin/hadoop_fs_ cat_'
8 | function func()
9 | {
10 | for (( i=1;i<${1+1};i++));
11 | do
12 |     echo "Processing_$i"
13 |     eval "$command_ input_$2/*_ output_tempoutput"
14 |     eval "rm_$2"
15 |     eval "$mv_/user/hadoop/tempoutput_$2"
16 | done
17 | }
18 | func $1 $2
19 | #eval "$cp $2/* /home/experiments/output/result.txt"
20 | eval "$ct_$2/part-*>_/home/hadoop/result.txt"

```

Thus we can find the final result at 'result.txt'. As I have said, to save time, I just let the code run for three times to get a demo result because the slow visual machine...

3 The Problems I Met and My Thoughts

In the experiment, there're just too many problems. And I'll only take the typical ones.

First, at the 'initial-letter-length finding' part, I have some trouble eliminating the punctuation of the essay, then I searched for it and solve it using the regular expression:

```

1 |         line = line.strip()
2 |         line = re.sub(r'^[a-zA-Z]', '', line)

```

```
hadoop@ieee-desktop: /usr/local/hadoop
文件(F) 编辑(E) 查看(V) 终端(T) 帮助(H)
tmpinput
Processing 1
packageJobJar: [/home/experiments/src/mapper.py, /home/experiments/src/reducer.p
y, /usr/local/hadoop/tmp/dir/hadoop-hadoop/hadoop-unjar602122464797658099/] [] /
tmp/streamjob5348916908319663823.jar tmpDir=null
12/11/17 19:04:29 INFO mapred.FileInputFormat: Total input paths to process : 1
12/11/17 19:04:30 INFO streaming.StreamJob: getLocalDirs(): [/usr/local/hadoop/t
mp/dir/hadoop-hadoop/mapred/local]
12/11/17 19:04:30 INFO streaming.StreamJob: Running job: job_201211171900_0001
12/11/17 19:04:30 INFO streaming.StreamJob: To kill this job, run:
12/11/17 19:04:30 INFO streaming.StreamJob: /usr/local/hadoop/bin/../bin/hadoop
job -Dmapred.job.tracker=localhost:9001 -kill job_201211171900_0001
12/11/17 19:04:30 INFO streaming.StreamJob: Tracking URL: http://localhost:50030
/jobdetails.jsp?jobid=job_201211171900_0001
12/11/17 19:04:31 INFO streaming.StreamJob: map 0% reduce 0%
12/11/17 19:04:55 INFO streaming.StreamJob: map 100% reduce 0%
12/11/17 19:05:07 INFO streaming.StreamJob: map 100% reduce 100%
12/11/17 19:05:10 INFO streaming.StreamJob: Job complete: job_201211171900_0001
12/11/17 19:05:10 INFO streaming.StreamJob: Output: tempoutput
Deleted hdfs://localhost:9000/user/hadoop/tmpinput
Processing 2
packageJobJar: [/home/experiments/src/mapper.py, /home/experiments/src/reducer.p
y, /usr/local/hadoop/tmp/dir/hadoop-hadoop/hadoop-unjar6115889282910932066/] []
/tmp/streamjob7973848118772075356.jar tmpDir=null
```

Figure 15: Processing 1

```
hadoop@ieee-desktop: /usr/local/hadoop
文件(F) 编辑(E) 查看(V) 终端(T) 帮助(H)
12/11/17 19:05:10 INFO streaming.StreamJob: Output: tempoutput
Deleted hdfs://localhost:9000/user/hadoop/tmpinput
Processing 2
packageJobJar: [/home/experiments/src/mapper.py, /home/experiments/src/reducer.p
y, /usr/local/hadoop/tmp/dir/hadoop-hadoop/hadoop-unjar6115889282910932066/] []
/tmp/streamjob7973848118772075356.jar tmpDir=null
12/11/17 19:05:15 INFO mapred.FileInputFormat: Total input paths to process : 1
12/11/17 19:05:15 INFO streaming.StreamJob: getLocalDirs(): [/usr/local/hadoop/t
mp/dir/hadoop-hadoop/mapred/local]
12/11/17 19:05:15 INFO streaming.StreamJob: Running job: job_201211171900_0002
12/11/17 19:05:15 INFO streaming.StreamJob: To kill this job, run:
12/11/17 19:05:15 INFO streaming.StreamJob: /usr/local/hadoop/bin/../bin/hadoop
job -Dmapred.job.tracker=localhost:9001 -kill job_201211171900_0002
12/11/17 19:05:15 INFO streaming.StreamJob: Tracking URL: http://localhost:50030
/jobdetails.jsp?jobid=job_201211171900_0002
12/11/17 19:05:16 INFO streaming.StreamJob: map 0% reduce 0%
12/11/17 19:05:25 INFO streaming.StreamJob: map 100% reduce 0%
12/11/17 19:05:38 INFO streaming.StreamJob: map 100% reduce 100%
12/11/17 19:05:43 INFO streaming.StreamJob: Job complete: job_201211171900_0002
12/11/17 19:05:43 INFO streaming.StreamJob: Output: tempoutput
Deleted hdfs://localhost:9000/user/hadoop/tmpinput
Processing 3
packageJobJar: [/home/experiments/src/mapper.py, /home/experiments/src/reducer.p
y, /usr/local/hadoop/tmp/dir/hadoop-hadoop/hadoop-unjar105899719768377528/] []
```

Figure 16: Processing 2


```
1 0.0375 [2, 3, 4]
2 1.0508 [3, 4]
3 0.6451 [4]
4 1.1089 [2]
```

Figure 19: the outcome of the first round

```
1 0.0375 [2, 3, 4]
2 1.1708 [3, 4]
3 0.7458 [4]
4 1.5958 [2]
```

Figure 20: the outcome of the third round

And then at the 'pagerank' part, I try to give the decimals four numbers after the point, and here is how I solve it at last:

```
1 | final_pr=float( '%.4f' % final_pr )
```

Of course, I said earlier that I can't pass the relationship to 'reducer' at first in the 'pagerank' part, and I also give my solution to the problem in the former section.

I think the mapreduce method is just like building lots of threads and giving each thread a certain job, which makes the problem easier and faster. Meanwhile, I'm pretty interested in the pagerank calculating method, because I think with which, we can make our searching engine more organized when giving out the searching results. And we can also give every page a suitable position in the final output page in terms of their ranks.