上机实验八 & 实验九
实验报告

刘瀚文
517030910294

2018 年 11 月 20 日

目录

Part I

实验八

Chapter 1

实验准备

1.1 环境配置

1.1.1 hadoop 安装

主要的难点就是 hadoop 的配置和安装，完成了环境的配置就几乎要成功了。

Prerequiste

Install Ubuntu

Create Hadoop User

Setup SSH Certification

Install Java and ssh-server

Download Hadoop 2.2.0

Setup Hadoop Enironment

Configure Hadoop

Format Namenode

Start Hadoop Service

Stop Services

1.2 背景知识

1.2.1 Brief Introduction for Hadoop

Hadoop 简介

Formally speaking, Hadoop is an open source framework for writing and running distributed applications that process large amounts of data.

A Hadoop cluster has many parallel machines that store and process large data sets. Client computers send jobs into this computer cloud and obtain results.

Hadoop 优点

Accessible Hadoop runs on large clusters of commodity machines or on cloud computing services such as Amazon' s Elastic Compute Cloud (EC2).

Robust Because it is intended to run on commodity hardware, Hadoop is architected with the assumption of frequent hardware malfunctions. It can gracefully handle most such failures.

Scalable Hadoop scales linearly to handle larger data by adding more nodes to the cluster.

Simple Hadoop allows users to quickly write efficient parallel code.

1.2.2 MapReduce Overview

Characteristic

Automatic parallelization & distribution

Fault-tolerant

Provides status and monitoring tools

Clean abstraction for programmers

1.2.3 Map

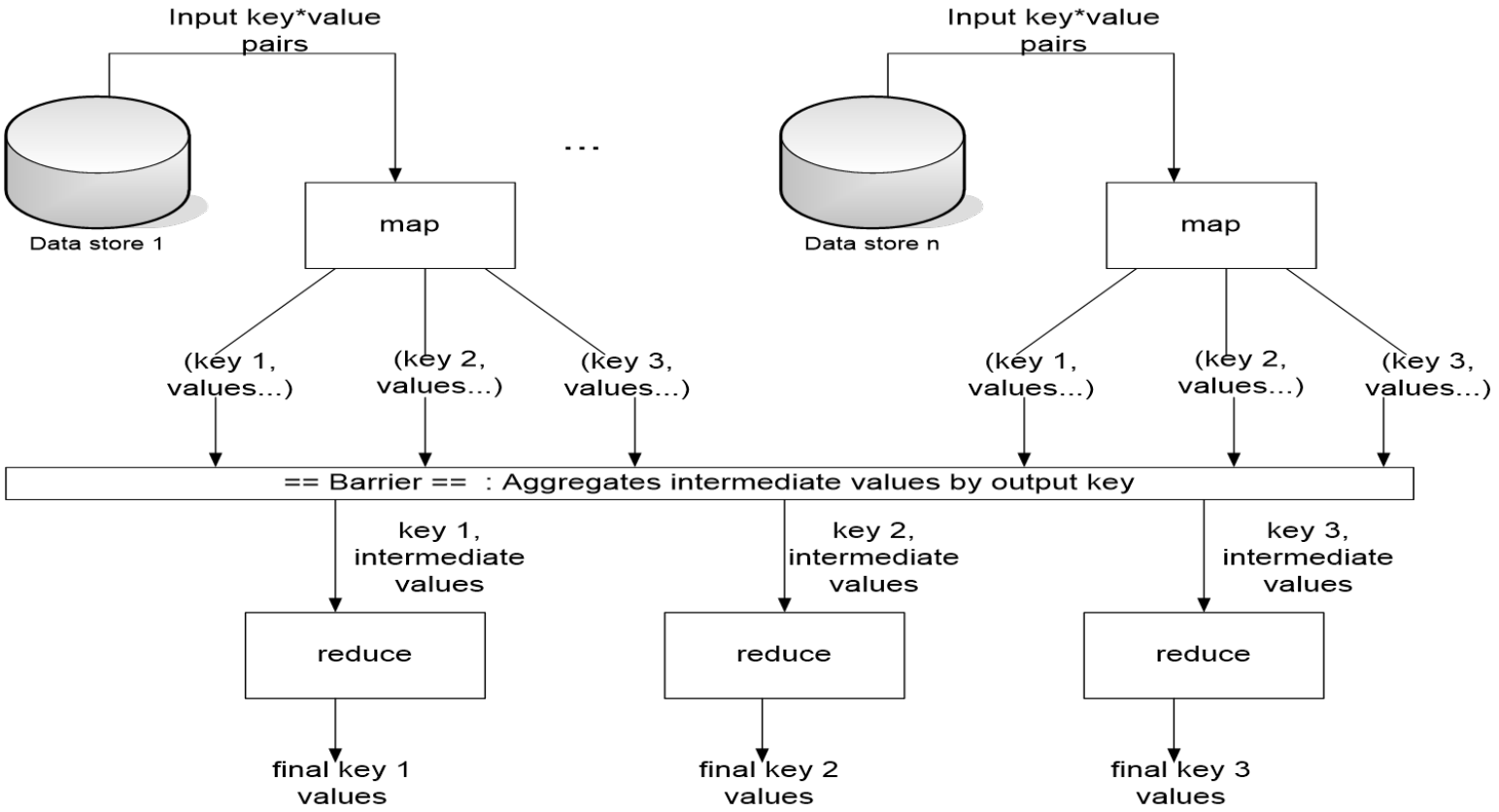
Records from the data source (lines out of files, rows of a database, etc) are fed into the map function as key*value pairs: e.g., (filename, line).

map() produces one or more intermediate values along with an output key from the input.

After the map phase is over, all the intermediate values for a given output key are combined together into a list

reduce() combines those intermediate values into one or more final values for that same output key.(in practice, usually only one final value per key)

1.2.4 Architecture



1.3 简单的自我尝试

WordCount on Hadoop
比较简单

Chapter 2

Mini Exercise

2.1 Exercise 1

2.1.1 要求

Practise using basic hadoop command and fill in the following table

2.1.2 实验过程

Start hadoop

Use command to compute pi <nMaps> is the number of mapper jobs and <nSamples> is the number of samples

```
hadoop jar /usr/local/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-
                                    examples-2.2.0.jar pi  <nMaps>  <
                                    nSamples>
```

Number of Maps	Number of samples	Time(s)	π
2	10	15.088 seconds	3.800
5	10	17.118	3.2800
10	10	18.179	3.200
2	100	14.996	3.1200
10	100	28.183	3.14800
100	100’0000	126.359 seconds	3.1415925600
100	1000’0000	133.294 seconds	3.14159273600
100	1’0000’0000	154.344 seconds	3.141592649200
1000	1’0000’0000	1417.978 seconds	3.1415926557200
2000	1’0000’0000	2628.074 seconds	3.1415926575600

随着测试数量的上升，时间和精确程度都在上升！

```
18/11/08 18:46:04 INFO mapreduce.Job: Job job_1541668829670_0009 completed succe
ssfully
18/11/08 18:46:04 INFO mapreduce.Job: Counters: 43
        File System Counters
                FILE: Number of bytes read=22006
                FILE: Number of bytes written=79680381
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=267890
                HDFS: Number of bytes written=215
                HDFS: Number of read operations=4003
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=3
        Job Counters
                Launched map tasks=1000
                Launched reduce tasks=1
                Data-local map tasks=1000
                Total time spent by all maps in occupied slots (ms)=5937503
                Total time spent by all reduces in occupied slots (ms)=1167148
        Map-Reduce Framework
                Map input records=1000
                Map output records=2000
                Map output bytes=18000
                Map output materialized bytes=28000
                Input split bytes=149890
                Combine input records=0
                Combine output records=0
                Reduce input groups=2
                Reduce shuffle bytes=28000
                Reduce input records=2000
                Reduce output records=0
                Spilled Records=4000
                Shuffled Maps =1000
                Failed Shuffles=0
                Merged Map outputs=1000
                GC time elapsed (ms)=72621
                CPU time spent (ms)=2787850
                Physical memory (bytes) snapshot=261899935744
                Virtual memory (bytes) snapshot=844577435648
                Total committed heap usage (bytes)=204668928000
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=118000
        File Output Format Counters
                Bytes Written=97
Job Finished in 1417.978 seconds
Estimated value of Pi is 3.14159265572000000000
```

展示 1000(Map) * 1'0000'0000(Samples) 的结果!

Get the result

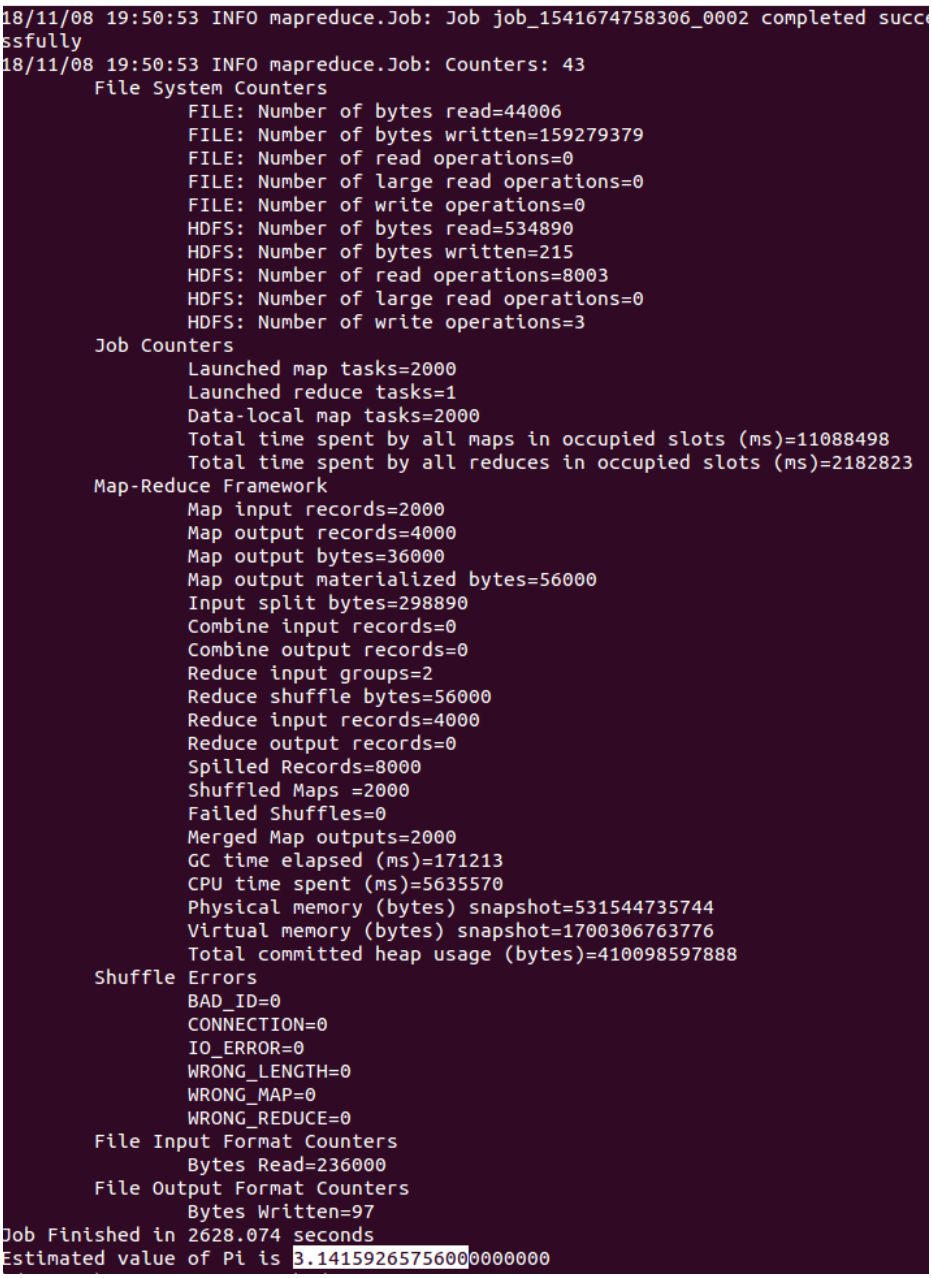
2.2 Exercise 2

2.2.1 要求

Work out a solution to make the computed pi approximate the 5th digit after the decimal dot correctly.

2.2.2 实验过程

实验过程和 1 类似，为了进行更精确的计算，调大了内存 (6G -> 7.5G)，分配了更多处理器内核 (6 -> 8)，观察是否有更快的结果输出。



展示 2000(Map) * 1'0000'0000(Samples) 的结果！

结果是：3.1415926575600 和 pi 的近似值 (3.1415926535898) 比较接近，满足练习要求！

Part II

实验总结

这次实验主要是学习新的 Hadoop，很有趣！

在配置环境的过程中，出现了一些奇妙的问题。通过使用 VMVare 的快照和自己 DeBUG 的过程学习到了新的 Ubuntu 的知识，有所进步！

期待在下一次实验中学习更多的知识！

Part III

实验 9

原本已经写好了实验 8 的实验报告，然后上课前才得知实验 8 和实验 9 一起交，所以实验 9 的报告就接在实验 8 的报告后面啦。

Chapter 3

实验准备

3.1 背景知识

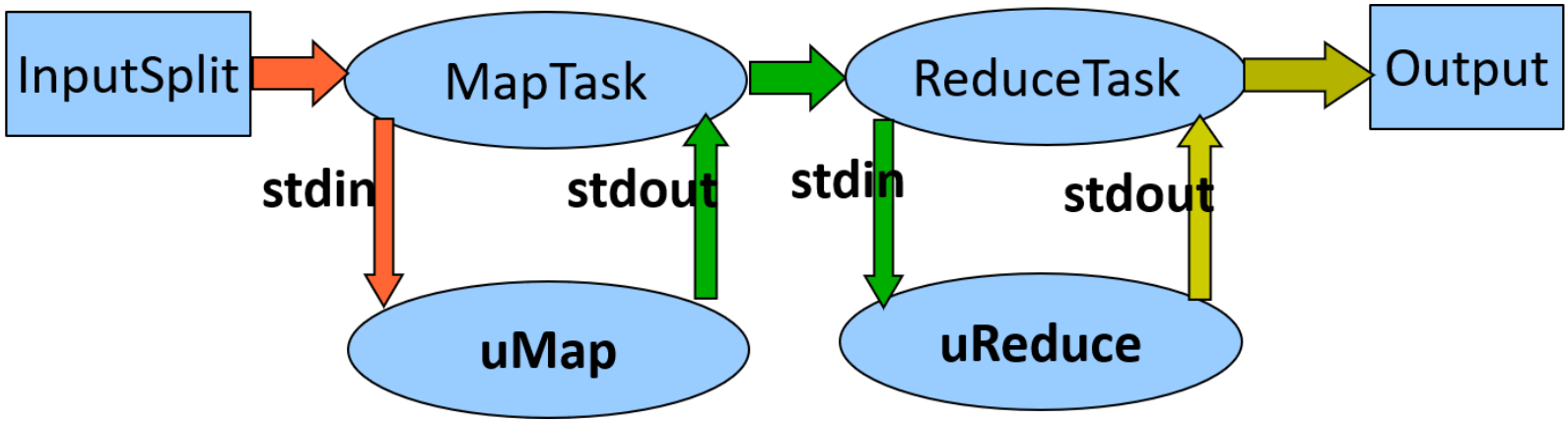
3.1.1 HDFS Concepts

Very large in this context means files that are hundreds of megabytes, gigabytes, or terabytes in size. There are Hadoop clusters running today that store petabytes of data.

Streaming data access HDFS is built around the idea that the most efficient data processing pattern is a write-once, read-many-times pattern.

Commodity hardware Hadoop doesn't require expensive, highly reliable hardware to run on.

3.1.2 Hadoop Streaming



Chapter 4

实验操作

4.1 exercise1

实验一非常简单，按照要求写出 mapper.py 和 reducer.py 就可以了。

4.1.1 mapper.py

```
import sys

# input comes from STDIN (standard input)
for line in sys.stdin:
# remove leading and trailing whitespace
line = line.strip()
# split the line into words
words = line.split()
# increase counters
for word in words:
# write the results to STDOUT (standard output);
# what we output here will be the input for the
# Reduce step, i.e. the input for reducer.py
#
# tab-delimited; the trivial word count is 1
print '%s\t%s' % (word, 1)
```

4.1.2 reducer.py

```
#!/usr/bin/env python

from operator import itemgetter
import sys

current_word = None
current_count = 0
current_len = 0
word = None

# input comes from STDIN
for line in sys.stdin:
# remove leading and trailing whitespace
line = line.strip()

# parse the input we got from mapper.py
word, len = line.split('\t', 1)

# convert count (currently a string) to int
try:
len = int(len)
except ValueError:
# count was not a number, so silently
# ignore/discard this line
continue

# this IF-switch only works because Hadoop sorts map output
# by key (here: word) before it is passed to the reducer
if current_word == word:
current_len += len
current_count += 1
else:
if current_word:
# write result to STDOUT
print '%s\t%s' % (current_word, round(current_len/current_count,2))
current_len = len
current_count=1
current_word = word

# do not forget to output the last word if needed!
if current_word == word:
print '%s\t%s' % (current_word, round(current_len/current_count,2))
```

在这次实验中还复习了 python 的相关操作，但总的来说比较基础就不再对代码进行过多的解释。

4.1.3 结果展示

```
hduser@ubuntu:~$ echo "we become what we do " |~/experiment/src2/ex1_
apper.py | sort -k1,1| ~/experiment/src2/ex1_reducer.py
b       6.0
d       2.0
w       2.67
hduser@ubuntu:~$
```

使用新的测试语句进行测试

```
hduser@ubuntu:~$ echo "Portrait of Maria Portinari " |~/experiment/src
2/ex1_mapper.py | sort -k1,1| ~/experiment/src2/ex1_reducer.py
M       5.0
o       2.0
P       8.5
hduser@ubuntu:~$
```

4.2 exercise2

这个实验应用的是迭代的操作计算 PageRank
我使用的方法和 accumulate 相似，不断迭代进行计算。

4.2.1 mapper.py

```
#!/usr/bin/env python

import sys

PL = {}
PN = {}

for line in sys.stdin:
  line = line.strip()
  if len(line) > 0:
    word, key = line.split('\t', 1)
    key = float(key)
    PL[word] = key


Ar = (0.15 / 4)
print 'A\t%f' % Ar
```

```
Br = (0.15 / 4) + 0.85 * ((PL['A'] / 3) + PL['D'])
print 'B\t%f' % Br
Cr = (0.15 / 4) + 0.85 * ((PL['A'] / 3) + (PL['B'] / 2))
print 'C\t%f' % Cr
Dr = (0.15 / 4) + 0.85 * ((PL['A'] / 3) + (PL['B'] / 2) + (PL['C']))
print 'D\t%f' % Dr
```

4.2.2 结果展示

使用 python 直接运行 mapper.py

```
hduser@ubuntu:~/experiment/src/PageRank$ cat rank.txt | python mapper.
A       0.037500
B       0.320833
C       0.214583
D       0.427083
```

使用 hadoop 的迭代方法，得到相同的结果。

```
hduser@ubuntu:~/experiment/src/PageRank$ ./batch_test.sh 1
Processing 1
```

结果

```
A       0.037500
B       0.320833
C       0.214583
D       0.427083
```

4.2.3 在第一次完成实验的基础上，我又重新写了一份代码使用 mapper + reducer 的方法来进行操作

mapper.py 文件如下

```
#!/usr/bin/env python

import sys
#file1 = open("test2.txt")
# input comes from STDIN (standard input)
for line in sys.stdin:
# remove leading and trailing whitespace
  line = line.strip()
  # split the line into parts
  eles = line.split()
```

```
  current_link = eles[0]
  current_pr = float(eles[1])

  print '%s\t%s\t%f' % (current_link, 0, 0.0375)

  try:
    outlinks = eles[2:]

    outnum = float(len(outlinks))

    # increase counters
    for link in outlinks:
    # write the results to STDOUT (standard output);
    # what we output here will be the input for the
    # Reduce step, i.e. the input for reducer.py
    #
    # tab-delimited; the trivial word count is 1

    print '%s\t%s\t%f' % (link, current_link, 0.85*current_pr/outnum)
  except:
  pass
```

reducer.py 文件如下

```
#!/usr/bin/env python

from operator import itemgetter
import sys

current_page = None
current_pr = 0
dict1={}
dict2={}
for i in range(5):
  dict1[i] = []
  dict2[i] = 0;
#file1 = open("reduce.txt")
# input comes from STDIN
for line in sys.stdin:
  #print line
  line = line.strip()
  # parse the input we got from mapper.py
  try:
    page, target, pr = line.split()

  except:
    break
```

```
    # convert count (currently a string) to int
    try:
      pr = float(pr)
      target = int(target)
    except ValueError:
      continue

    dict1[target].append(page)
    dict2[int(page)]+=pr

for key,value in dict2.items():
    if key != 0:
      print key,'\t',value,'\t','\t'.join(dict1[int(key)])
```

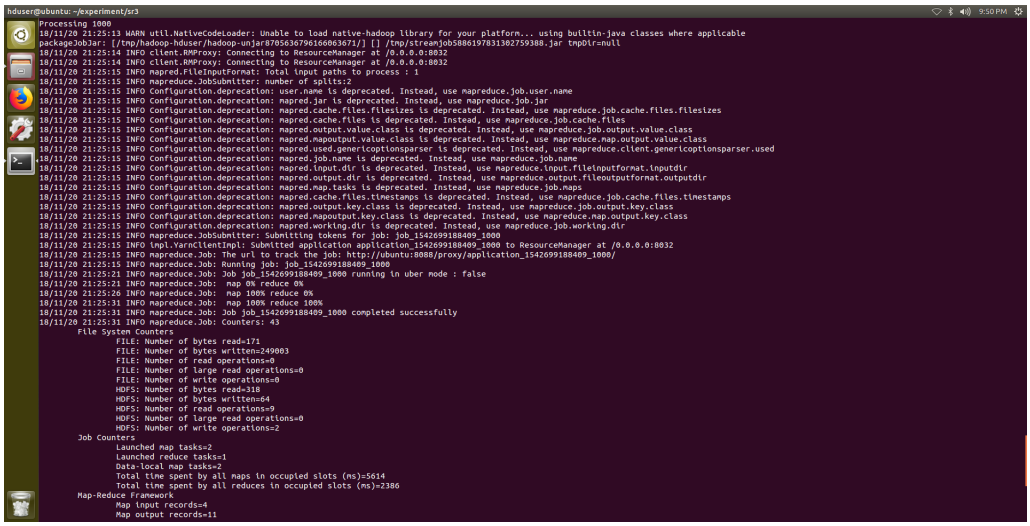
更新后的 batch_test.sh 文件

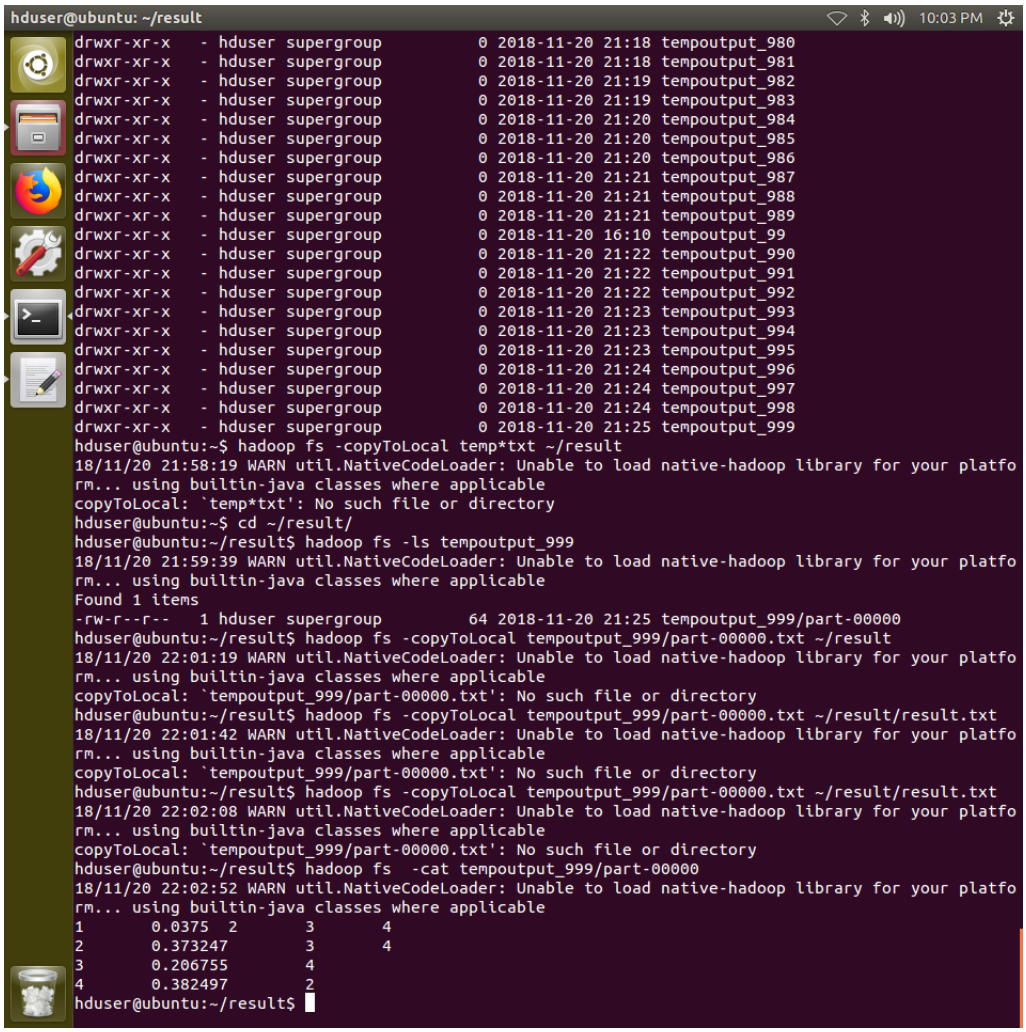
```
#/bin/bash

command='hadoop jar /usr/local/hadoop/share/hadoop/tools/lib/hadoop-
                                    streaming-2.2.0.jar -files mapper.py,
                                    reducer.py -mapper mapper.py -reducer
                                    reducer.py'
mv='hadoop fs -mv '
rm='hadoop fs -rm -r '
cp2local='hadoop fs -copyToLocal '
input='tempinput'
for ((i=1;i<$1+1;i++));
do
  echo "Processing $i"
  output="tempoutput_$i"
  eval "$command -input $input/* -output $output"
  input=$output
  eval "$rm $input/_SUCCESS"
done
mkdir /home/hduser/result
eval "$cp2local $output/* /home/hduser/result"
```

4.2.4 实验结果展示

现在有了 mapper 和 reducer 之后，一切都变得顺理成章起来。使用语句让 hadoop 执行 1000 次之后，发现结果收敛。（其实不用那么多次，50-100 次过程中就可以看到明显的收敛结果）


hduser@ubuntu: ~/experiment/sr3
9:50 PM
Processing 1000
18/11/20 21:25:13 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
packageJobJar: [/tmp/hadoop-hduser/hadoop-unjar8705636796160063671/] [] /tmp/streamjob5886197831302759388.jar tmpDir=null
18/11/20 21:25:14 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
18/11/20 21:25:14 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
18/11/20 21:25:15 INFO mapred.FileInputFormat: Total input paths to process : 1
18/11/20 21:25:15 INFO mapreduce.JobSubmitter: number of splits:2
18/11/20 21:25:15 INFO Configuration.deprecation: user.name is deprecated. Instead, use mapreduce.job.user.name
18/11/20 21:25:15 INFO Configuration.deprecation: mapred.jar is deprecated. Instead, use mapreduce.job.jar
18/11/20 21:25:15 INFO Configuration.deprecation: mapred.cache.files.filesizes is deprecated. Instead, use mapreduce.job.cache.files.filesizes
18/11/20 21:25:15 INFO Configuration.deprecation: mapred.cache.files is deprecated. Instead, use mapreduce.job.cache.files
18/11/20 21:25:15 INFO Configuration.deprecation: mapred.output.value.class is deprecated. Instead, use mapreduce.job.output.value.class
18/11/20 21:25:15 INFO Configuration.deprecation: mapred.mapoutput.value.class is deprecated. Instead, use mapreduce.map.output.value.class
18/11/20 21:25:15 INFO Configuration.deprecation: mapred.used.genericoptionsparser is deprecated. Instead, use mapreduce.client.genericoptionsparser.used
18/11/20 21:25:15 INFO Configuration.deprecation: mapred.job.name is deprecated. Instead, use mapreduce.job.name
18/11/20 21:25:15 INFO Configuration.deprecation: mapred.input.dir is deprecated. Instead, use mapreduce.input.fileinputformat.inputdir
18/11/20 21:25:15 INFO Configuration.deprecation: mapred.output.dir is deprecated. Instead, use mapreduce.output.fileoutputformat.outputdir
18/11/20 21:25:15 INFO Configuration.deprecation: mapred.map.tasks is deprecated. Instead, use mapreduce.job.maps
18/11/20 21:25:15 INFO Configuration.deprecation: mapred.cache.files.timestamps is deprecated. Instead, use mapreduce.job.cache.files.timestamps
18/11/20 21:25:15 INFO Configuration.deprecation: mapred.output.key.class is deprecated. Instead, use mapreduce.job.output.key.class
18/11/20 21:25:15 INFO Configuration.deprecation: mapred.mapoutput.key.class is deprecated. Instead, use mapreduce.map.output.key.class
18/11/20 21:25:15 INFO Configuration.deprecation: mapred.working.dir is deprecated. Instead, use mapreduce.job.working.dir
18/11/20 21:25:15 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1542699188409_1000
18/11/20 21:25:15 INFO impl.YarnClientImpl: Submitted application application_1542699188409_1000 to ResourceManager at /0.0.0.0:8032
18/11/20 21:25:15 INFO mapreduce.Job: The url to track the job: http://ubuntu:8088/proxy/application_1542699188409_1000/
18/11/20 21:25:15 INFO mapreduce.Job: Running job: job_1542699188409_1000
18/11/20 21:25:21 INFO mapreduce.Job: Job job_1542699188409_1000 running in uber mode : false
18/11/20 21:25:21 INFO mapreduce.Job: map 0% reduce 0%
18/11/20 21:25:26 INFO mapreduce.Job: map 100% reduce 0%
18/11/20 21:25:31 INFO mapreduce.Job: map 100% reduce 100%
18/11/20 21:25:31 INFO mapreduce.Job: Job job_1542699188409_1000 completed successfully
18/11/20 21:25:31 INFO mapreduce.Job: Counters: 43
File System Counters
FILE: Number of bytes read=171
FILE: Number of bytes written=249003
FILE: Number of read operations=0
FILE: Number of large read operations=0
FILE: Number of write operations=0
HDFS: Number of bytes read=318
HDFS: Number of bytes written=64
HDFS: Number of read operations=9
HDFS: Number of large read operations=0
HDFS: Number of write operations=2
Job Counters
Launched map tasks=2
Launched reduce tasks=1
Data-local map tasks=2
Total time spent by all maps in occupied slots (ms)=5614
Total time spent by all reduces in occupied slots (ms)=2386
Map-Reduce Framework
Map input records=4
Map output records=11



最后的结果：迭代 1 次：A 0.0375 B 0.320833 C 0.214583 D 0.429083

迭代 1000 次：A 0.0375 B 0.373247 C 0.206755 D 0.382497

Part IV

实验总结

这次实验主要是学习新的 Hadoop 知识及新的应用，很有趣！

在实践过程中，由于第一次的实验深入了解了 hadoop 的知识，所以在 exercise 操作中写 py 文件都很顺利。但是也确实在学习过程中出现了很多基础语言不清楚、基础概念不明晰而导致的错误，所以会在基础的语言角度更多地再去理解！

期待在下一次实验中学习更多的知识！