# MapReduce：Wordcount实例测试及源代码透析[¶](#MapReduce：Wordcount实例测试及源代码透析)

## 实验介绍[¶](#实验介绍)

本实验是基于MapReduce思想，编写WordCount程序。

## 实验原理[¶](#实验原理)

MapReduce是一种计算模型，简单的说就是将大批量的工作（数据）分解（MAP）执行，然后再将结果合并成最终结果（REDUCE）。这样做的好处是可以在任务被分解后，可以通过大量机器进行并行计算，减少整个操作的时间。

适用范围：数据量大，但是数据种类小可以放入内存。

基本原理及要点：将数据交给不同的机器去处理，数据划分，结果归约。

理解MapReduce和Yarn：在新版Hadoop中，Yarn作为一个资源管理调度框架，是Hadoop下MapReduce程序运行的生存环境。其实MapRuduce除了可以运行Yarn框架下，也可以运行在诸如Mesos，Corona之类的调度框架上，使用不同的调度框架，需要针对Hadoop做不同的适配。

一个完成的MapReduce程序在Yarn中执行过程如下：

（1）ResourcManager JobClient向ResourcManager提交一个job。

（2）ResourcManager向Scheduler请求一个供MRAppMaster运行的container，然后启动它。

（3）MRAppMaster启动起来后向ResourcManager注册。

（4）ResourcManagerJobClient向ResourcManager获取到MRAppMaster相关的信息，然后直接与MRAppMaster进行通信。

（5）MRAppMaster算splits并为所有的map构造资源请求。

（6）MRAppMaster做一些必要的MR OutputCommitter的准备工作。

（7）MRAppMaster向RM(Scheduler)发起资源请求，得到一组供map/reduce task运行的container，然后与NodeManager一起对每一个container执行一些必要的任务，包括资源本地化等。

（8）MRAppMaster 监视运行着的task 直到完成，当task失败时，申请新的container运行失败的task。

（9）当每个map/reduce task完成后，MRAppMaster运行MR OutputCommitter的cleanup 代码，也就是进行一些收尾工作。

（10）当所有的map/reduce完成后，MRAppMaster运行OutputCommitter的必要的job commit或者abort APIs。

（11）MRAppMaster退出。

#### 1. MapReduce编程[¶](#1.-MapReduce编程)

编写在Hadoop中依赖Yarn框架执行的MapReduce程序，并不需要自己开发MRAppMaster和YARNRunner，因为Hadoop已经默认提供通用的YARNRunner和MRAppMaster程序， 大部分情况下只需要编写相应的Map处理和Reduce处理过程的业务程序即可。

编写一个MapReduce程序并不复杂，关键点在于掌握分布式的编程思想和方法，主要将计算过程分为以下五个步骤：

（1）迭代。遍历输入数据，并将之解析成key/value对。

（2）将输入key/value对映射(map)成另外一些key/value对。

（3）依据key对中间数据进行分组(grouping)。

（4）以组为单位对数据进行归约(reduce)。

（5）迭代。将最终产生的key/value对保存到输出文件中。

## 实验要求[¶](#实验要求)

**我们是基于Hadoop V2.6.0版本，请大家参考前几节hadoop的相关安装配置实验进行安装配置。**

**注意：所有操作都在hadoop用户下进行操作**

## 实验步骤[¶](#实验步骤)

#### 1.首先启动hadoop集群[¶](#1.首先启动hadoop集群)

$ su - hadoop

口令输入：hadoop

$ echo $HADOOP\_HOME

/hadoop/hadoop

上述输出确认hadoop的环境变量设置有效，如果无效则激活环境变量：

$ bash

$ source ~/.bash\_profile

启动ssh,口令输入：hadoop

hadoop@357987c120a9:~$ sudo service ssh start

[sudo] password for hadoop:

[ ok ] Starting OpenBSD Secure Shell server: sshd.

启动命令为：

$ start-all.sh

检查是否运行成功

#执行jps命令可以查看到hadoop的几个主要进程:

$ jps

288 NameNode

528 SecondaryNameNode

367 DataNode

753 NodeManager

1063 Jps

#### 2.上传数据文件到HDFS[¶](#2.上传数据文件到HDFS)

创建/hadoop/hadoop-data/，并进入到/hadoop/hadoop-data/目录下

$ mkdir /hadoop/hadoop-data/

$ cd /hadoop/hadoop-data/

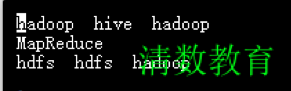
首先我们先创建一个test.txt文件,写入如下内容:

$ vi test.txt

hadoop hive hadoop

MapReduce

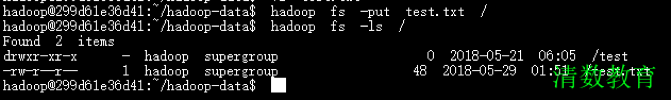
hdfs hdfs hadoop



上传至hdfs：

$ hadoop fs -put test.txt /

$ hadoop fs -ls /



#### 3.MapReduce实现WordCount实例（Python）[¶](#3.MapReduce实现WordCount实例（Python）)

（1）首先编写MapReduce WordCount 代码。

①首先编写map阶段的代码，创建一个Python程序，命名为“count\_mapper.py”

使用vi count\_mapper.py创建该程序，写入如下内容：

#!/usr/bin/env python

import sys

for line in sys.stdin:

line = line.strip()

words = line.split()

for word in words:

print '%s\t%s' % (word, 1)

写入完成后使用如下命令查看该文件中是否写入了上述内容：

$ more count\_mapper.py

②编写Reduce阶段的代码，创建一个Python程序，命名为“count\_reducer.py”

使用vi count\_reducer.py创建该程序，写入如下内容：

#!/usr/bin/env python

from operator import itemgetter

import sys

current\_word = None

current\_count = 0

word = None

for line in sys.stdin:

line = line.strip()

word, count = line.split('\t', 1)

try:

count = int(count)

except ValueError:

continue

if current\_word == word:

current\_count += count

else:

if current\_word:

print '%s\t%s' % (current\_word, current\_count)

current\_count = count

current\_word = word

# do not forget to output the last word if needed!

if current\_word == word:

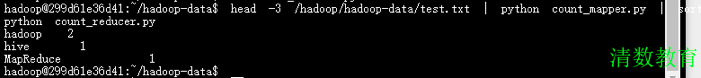
print '%s\t%s' % (current\_word, current\_count)

写入完成后使用如下命令查看该文件中是否写入了上述内容：

$ more count\_reducer.py

（2）程序编写完成后，首先在本地测试一下map和reduce，命令及图片如下：

$ head -3 /hadoop/hadoop-data/test.txt | python count\_mapper.py | sort | python count\_reducer.py

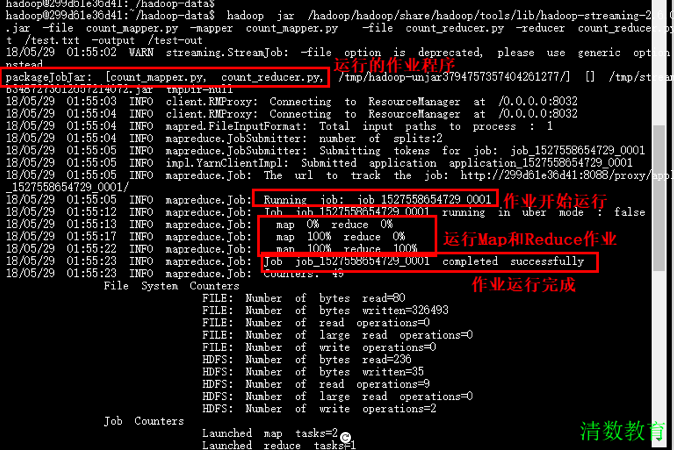


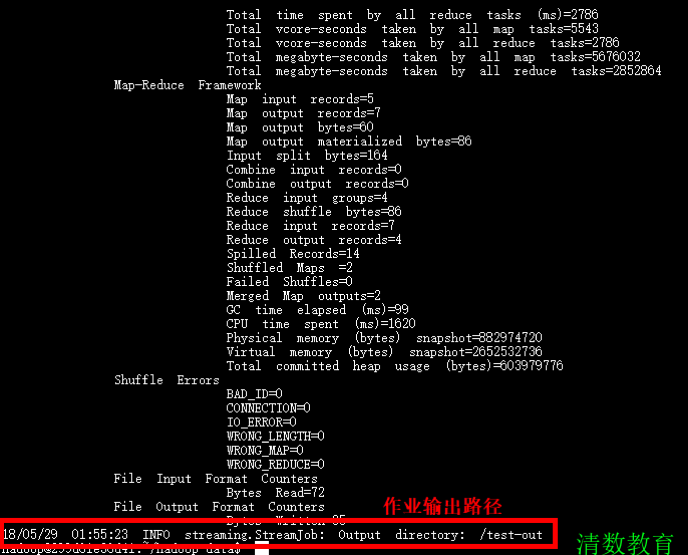
如上图，就证明map和reduce程序编写成功。

（3）运行该实例，命令如下：

$ hadoop jar /hadoop/hadoop/share/hadoop/tools/lib/hadoop-streaming-2.6.0.jar -file count\_mapper.py -mapper count\_mapper.py -file count\_reducer.py -reducer count\_reducer.py -input /test.txt -output /test-out

（4）运行结果





（5）代码透析

①map阶段的代码count\_mapper.py解析如下：

#!/usr/bin/env python

import sys

# 从标准输入过来的数据

for line in sys.stdin:

# 将首位的空格去掉

line = line.strip()

# 将这一行文本切分成单词（按空格）

words = line.split()

# 读一个单词写出一个<单词,1>

for word in words:

print '%s\t%s' % (word, 1)

②reduce阶段的代码count\_reducer.py解析如下：

#!/usr/bin/env python

from operator import itemgetter

import sys

current\_word = None

current\_count = 0

word = None

# 从标准输入过来的数据

for line in sys.stdin:

# 去除左右空格

line = line.strip()

# 按照tab键进行切分，得到word和次数1

word, count = line.split('\t', 1)

# 得到的1是一个字符串，需要类型转化

try:

count = int(count)

except ValueError:

# 如果不能转化成数字，输入有问题，转到下一行

continue

# 如果本次读取的单词和上一次一样，对次数加1

if current\_word == word:

current\_count += count

else:

if current\_word:

# 输出统计结果

print '%s\t%s' % (current\_word, current\_count)

current\_count = count

current\_word = word

# do not forget to output the last word if needed!

if current\_word == word:

print '%s\t%s' % (current\_word, current\_count)

## 实验结果[¶](#实验结果)

在HDFS上查看结果：

$ hadoop fs -ls /

$ hadoop fs -ls /test-out

$ hadoop fs -tail /test-out/part-00000

