# 实验2- MapReduce与Spark内存计算的性能对比

## 实验介绍

#### 1.实验内容

本节课程我们使用Hadoop MapReduce、Spark框架分别运行wordcount分析程序，来对MapReduce和Spark的性能进行对比。

#### 2.概述

2.1Hadoop MapReduce

Hadoop 的设计思路来源于 Google 的 GFS 和 MapReduce。它是一个开源软件框架，通过在集群计算机中使用简单的编程模型，可编写和运行分布式应用程序处理大规模数据。

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。

2.2 Spark

Spark 是一个基于内存计算的开源的集群计算系统，目的是让数据分析更加快速。Spark 提供了基于内存的计算集群，在分析数据时将数据导入内存以实现快速查询，“速度比”基于磁盘的系统，如比 Hadoop 快很多。Spark 最初是为了处理迭代算法，如机器学习、图挖掘算法等，以及交互式数据挖掘算法而开发的。在这两种场景下，Spark 的运行速度可以达到 Hadoop 的几百倍。

#### 3.数据集介绍

请自行准备本实验使用的数据集，保存成word.txt文件。

（1）查看数据集的大小：du -h word.txt

（2）查看数据集的字符串数：wc -w word.txt

（3）查看字符集的内容：more word.txt

## 实验要求

#### 1. Java （需要安装1.6.x及其以上版本)

#### 2. Hadoop和Spark

在这里，我们使用了Hadoop V3.1.4版本以及Spark-3.0.1版本。

## 实验步骤

#### 1.实验前准备

$ su - hadoop

口令输入：hadoop

$ bash

$ 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：

$ 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.

hadoop@357987c120a9:~$

#### 2.启动hadoop

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

3409 NodeManager

3733 Jps

2698 NameNode

3258 ResourceManager

2843 DataNode

3084 SecondaryNameNode

#### 3.启动spark

①首先启动master

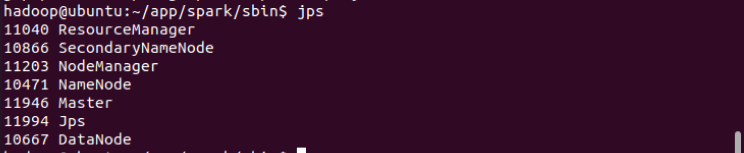
$ cd /hadoop/app/spark/sbin/

$ ./start-master.sh



查看Master进程是否启动

$ jps



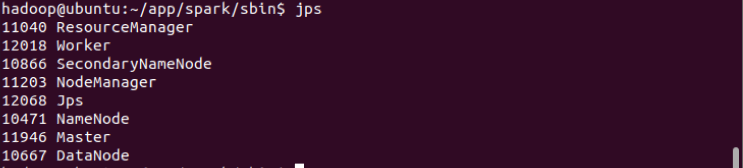
②启动slave

$ ./start-slave.sh spark://127.0.0.1:7077



查看Woker进程是否启动

$ jps



#### 4.将本次实验的数据文件上传到HDFS文件系统

我们可以建立一个/hadoop/data目录。

$ mkdir /hadoop/data

$ cd /hadoop/data

将word.txt上传至该目录下。

查看该目录下是否有了word.txt文件

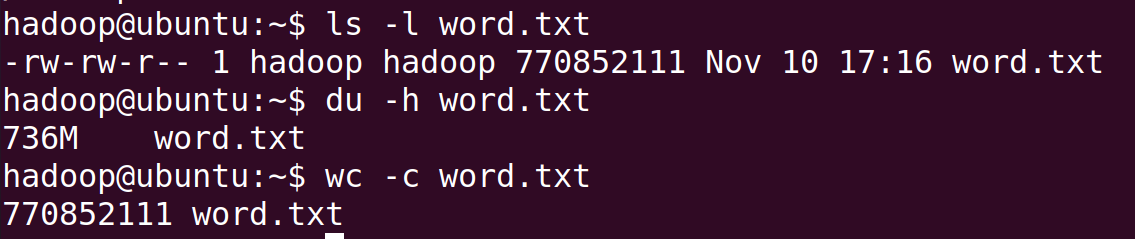
$ ls -l

查看数据集的大小：

$ du -h word.txt

查看数据集的字符串数：

$ wc -c word.txt



查看字符集的内容：

$ more word.txt

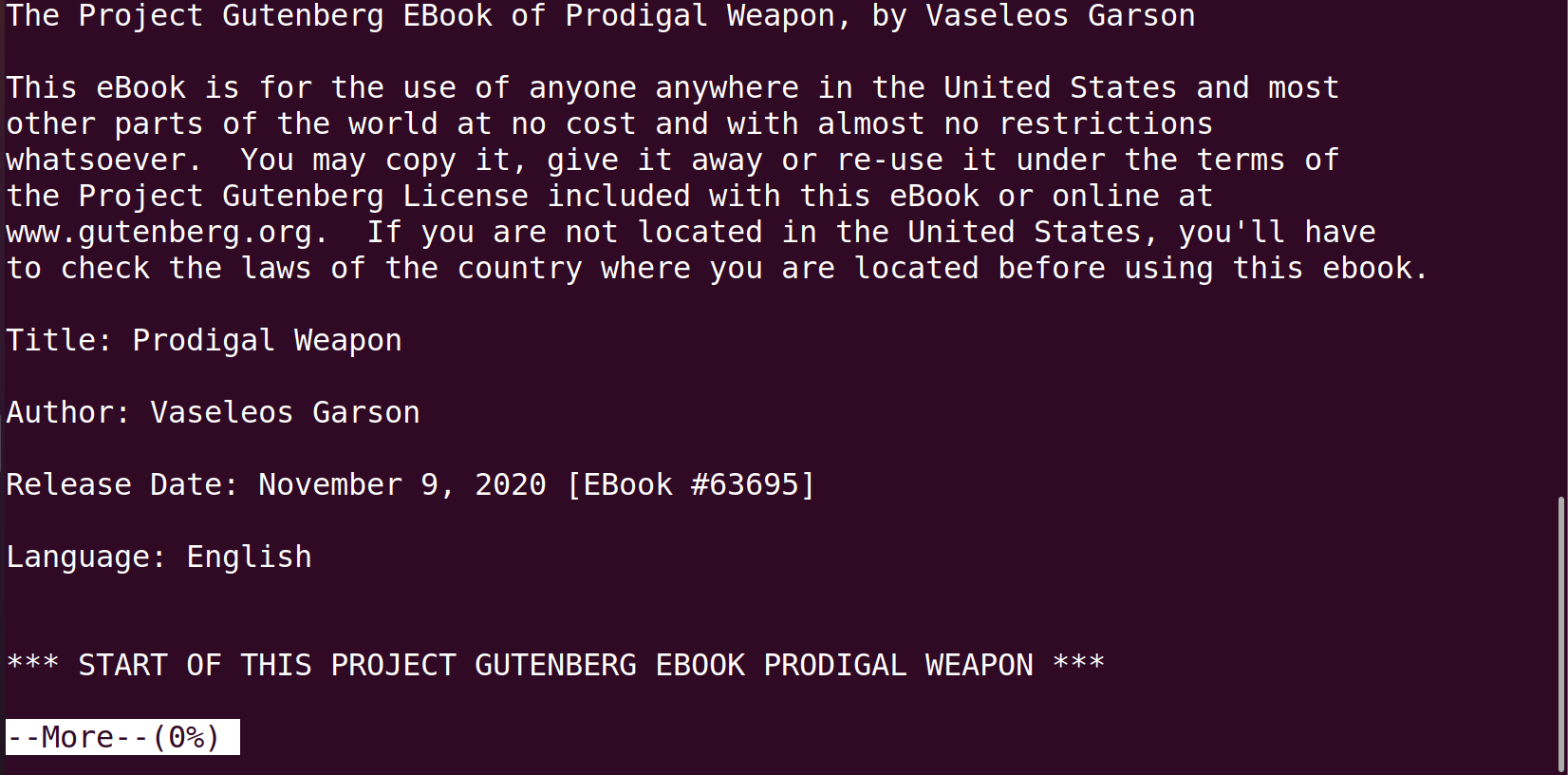
more命令说明：

①more命令会以一页一页的显示方便使用者逐页阅读，而最基本的指令就是按空白键（space）就往下一页显示，按 b 键就会往回（back）一页显示，而且还有搜寻字串的功能 。more命令从前向后读取文件，因此在启动时就加载整个文件。

②空格键 向下滚动一屏

③q 退出more

因此，向下一页按“空格键”；退出按q

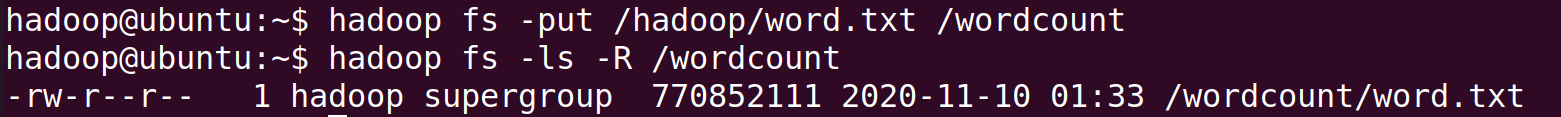


将文件上传到HDFS/wordcount

$ hadoop fs -mkdir /wordcount

$ hadoop fs -put /hadoop/word.txt /wordcount

$ hadoop fs -ls -R /wordcount



#### 5.MapReduce实现WordCount实例（Python）

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

$ mkdir /hadoop/hadoop-data

$ cd /hadoop/hadoop-data

（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”

$ vi count\_mapper.py

写入如下内容：

#!/usr/bin/env python3

import sys

for line in sys.stdin:

line = line.strip()

words = line.split()

for word in words:

print("%s\t%s" % (word, 1))

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

$ vi count\_reducer.py

写入如下内容：

#!/usr/bin/env python3

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

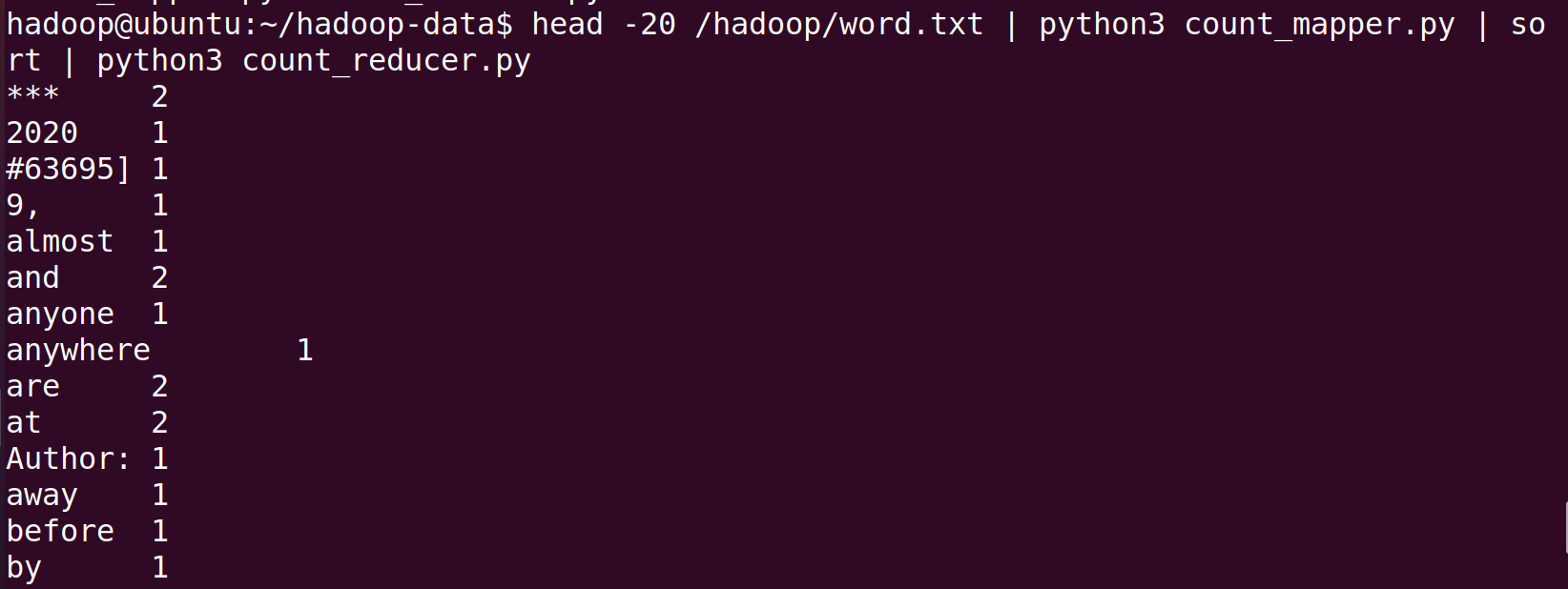
# do not forget to output the last word if needed!

if current\_word == word:

print ("%s\t%s" % (current\_word, current\_count))

（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 -20 /hadoop/word.txt | python3 count\_mapper.py | sort | python3 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-3.1.4/share/hadoop/tools/lib/hadoop-streaming-3.1.4.jar -file count\_mapper.py -mapper count\_mapper.py -file count\_reducer.py -reducer count\_reducer.py -input /wordcount/word.txt -output /wordcount-out/mapreduce-out

（4）查看结果

$ hadoop fs -tail /wordcount-out/mapreduce-out/part-00000

#### 6.Spark实现WordCount实例（python）

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

$ mkdir /hadoop/spark-data

$ cd /hadoop/spark-data

（1）首先编写Spark WordCount 代码，创建一个Python程序，命名为“wordcount.py”。

$ vi wordcount.py

写入如下内容：

#!/usr/bin/env python3

from pyspark import SparkContext

inputFile = 'hdfs://localhost:9000/wordcount/word.txt'

outputFile = 'hdfs://localhost:9000/wordcount-out/spark-out'

sc = SparkContext('local', 'wordcount')

text\_file = sc.textFile(inputFile)

counts = text\_file.flatMap(lambda line: line.split(' ')).map(lambda word: (word, 1)).reduceByKey(lambda a, b: a+b)

counts.saveAsTextFile(outputFile)

（2）运行该实例

$ cd /hadoop/app/spark/

$ ./bin/spark-submit --master spark://localhost:7077 /hadoop/spark-data/wordcount.py

（4）查看运行结果

$ hadoop fs -tail /wordcount-out/spark-out/part-00000

#### 7.代码透析

#### （1）MapReduce实现WordCount实例（Python）

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)

#### （2）Spark实现WordCount实例（python）

#!/usr/bin/env python

#导入包

from pyspark import SparkContext

# 输入输出路径，输出路径不需要自己创建，系统会自动生成

inputFile = 'hdfs://localhost:9000/wordcount/word.txt'

outputFile = 'hdfs://localhost:9000/wordcount-out/spark-out'

sc = SparkContext('local', 'wordcount')

text\_file = sc.textFile(inputFile)

##counts

counts = text\_file.flatMap(lambda line: line.split(' ')).map(lambda word: (word, 1)).reduceByKey(lambda a, b: a+b)

counts.saveAsTextFile(outputFile)

## 实验结果

分析对比使用Hadoop的MapReduce和Spark两者的计算速度。