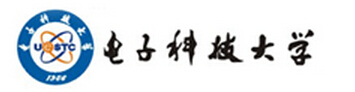
****

**课 程 报 告**

**词频统计**

学 院： 信息与软件工程学院

小组成员1： 201821090124 徐增

学生姓名2： 201822090316 杨庆

学生姓名3： 201852090632张晟铭

学生姓名4： 2015220304002敬鑫

学生姓名5： 201822090313 邹洋

指导教师： 林迪

目录

[MapReduce及Spark词频统计](#_Toc4868_WPSOffice_Level1) [3](#_Toc4868_WPSOffice_Level1)

[1 实验内容](#_Toc24615_WPSOffice_Level1) [3](#_Toc24615_WPSOffice_Level1)

[2 实验环境](#_Toc21592_WPSOffice_Level1) [3](#_Toc21592_WPSOffice_Level1)

[3 Hadoop介绍](#_Toc9463_WPSOffice_Level1) [3](#_Toc9463_WPSOffice_Level1)

[3.1 Hadoop产生背景](#_Toc4150_WPSOffice_Level2) [3](#_Toc4150_WPSOffice_Level2)

[3.2 Hadoop简介](#_Toc18708_WPSOffice_Level2) [4](#_Toc18708_WPSOffice_Level2)

[3.3 Hadoop架构](#_Toc14548_WPSOffice_Level2) [4](#_Toc14548_WPSOffice_Level2)

[3.3.1 分布式架构简介](#_Toc1326_WPSOffice_Level3) [4](#_Toc1326_WPSOffice_Level3)

[3.3.2 Hadoop2.0](#_Toc30184_WPSOffice_Level3) [5](#_Toc30184_WPSOffice_Level3)

[3.3.3 伪分布式架构](#_Toc1753_WPSOffice_Level3) [5](#_Toc1753_WPSOffice_Level3)

[4 环境搭建](#_Toc4150_WPSOffice_Level1) [6](#_Toc4150_WPSOffice_Level1)

[4.1 ubuntu16.04配置Hadoop伪分布式](#_Toc15354_WPSOffice_Level2) [6](#_Toc15354_WPSOffice_Level2)

[4.1.1 实验环境](#_Toc11603_WPSOffice_Level3) [6](#_Toc11603_WPSOffice_Level3)

[4.1.2 SSH免密码登录](#_Toc26094_WPSOffice_Level3) [6](#_Toc26094_WPSOffice_Level3)

[4.1.3 安装java1.8.0\_181](#_Toc20533_WPSOffice_Level3) [8](#_Toc20533_WPSOffice_Level3)

[4.1.4 安装hadoop-2.7.6](#_Toc19593_WPSOffice_Level3) [8](#_Toc19593_WPSOffice_Level3)

[4.1.5 伪分布式配置](#_Toc32702_WPSOffice_Level3) [9](#_Toc32702_WPSOffice_Level3)

[4.1.6 运行Hadoop](#_Toc8321_WPSOffice_Level3) [13](#_Toc8321_WPSOffice_Level3)

[4.2 ubuntu16.04配置spark](#_Toc5902_WPSOffice_Level2) [15](#_Toc5902_WPSOffice_Level2)

[4.2.1 ubuntu16.04安装scala-2.12.8](#_Toc13894_WPSOffice_Level3) [15](#_Toc13894_WPSOffice_Level3)

[4.2.2 ubuntu16.04安装spark-2.4.1-bin-hadoop2.7](#_Toc22328_WPSOffice_Level3) [16](#_Toc22328_WPSOffice_Level3)

[4.3 ubuntu16.04配置Hadoop全分布式集群](#_Toc1326_WPSOffice_Level2) [19](#_Toc1326_WPSOffice_Level2)

[5 实验过程](#_Toc18708_WPSOffice_Level1) [21](#_Toc18708_WPSOffice_Level1)

[5.1 WordCount词频统计](#_Toc30184_WPSOffice_Level2) [21](#_Toc30184_WPSOffice_Level2)

[5.1.1 MapReduce词频统计](#_Toc2261_WPSOffice_Level3) [21](#_Toc2261_WPSOffice_Level3)

[5.1.2 Spark词频统计](#_Toc32225_WPSOffice_Level3) [23](#_Toc32225_WPSOffice_Level3)

[6 实验结果对比分析](#_Toc14548_WPSOffice_Level1) [26](#_Toc14548_WPSOffice_Level1)

[6.1 MapReduce执行效率](#_Toc1753_WPSOffice_Level2) [26](#_Toc1753_WPSOffice_Level2)

[6.1.1 运行时间](#_Toc22196_WPSOffice_Level3) [26](#_Toc22196_WPSOffice_Level3)

[6.1.2 资源分配量](#_Toc25802_WPSOffice_Level3) [32](#_Toc25802_WPSOffice_Level3)

[6.2 Spark执行效率](#_Toc11603_WPSOffice_Level2) [34](#_Toc11603_WPSOffice_Level2)

[6.2.1 运行时间](#_Toc28551_WPSOffice_Level3) [34](#_Toc28551_WPSOffice_Level3)

[6.2.2 输出文本大小](#_Toc29760_WPSOffice_Level3) [35](#_Toc29760_WPSOffice_Level3)

[6.3 两者结果对比](#_Toc26094_WPSOffice_Level2) [35](#_Toc26094_WPSOffice_Level2)

[7 从软件体系架构角度解释分析实验结果](#_Toc15354_WPSOffice_Level1) [36](#_Toc15354_WPSOffice_Level1)

[7.1 运行时间差异](#_Toc20533_WPSOffice_Level2) [36](#_Toc20533_WPSOffice_Level2)

[7.2 CPU使用率](#_Toc19593_WPSOffice_Level2) [36](#_Toc19593_WPSOffice_Level2)

[7.3 内存使用率](#_Toc32702_WPSOffice_Level2) [36](#_Toc32702_WPSOffice_Level2)

[7.4 容错性](#_Toc8321_WPSOffice_Level2) [37](#_Toc8321_WPSOffice_Level2)

[7.5 通用性](#_Toc13894_WPSOffice_Level2) [37](#_Toc13894_WPSOffice_Level2)

[8 实验总结](#_Toc5902_WPSOffice_Level1) [37](#_Toc5902_WPSOffice_Level1)

[9 实验遇到的问题](#_Toc1326_WPSOffice_Level1) [38](#_Toc1326_WPSOffice_Level1)

**MapReduce**及**Spark**词频统计

1. **实验内容**
2. 安装部署Hadoop、HDFS、MapReduce、Spark，配置分布式开发环境；
3. 分别在MapReduce及Spark执行词频统计，；
4. 比较不同文本大小在MapReduce及Spark执行效率异同；
5. 从软件体系架构角度解释分析实验结果。
6. 实验环境
7. 操作系统：Ubuntu16.04；
8. 系统内存：24G；
9. 小组分工：

|  |  |  |
| --- | --- | --- |
| 姓名 | 学号 | 任务 |
| 徐增 | 201821090124 | 实验结果分析，Spark实验 |
| 杨庆 | 201822090316 | 环境搭建，MapReduce实验 |
| 张晟铭 | 201852090632 | 问题解决，实验分析 |
| 敬鑫 | 2015220304002 | 资料收集，撰写PPT |
| 邹洋 | 201822090313 | 实验汇总，撰写实验报告 |

1. **Hadoop介绍**
   1. Hadoop产生背景

Hadoop最早起源于Nutch。Nutch的设计目标是构建一个大型的全网搜索引擎，包括网页抓取、索引、查询等功能，但随着抓取网页数量的增加，遇到了严重的可扩展性问题——如何解决数十亿网页的存储和索引问题。

2003年、2004年谷歌发表的两篇论文为该问题提供了可行的解决方案。

（1）分布式文件系统（GFS），可用于处理海量网页的存储

（2）分布式计算框架（MapReduce），可用于处理海量网页的索引计算问题。

Nutch的开发人员完成了相应的开源实现HDFS和MapReduce，并从Nutch中剥离成为独立项目Hadoop，到2008年1月，Hadoop成为Apache顶级项目，迎来了它的快速发展期。

* 1. Hadoop简介

Hadoop是Apache旗下的一套开源软件平台，Hadoop是利用服务器集群对数据进行存储，根据用户的自定义业务逻辑，对海量数据进行分布式计算。广义上来说，Hadoop通常是指一个更广泛的概念——Hadoop生态圈。

Hadoop解决了海量数据的存储（HDFS）、海量数据的技术（MapReduce）、资源调度（YARN）等问题。其中重点组件包括：

1. HDFS：分布式文件系统；
2. MapReduce：分布式运算程序开发框架；
3. YARN:资源调度系统。
4. ZOOKEEPER：分布式协调服务基础组件；
5. HIVE：SQL数据仓库工具；
6. HBASE：基于Hadoop的分布式海量数据库；
7. Sqoop：数据迁移工具；
8. Flume：日志数据采集框架；
   1. Hadoop架构
      1. 分布式架构简介

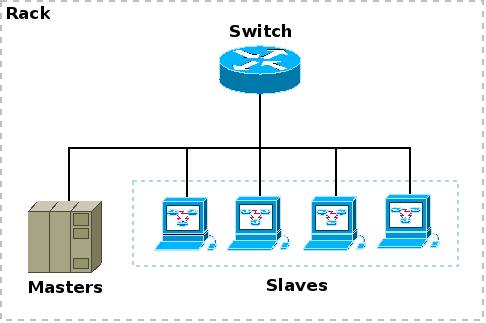
1.单机的问题

1. 存储能力有限；
2. 计算能力有限；
3. 有单点故障等。

2.分布式架构解决了单机的问题

3.经典分布式主从架构（Master-Slave）

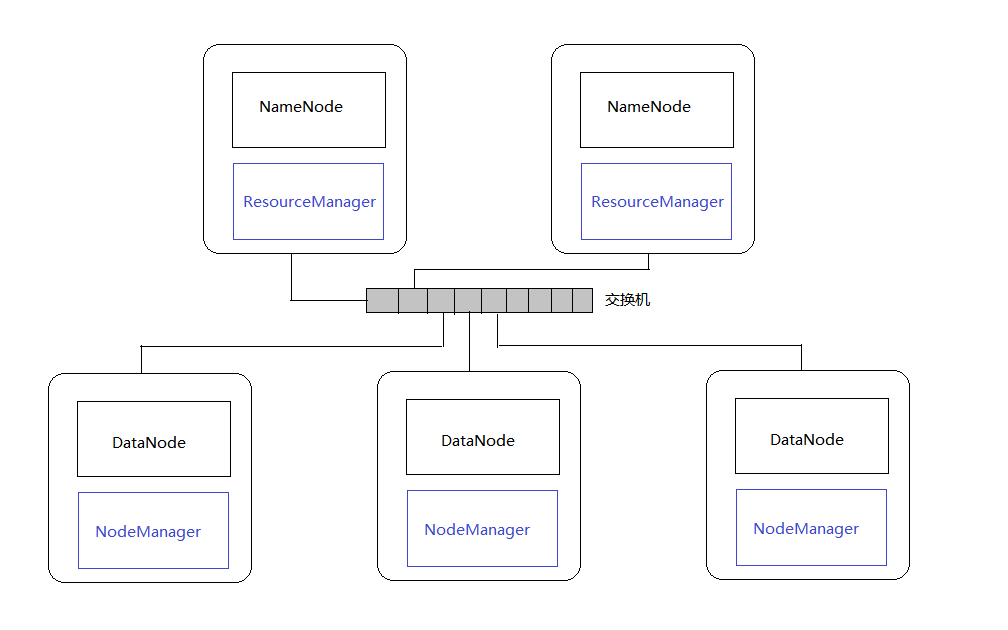
4.Master负责管理，且可以有多个，防止单点故障的发生。Slave负责干活，Slave有多个，并且可以动态的添加或移除。



* + 1. Hadoop2.0

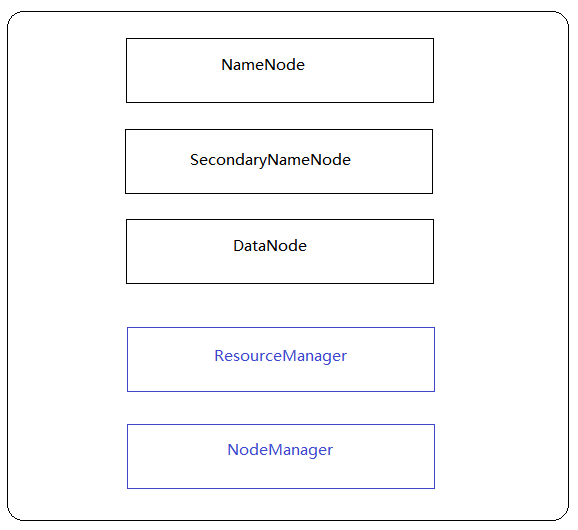
（1）HDFS ：NameNode（老大） DataNode（小弟）

（2）YARN ：ResourceManager（老大） NodeManager（小组长）

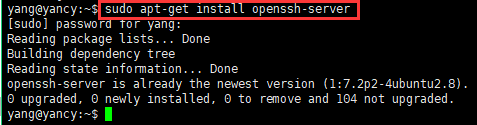


* + 1. 伪分布式架构

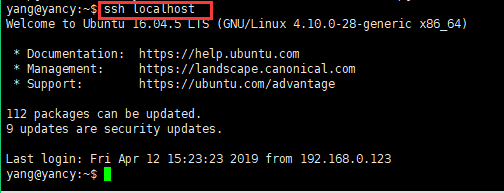
1. NameNode：HDFS的管理节点，负责DataNode的管理和元数据管理；
2. SecondaryNameNode：NameNode的一个助理，帮助NameNode管理元数据，防止元数据丢失；
3. DataNode：负责数据存储；
4. ResourceManager：YARN的管理节点，负责NodeManager的管理、任务调度等；
5. NodeManager：YARN的节点管理器，负责向ResourceManager汇报当前节点的状态和启动计算任务进程（YarnChild）并监控YarnChild。



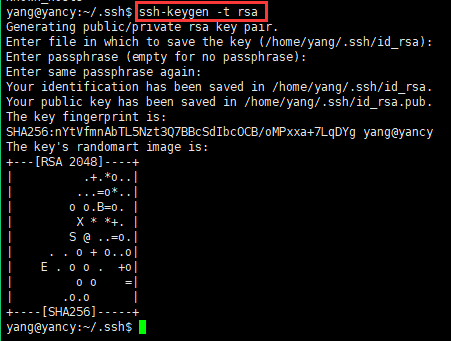
1. 环境搭建
   1. ubuntu16.04配置Hadoop伪分布式
      1. 实验环境
2. 操作系统：Ubuntu16.04
3. Java环境：jdk1.8.0\_181
4. Hadoop版本：hadoop-2.7.6
5. Scala版本：scala-2.12.8
6. Spark版本：spark-2.4.1-bin-hadoop2.7
   * 1. SSH免密码登录
7. 输入：sudo apt-get install openssh-server，安装SSH server；



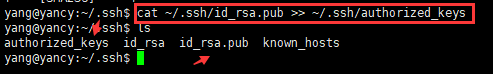
1. 输入：cd ~/.ssh/，如果没法进入该目录，执行一次ssh localhost；



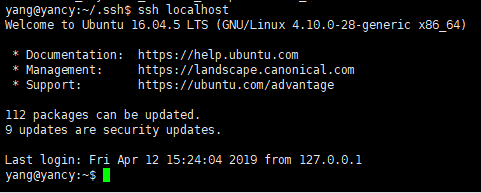
1. 输入：ssh-keygen -t rsa，三次回车后，该目录下将会产生id\_rsa，id\_rsa.pub文件；



1. 输入：cat ~/.ssh/id\_rsa.pub >> ~/.ssh/authorized\_keys加入授权；



1. 输入：ssh localhost，如果不提示输入密码则SSH无密登陆配置成功；



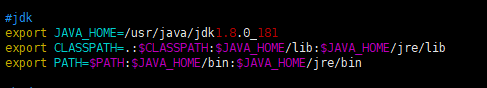
* + 1. 安装java1.8.0\_181

1. https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html下载jdk-8u181-linux-x64.tar.gz，输入：sudo tar zxvf jdk-8u201-linux-x64.tar.gz -C /usr/java/jdk1.8.0\_181将jdk-8u181-linux-x64.tar.gz解压到/usr/java/目录下；
2. 输入：sudo vim ~/.bashrc 配置环境变量，在最后添加三行：

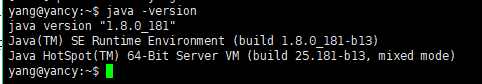
export JAVA\_HOME=/usr/java/jdk1.8.0\_181

export CLASSPATH=.:$CLASSPATH:$JAVA\_HOME/lib:$JAVA\_HOME/jre/lib

export PATH=$PATH:$JAVA\_HOME/bin:$JAVA\_HOME/jre/bin



1. 输入：source ~/.bashrc，使新配置的环境变量生效；
2. 输入：java -version，查看Java版本，检测是否安装成功；



* + 1. 安装hadoop-2.7.6

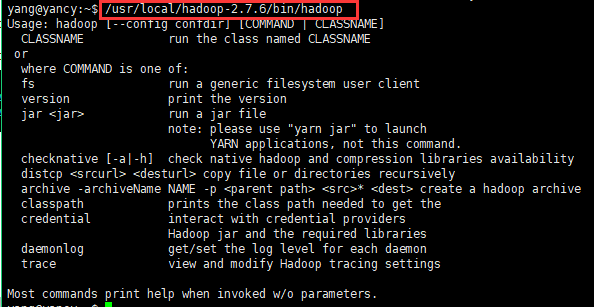
1. 在https://mirrors.tuna.tsinghua.edu.cn/apache/hadoop/common/下载hadoop-2.7.6.tar.gz，输入：sudo tar zxvf hadoop-2.7.6.tar.gz -C /usr/local/hadoop-2.7.6将hadoop-2.7.6.tar.gz解压到/usr/local/目录下；
2. 输入：sudo vim ~/.bashrc添加如下两行，然后输入source ~./bashrc；

export HADOOP\_HOME=/usr/local/hadoop-2.7.6

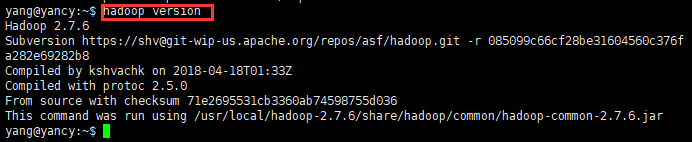
export PATH=$PATH:$HADOOP\_HOME/bin:$HADOOP\_HOME/sbin



1. 输入：/usr/local/hadoop-2.7.6/bin/hadoop查看hadoop命令相关使用信息；



1. 输入：hadoop version查看版本。



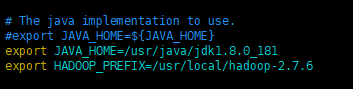
* + 1. 伪分布式配置

Hadoop 可以在单节点上以伪分布式的方式运行，Hadoop 进程以分离的 Java 进程来运行，节点既作为 NameNode 也作为 DataNode，同时，读取的是 HDFS 中的文件。Hadoop 的配置文件位于 /usr/local/hadoop-2.7.6/etc/hadoop/ 中，伪分布式需要修改2个配置文件 core-site.xml 和 hdfs-site.xml 。Hadoop的配置文件是 xml 格式，每个配置以声明 property 的 name 和 value 的方式来实现。此处我们另外修改了配置文件。

（1）JAVA\_HOME位于/usr/java/jdk1.8.0\_181，Hadoop在/usr/local/hadoop-2.7.6。输入：sudo vim /usr/local/hadoop-2.7.6/etc/hadoop/hadoop-env.sh添加两行参数：

export JAVA\_HOME=/usr/java/jdk1.8.0\_181

export HADOOP\_PREFIX=/usr/local/hadoop-2.7.6



（2）输入：sudo vim /usr/local/hadoop-2.7.6/etc/hadoop/core-site.xml修改core-site.xml文件添加如下内容：

<configuration>

<!-- 配置Hadoop运行时产生数据的存储目录，不是临时的数据 -->

<property>

<name>hadoop.tmp.dir</name>

<value>file:/usr/local/hadoop-2.7.6/tmp</value>

<description>Abase for other temporary directories.</description>

</property>

<!-- 配置hdfs的Namenode（老大）的地址 -->

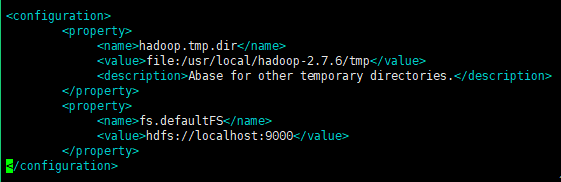
<property>

<name>fs.defaultFS</name>

<value>hdfs://localhost:9000</value>

</property>

</configuration>



（3）输入：sudo vim /usr/local/hadoop-2.7.6/etc/hadoop/hdfs-site.xml修改配置文件hdfs-site.xml添加如下内容：

<configuration>

<!-- 指定HDFS存储数据的副本数量 -->

<property>

<name>dfs.replication</name>

<value>1</value>

</property>

<property>

<name>dfs.namenode.name.dir</name>

<value>file:/usr/local/hadoop-2.7.6/tmp/dfs/name</value>

</property>

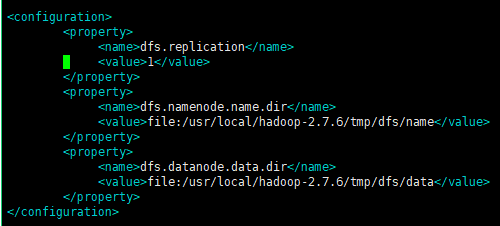
<property>

<name>dfs.datanode.data.dir</name>

<value>file:/usr/local/hadoop-2.7.6/tmp/dfs/data</value>

</property>

</configuration>



（4）将mapred-site.xml.template重命名为mapred-site.xml，

输入：sudo mv mapred-site.xml.template mapred-site.xml

然后输入：sudo vim /usr/local/hadoop-2.7.6/etc/hadoop/mapred-site.xml修改配置文件mapred-site.xml添加如下内容：

<configuration>

<!-- 指定Mapreduce编程模型运行在yarn上 -->

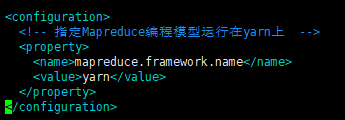
<property>

<name>mapreduce.framework.name</name>

<value>yarn</value>

</property>

</configuration>



（5）输入：sudo vim /usr/local/hadoop-2.7.6/etc/hadoop/yarn-site.xml修改配置文件yarn-site.xml添加如下内容：

<configuration>

<!-- 指定yarn的老大（ResourceManager的地址） -->

<property>

<name>yarn.resourcemanager.hostname</name>

<value>yancy</value>

</property>

<!-- mapreduce执行shuffle时获取数据的方式 -->

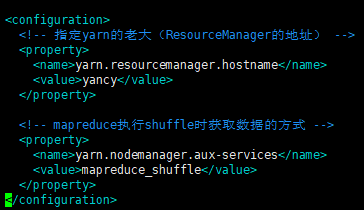
<property>

<name>yarn.nodemanager.aux-services</name>

<value>mapreduce\_shuffle</value>

</property>

</configuration>



* + 1. 运行Hadoop

Hadoop 的运行方式是由配置文件决定的（运行 Hadoop 时会读取配置文件），因此如果需要从伪分布式模式切换回非分布式模式，需要删除core-site.xml中的配置项。此外，伪分布式虽然只需要配置 fs.defaultFS 和 dfs.replication 就可以运行（可参考官方教程），不过若没有配置 hadoop.tmp.dir 参数，则默认使用的临时目录为 /tmp/hadoo-hadoop，而这个目录在重启时有可能被系统清理掉，导致必须重新执行 format才行。所以我们进行了设置，同时也指定 dfs.namenode.name.dir 和 dfs.datanode.data.dir，否则在接下来的步骤中可能会出错。

在Hadoop安装包目录下有几个比较重要的目录：

1）sbin：启动或停止Hadoop相关服务的脚本；

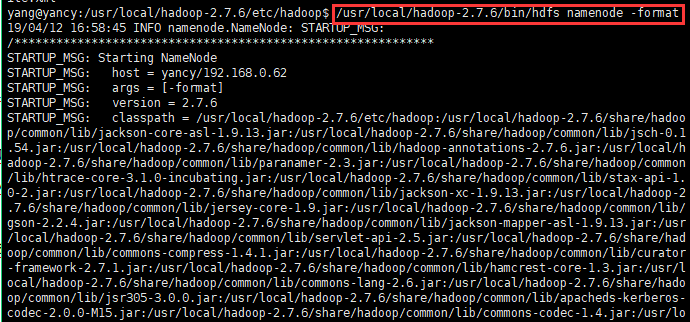
2）bin：对Hadoop相关服务（HDFS,YARN）进行操作的脚本；

3）etc：Hadoop的配置文件目录；

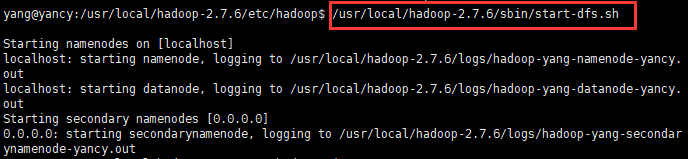
4）share：Hadoop的依赖jar包和文档，文档可以被删掉；

5）lib：Hadoop的本地库（对数据进行压缩解压缩功能的）。

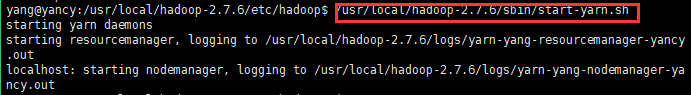
1. 输入：/usr/local/hadoop-2.7.6/bin/hdfs namenode -format执行 NameNode 的格式化；



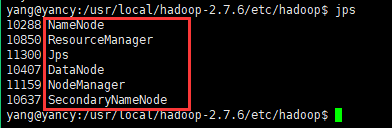
1. 输入：/usr/local/hadoop-2.7.6/sbin/start-dfs.sh启动NameNode和DataNode进程，并查看启动结果；



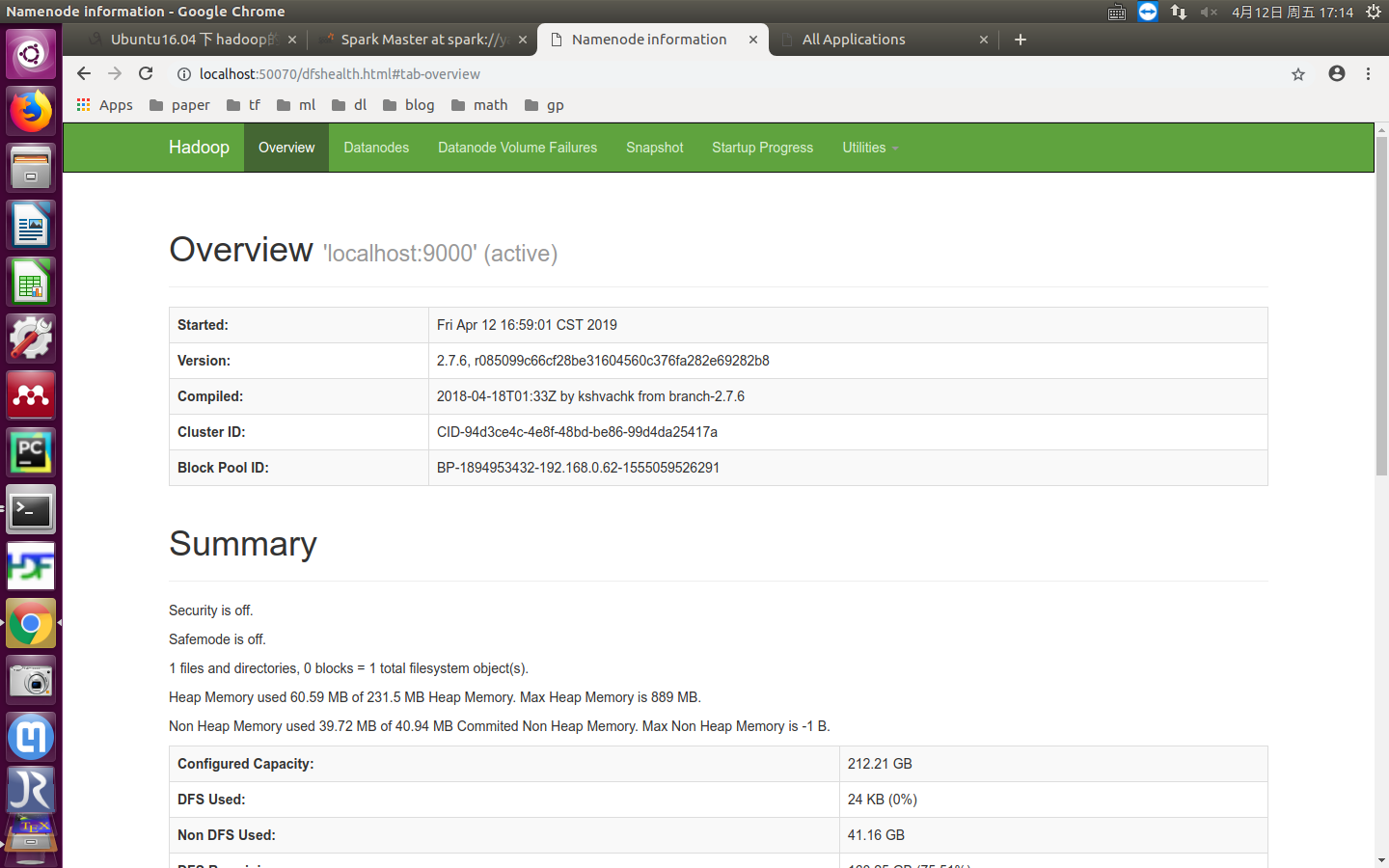
1. 输入：/usr/local/hadoop-2.7.6/sbin/start-yarn.sh启动ResourceManager和NodeManager；



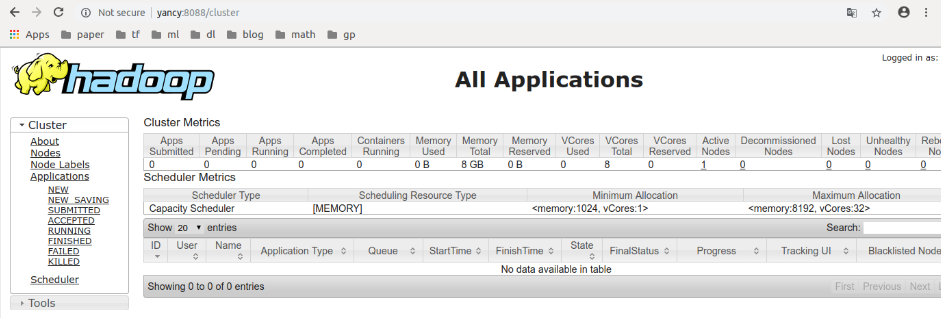
1. 输入：jps，判断是否成功启动，若成功启动则会列出如下进程: “NameNode”、”DataNode”、“SecondaryNameNode”、“ResourceManager”和NodeManager；



1. 访问HDFS的管理界面：在浏览器访问http://localhost:50070 查看 NameNode 和 DataNode的相关信息，还可以在线查看 HDFS 中的文件；



1. 访问YARN的管理界面：在浏览器访问http://yancy:8088查看Cluster相关信息。



* 1. ubuntu16.04配置spark
     1. ubuntu16.04安装scala-2.12.8

1. 输入：sudo tar -xzvf scala-2.12.8.tgz -C /usr/local，解压scala到/usr/local。网址为：https://www.scala-lang.org/download/；
2. 输入：sudo vim ~/.bashrc，在最后添加下面内容：

export SCALA\_HOME=/usr/local/scala-2.12.8

export PATH=$SCALA\_HOME/bin:$PATH



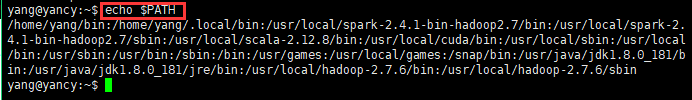
1. 输入：source ~/.bashrc，使新配置的环境变量生效；
2. 输入：scala -version查看版本。
   * 1. ubuntu16.04安装spark-2.4.1-bin-hadoop2.7
3. 输入：sudo tar -zxf spark-2.4.1-bin-hadoop2.7.tgz -C /usr/local，解压下载的spark文件。网址为：<http://spark.apache.org/downloads.html>
4. 输入：sudo vim ~/.bashrc，在最后添加下面内容：

export SPARK\_HOME=/usr/local/spark-2.4.1-bin-hadoop2.7

export PATH=$SPARK\_HOME/bin:$SPARK\_HOME/sbin:$PATH



1. 输入：source ~/.bashrc，使新配置的环境变量生效；



1. 拷贝配置文件：

cd /usr/local/spark-2.4.1-bin-hadoop2.7

sudo cp ./conf/spark-env.sh.template ./conf/spark-env.sh

1. 输入：sudo vim /usr/local/spark-2.4.1-bin-hadoop2.7/conf/spark-env.sh，修改配置文件，添加下面一行：

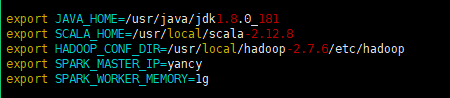
export JAVA\_HOME=/usr/java/jdk1.8.0\_181

export SCALA\_HOME=/usr/local/scala-2.12.8

export HADOOP\_CONF\_DIR=/usr/local/hadoop-2.7.6/etc/hadoop

export SPARK\_MASTER\_IP=yancy

export SPARK\_WORKER\_MEMORY=1g

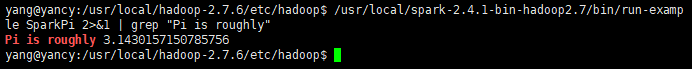


1. 输入：sudo cp /usr/local/spark-2.4.1-bin-hadoop2.7/conf/slaves.template slaves，将slaves.template重命名为slaves；
2. 输入：sudo vim /usr/local/spark-2.4.1-bin-hadoop2.7/conf/slaves，将slaves中的localhost修改为主机名，我的是yancy。



1. 运行简单示例：

/usr/local/spark-2.4.1-bin-hadoop2.7/bin/run-example SparkPi 2>&1 | grep "Pi is roughly"



1. 输入；sudo chown -R yang:yang spark-2.4.1-bin-hadoop2.7/，修改权限；
2. 输入：/usr/local/spark-2.4.1-bin-hadoop2.7/sbin/start-all.sh，启动Spark；
3. 编写启动脚本start\_script.sh启动Hadoop以及Spark：

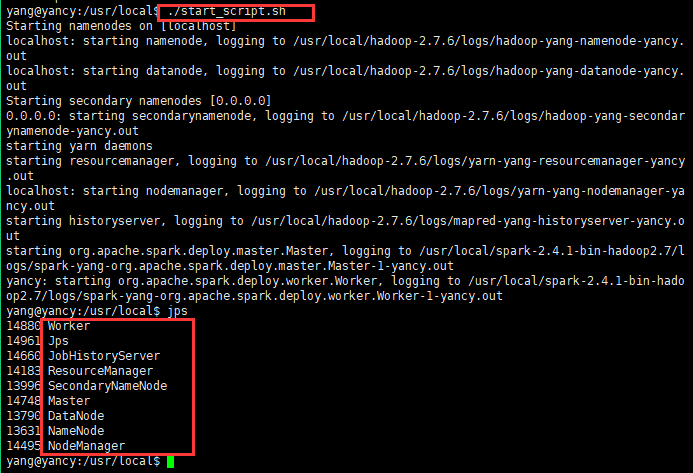
#!/bin/bash

start-dfs.sh # 启动Hadoop

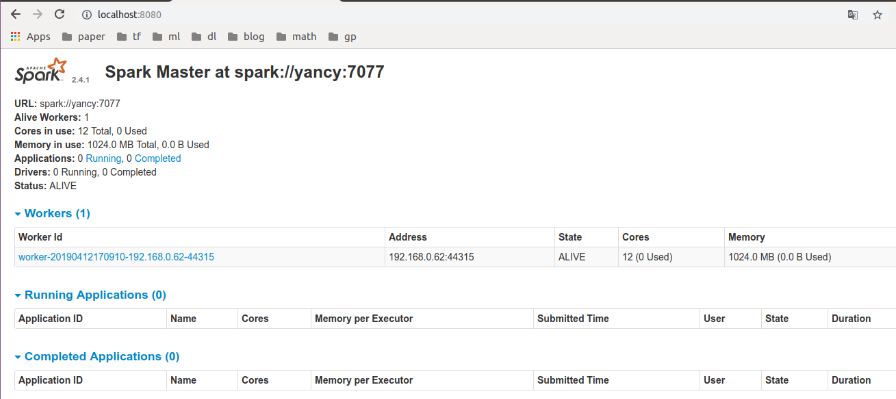
start-yarn.sh # 启动Yarn

mr-jobhistory-daemon.sh start historyserver # 启动历史服务器,以便在Web中查看任务运行情况。

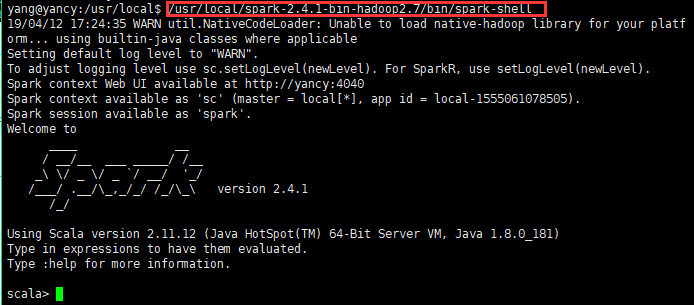
/usr/local/spark-2.4.1-bin-hadoop2.7/sbin/start-all.sh # 启动Spark



1. 通过WEB页面查看：浏览器中输入地址：localhost:8080。



1. 输入：/usr/local/spark-2.4.1-bin-hadoop2.7/bin/spark-shell，启动SparkContext。



1. 编写停止脚本stop\_script.sh停止Hadoop以及Spark：

#!/bin/bash

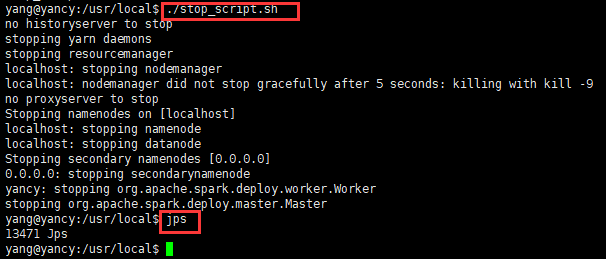
mr-jobhistory-daemon.sh stop historyserver # 停止历史服务器

stop-yarn.sh # 停止Yarn

stop-dfs.sh # 停止Hadoop

#/usr/local/hadoop/sbin/stop-all.sh # 停止Hadoop以及yarn

/usr/local/spark-2.4.1-bin-hadoop2.7/sbin/stop-all.sh # 停止Spark



* 1. **ubuntu16.04配置Hadoop全分布式集群**

搭建完全分布式集群需准备三台主机，一个主节点yancy和两个从节点Slave、Slave2，首先需要对主机名进行修改。

* + 1. 配置hosts文件

（1）输入：sudo vim /etc/hostname，修改主机名为yancy；

（2）输入：sudo vim /etc/hosts，添加各主机IP地址如下：

192.168.0.62 yancy

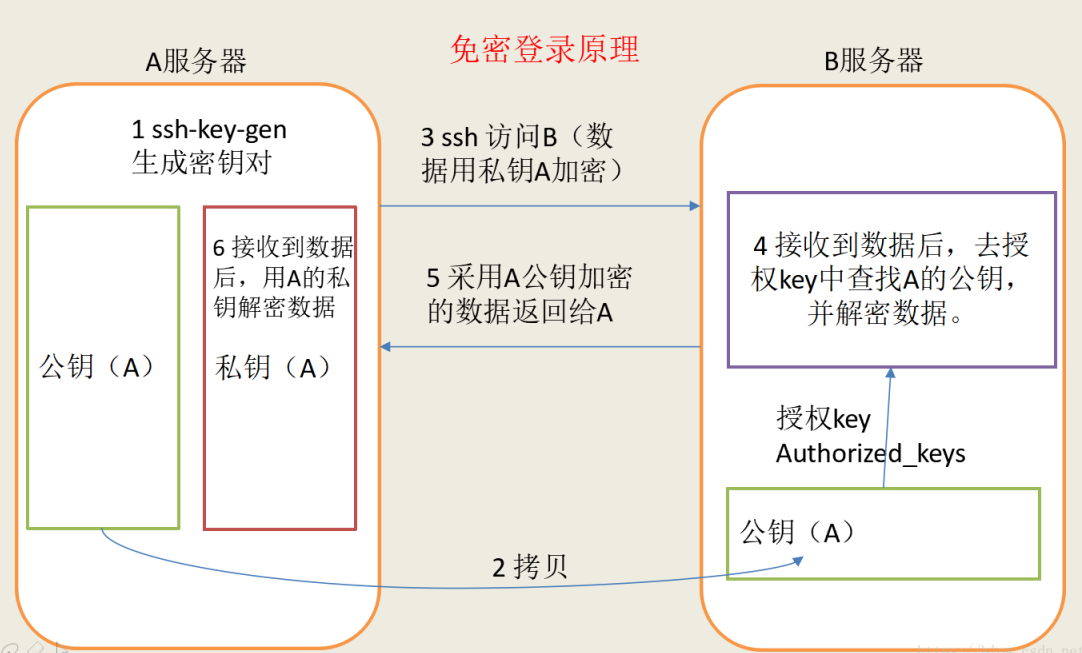
192.168.0.104 Slave1

192.168.0.50 Slave2

（3）其它两台从节点也都要修改hostname和hosts文件。配置完hosts后三台主机就可以进行通信了，可以互相ping通，是可以ping通的。

* + 1. SSH免密码登录

1. 输入：dpkg --list|grep ssh，查看安装的openssh-server；
2. 输入：ssh-keygen -t rsa，三次回车后，该目录下将会产生id\_rsa，id\_rsa.pub文件；
3. 输入：cat ~/.ssh/id\_rsa.pub >> ~/.ssh/authorized\_keys加入授权；
4. 输入：scp /home/yang/.ssh/id\_rsa.pub qxxhemu@Slave1:~/.ssh/，将公钥复制到其他从机，或ssh-copy-id -i ~/.ssh/id\_rsa.pub Slave1；
5. 输入：scp /home/yang/.ssh/id\_rsa.pub long@Slave2:~/.ssh/
6. 输入：cat ~/.ssh/id\_rsa.pub >> ~/.ssh/authorized\_keys加入授权；
7. 输入：ssh qxxhemu@Slave1，第一次需要密码，之后exit退出，再ssh qxxhemu@Slave1就不需要密码登录成功。



* + 1. 修改slaves文件

1. 输入：sudo vim /usr/local/hadoop-2.7.6/etc/hadoop/slaves，添加内容：Slave1和Slave2；
   * 1. 配置Hadoop及spark文件

其余步骤类似，不再详述。

1. 实验过程
   1. WordCount词频统计
      1. MapReduce词频统计

下面为一些HDFS常用命令：

hadoop fs -mkdir /tmp/input 在HDFS上目录/tmp/input；

hadoop fs -put input1.txt /tmp/input 把本地文件input1.txt传到HDFS的/tmp/input目录下；

hadoop fs -get input1.txt /tmp/input/input1.txt 把HDFS文件拉到本地；

hadoop fs -ls /tmp/output 列出HDFS的目录/tmp/output；

hadoop fs -cat /tmp/ouput/output1.txt 查看HDFS上文件/tmp/ouput/output1.txt；

hadoop fs -rmr /tmp/intput 删除HDFS上的目录/tmp/intput；

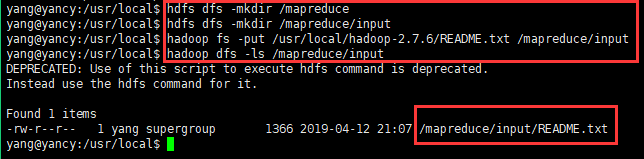
hadoop dfsadmin -report 查看HDFS状态，比如每个DataNode的情况；

hadoop dfsadmin -safemode leave 离开安全模式；

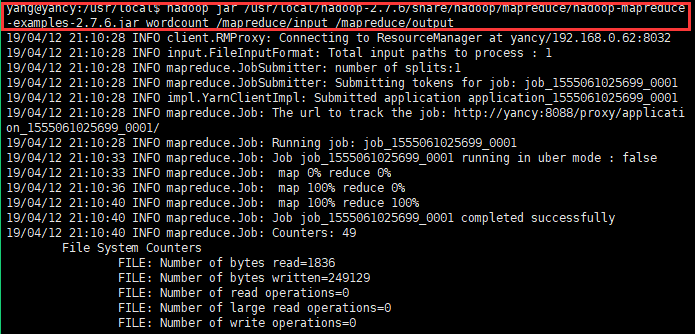
hadoop dfsadmin -safemode enter 进入安全模式。

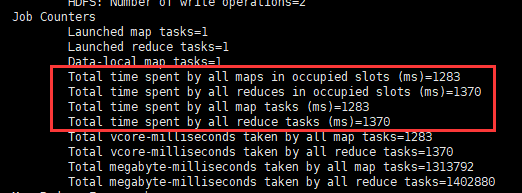
WordCount词频统计如下步骤：

1. 输入：start-all.sh，启动HDFS；
2. 输入：hadoop dfs -ls /，查看HDFS下面包含的文件目录，第一次运行hdfs什么文件都没有；
3. 输入：hdfs dfs -mkdir -p /mapreduce/input，在HDFS中创建一个文件目录/mapreduce/input；
4. 输入：hadoop fs -put /usr/local/hadoop-2.7.6/README.txt /mapreduce/input，将/usr/local/hadoop-2.7.6/README.txt上传至/mapreduce/input中；
5. 输入：hadoop dfs -ls /mapreduce/input查看/mapreduce/input下多了一个README.txt；

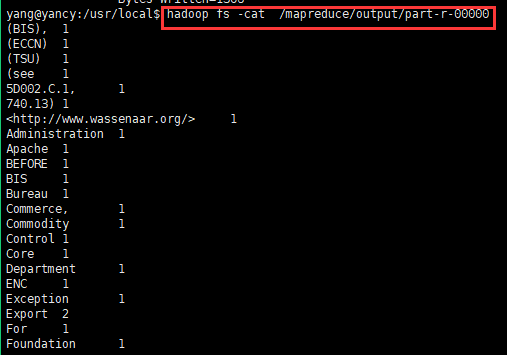


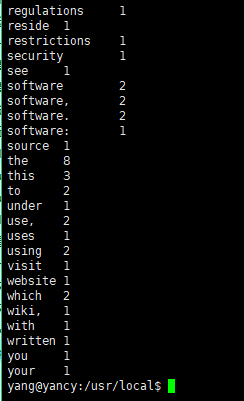
1. 执行如下命令运行wordcount 并将结果输到output：hadoop jar /usr/local/hadoop-2.7.6/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.6.jar wordcount /mapreduce/input /mapreduce/output；





1. 执行成功后output目录下会生成两个文件：一个是\_SUCCESS 成功标志的文件，里面没有内容，另一个是 part-r-00000 ，通过以下命令查看执行的结果：hadoop fs -cat /mapreduce/output/part-r-00000。





* + 1. Spark词频统计

1. 输入：/usr/local/spark-2.4.1-bin-hadoop2.7/bin/spark-shell，启动spark shell；
2. 或者：/usr/local/spark-2.4.1-bin-hadoop2.7/bin/spark-shell \

> --master spark://yancy:7077 \

> --executor-memory 500m

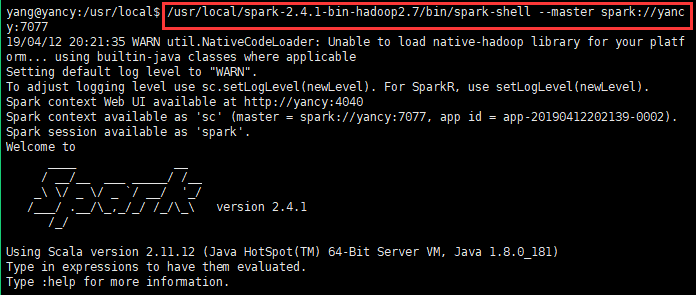
> --total-executor-cores 1

参数说明：

--master spark://yancy:7077 ：指定Master的地址；

--executor-memory 500m ：指定每个worker可用内存为500m；

--total-executor-cores 1 ：指定整个集群使用的CPU核数为1个；



1. 输入：hadoop fs -mkdir -p /spark/input，创建/spark/input文件夹；
2. 输入：hadoop fs -put /usr/local/hadoop-2.7.6/README.txt /spark/input，将/usr/local/hadoop-2.7.6/README.txt上传至/spark/input中；



1. 在spark shell中用scala编写spark程序，按空格分割数据，输入：

sc.textFile("/spark/input/README.txt").flatMap(\_.split(" ")).map((\_,1)).reduceByKey(\_+\_).saveAsTextFile("/spark/output")；



说明：

sc是SparkContext对象，该对象是提交spark程序的入口；

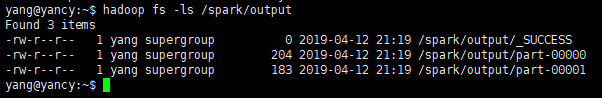
textFile("/spark/input/README.txt") 是从hdfs中读取数据；

flatMap(\_.split(" ")) 先map再压平；

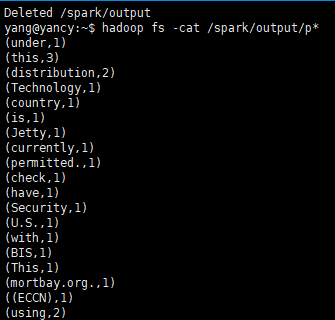
map((\_,1)) 将单词和1构成元组；

reduceByKey(\_+\_) 按照key进行reduce，并将value累加；

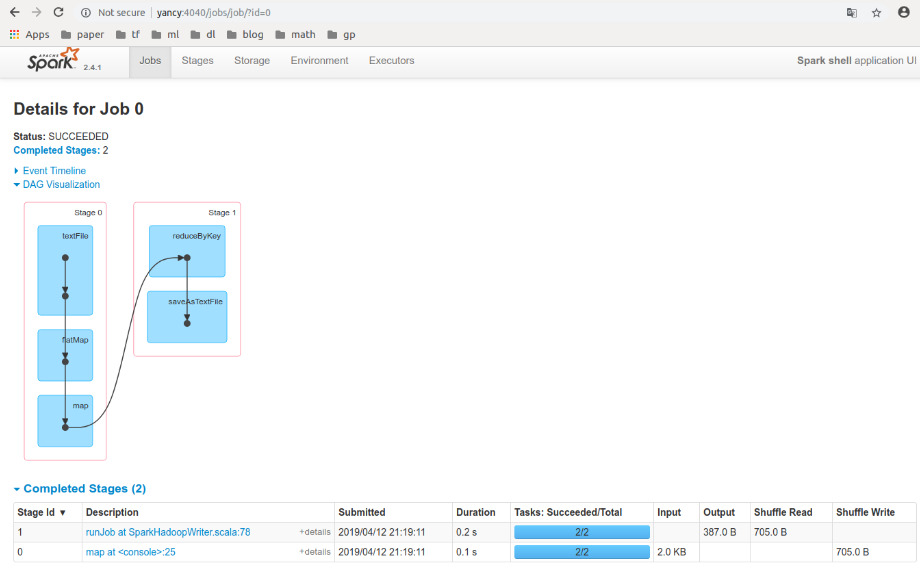
saveAsTextFile("/spark/output") 将结果写入到hdfs中。



1. 输入：hadoop fs -cat /spark/output/p\*，查看hdfs的执行结果：



1. 在浏览器输入：yancy:4040，查看spark UI 。

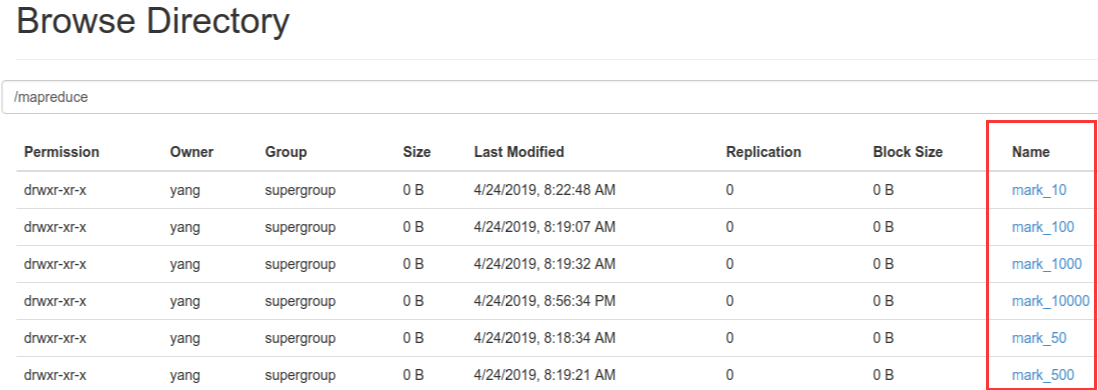


1. 实验结果对比分析
   1. MapReduce执行效率

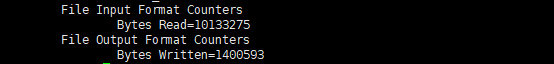
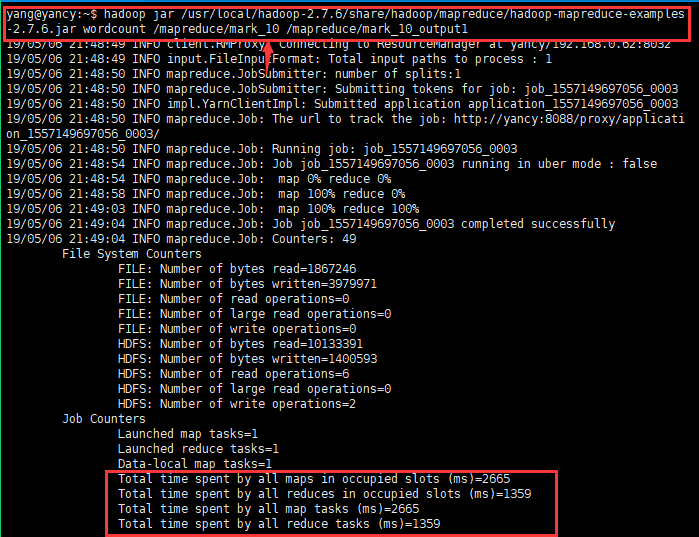
本次实验选取的实验样本为五个大小分别为10M、50M、100M、500M、100M、10000M的英文文本文件，使用MapReduce、Spark框架分别运行不同大小文本进行对比分析相关指标，进而统计各自的运行时间、运行时CPU占用率、内存占用率等执行效率和容错性、通用性的对比。

* + 1. 运行时间

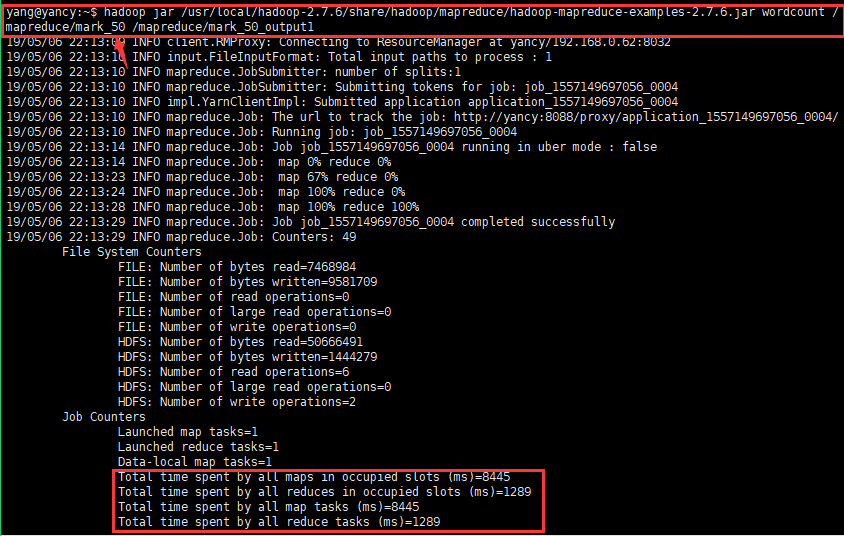
（1）查看10M、50M、100M、500M、100M、10000M大小文本：



（2）10M大小文本执行 Map、Reduce 时间及文件大小输入输出结果图：

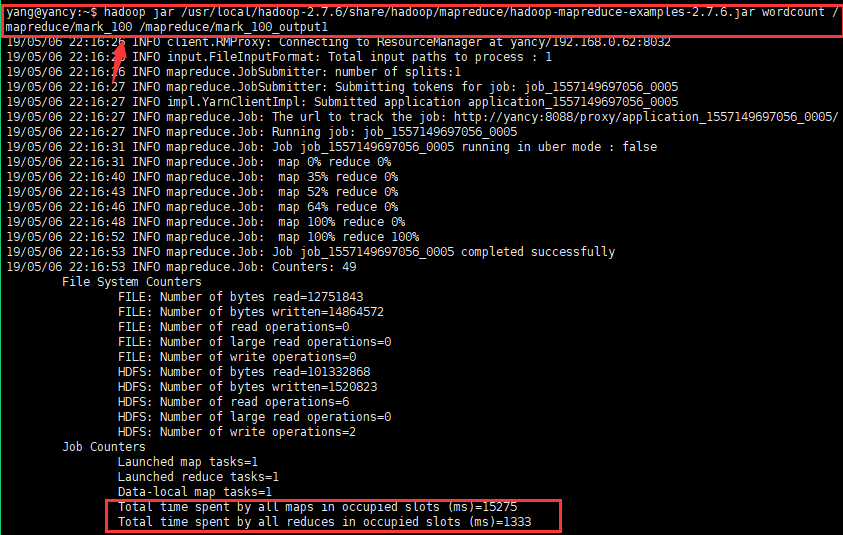


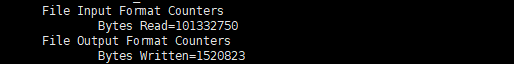
（3）50M大小文本执行 Map、Reduce 时间及文件大小输入输出结果图：



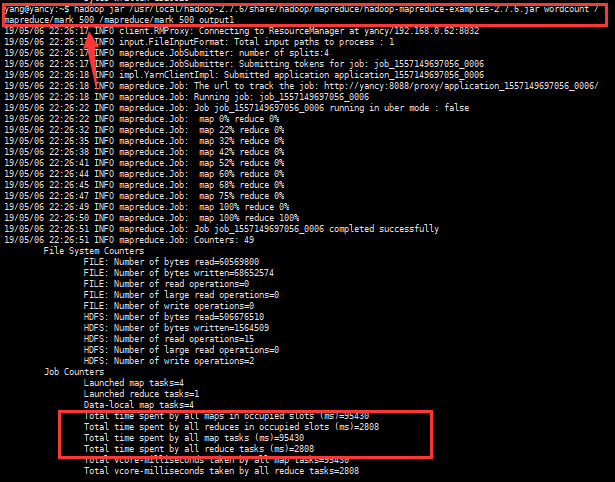


（4）100M大小文本执行 Map、Reduce 时间及文件大小输入输出结果图：

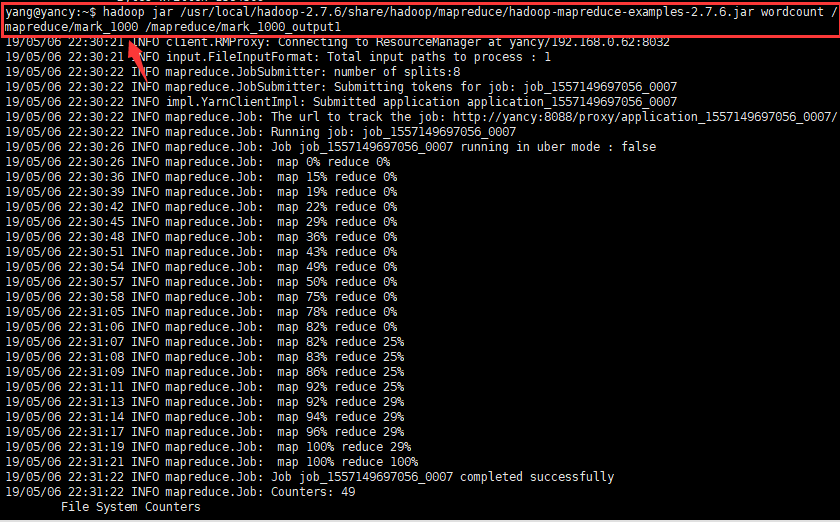


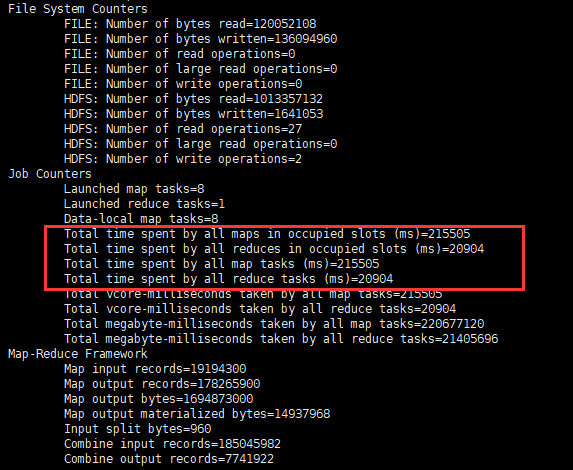


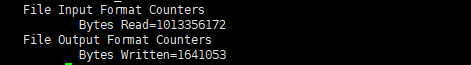
（5）500M大小文本执行 Map、Reduce 时间及文件大小输入输出结果图：



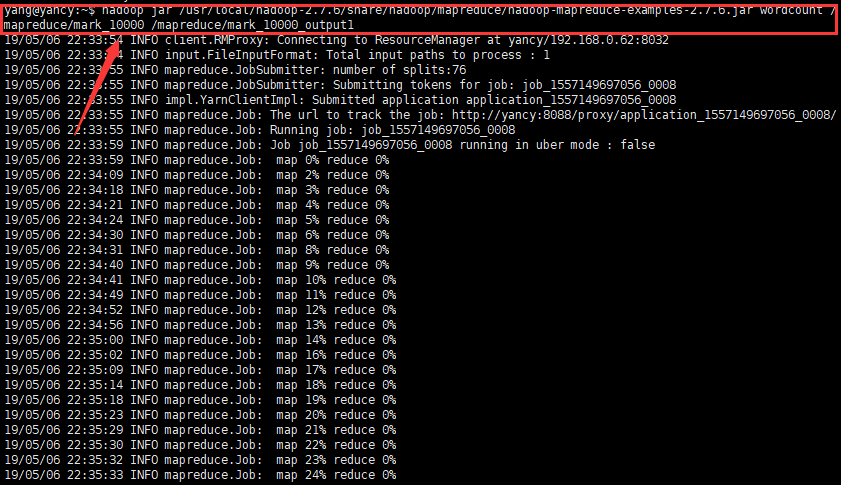
（6）1000M大小文本执行 Map、Reduce 时间及文件大小输入输出结果图：

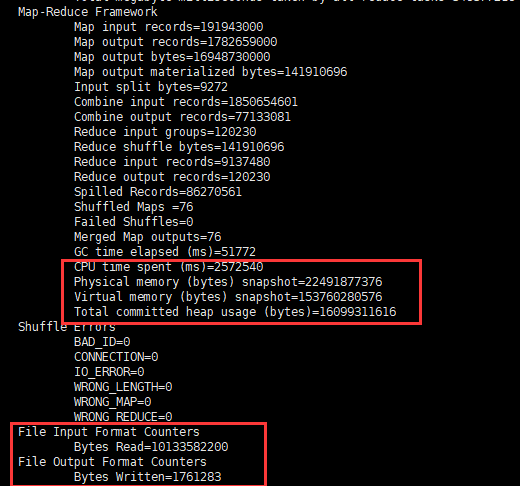
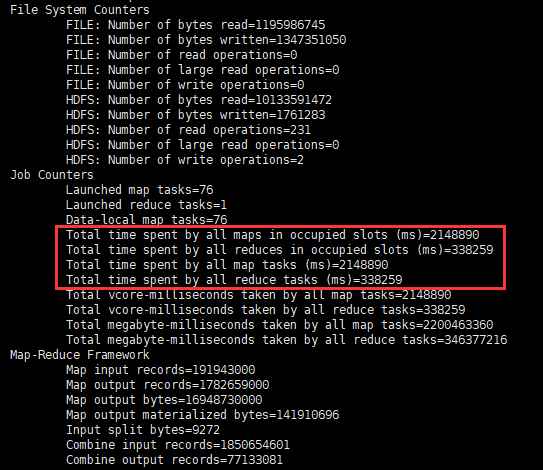




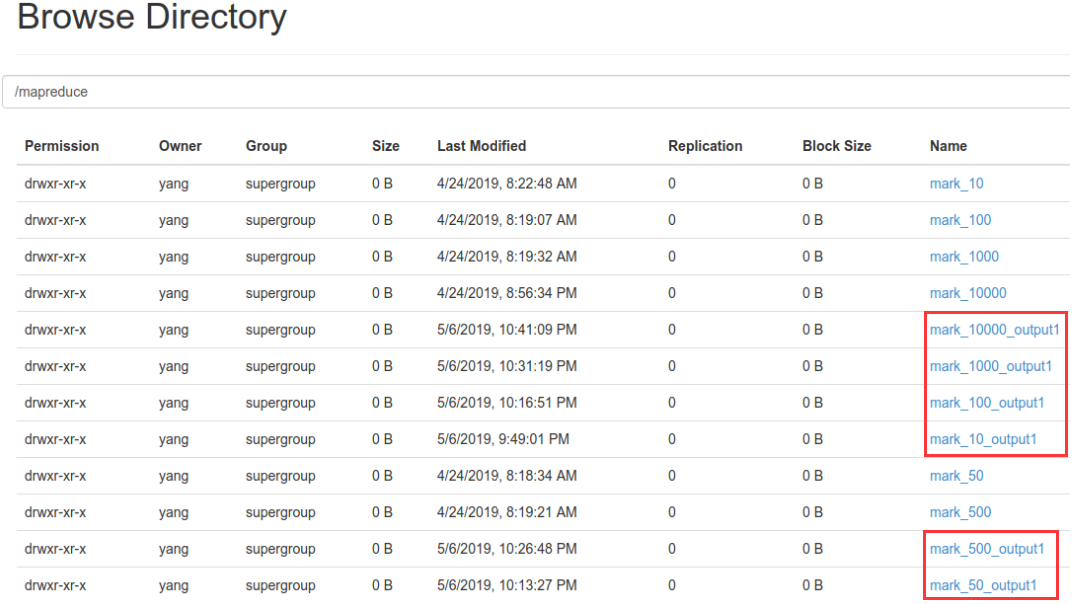


（7）10000M大小文本执行 Map、Reduce 时间及文件大小输入输出结果图：

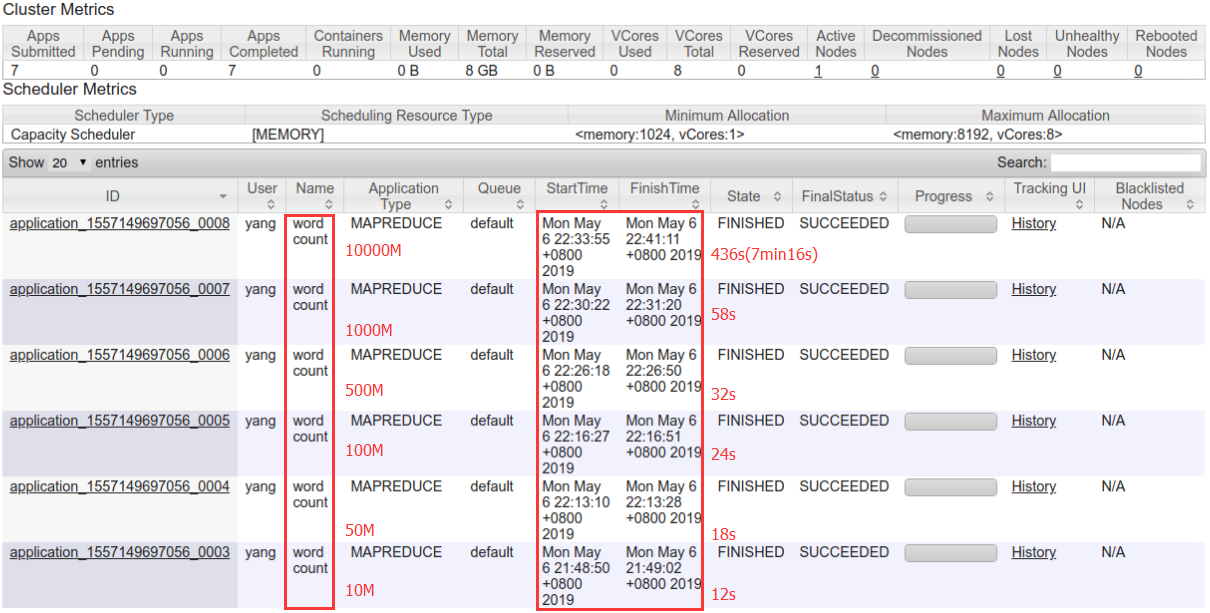




（8）10M、50M、100M、500M、100M、10000M文本执行生成输出图：

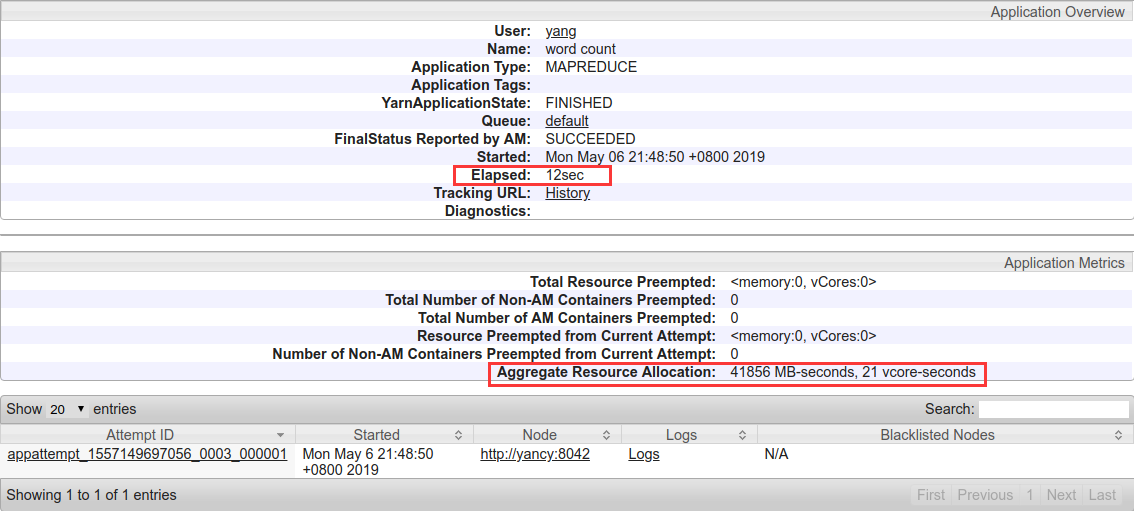


（9）10M、50M、100M、500M、100M、10000M文本执行MapReduce各自时耗：

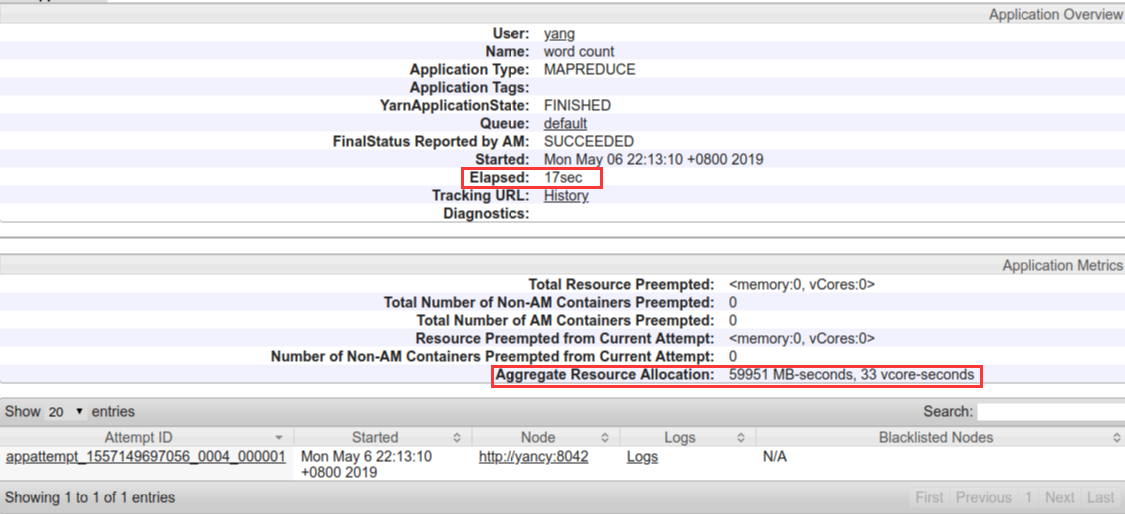


* + 1. 资源分配量

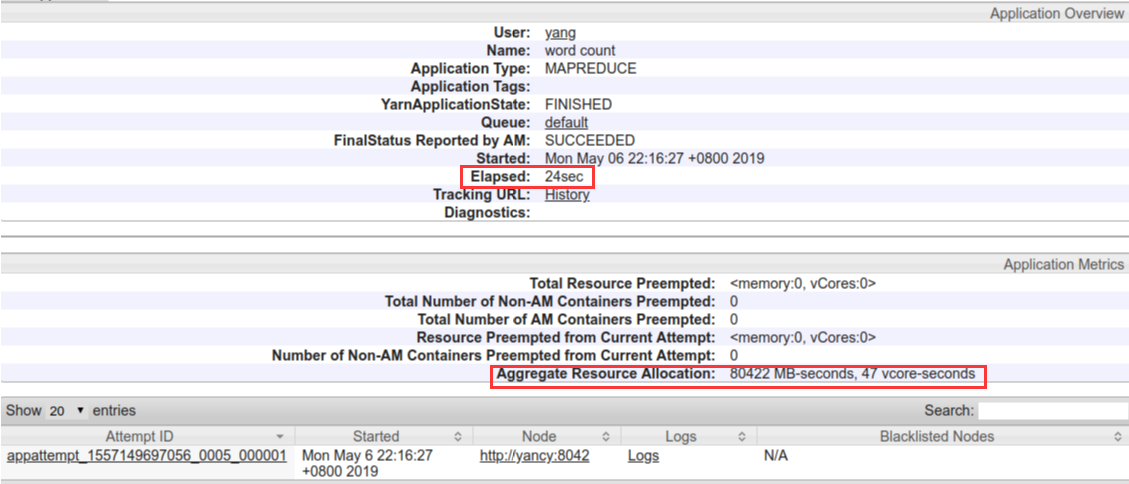
（1）10M文本执行 MapReduce 耗时及资源分配：



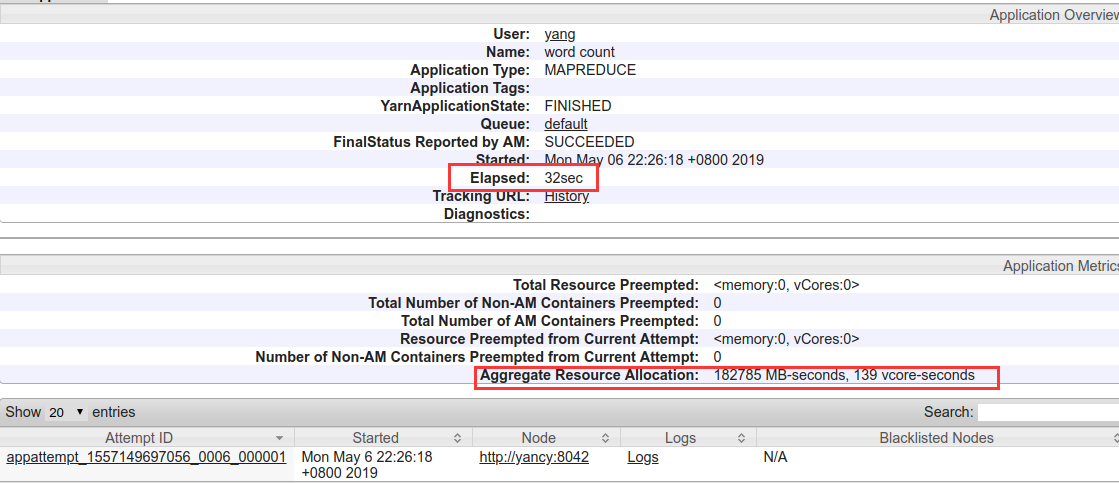
（2）50M文本执行 MapReduce 耗时及资源分配：



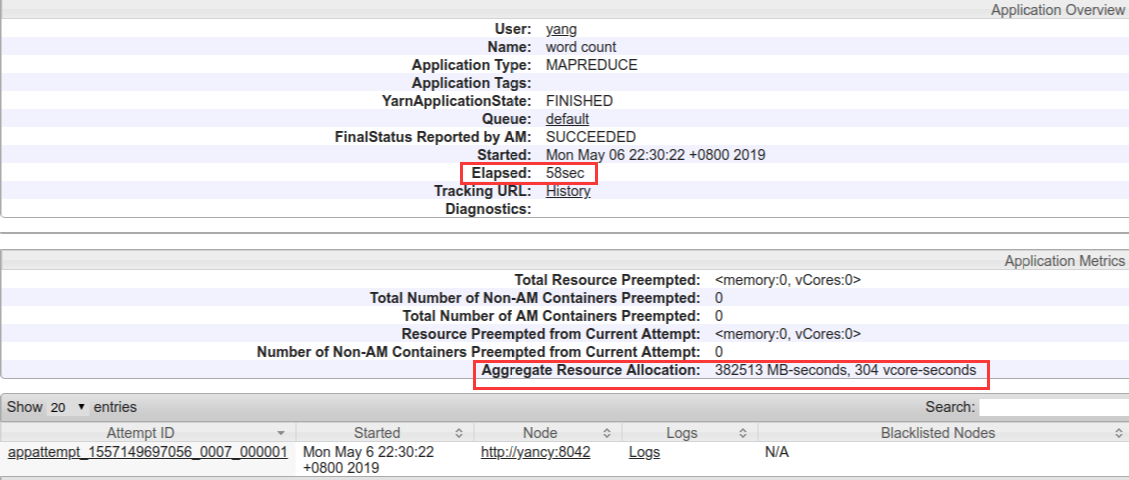
（3）100M文本执行 MapReduce 耗时及资源分配：



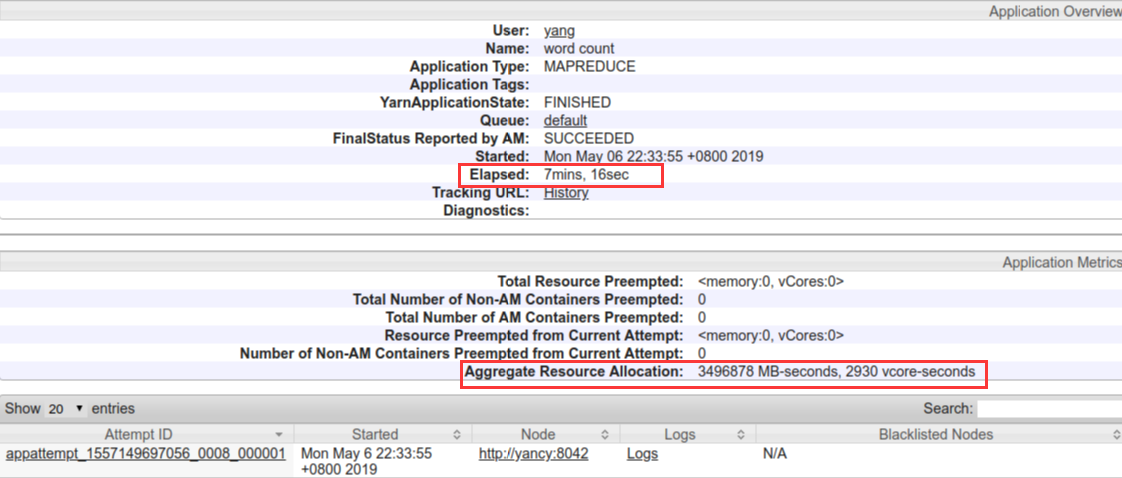
（4）500M文本执行 MapReduce 耗时及资源分配：



（5）1000M文本执行 MapReduce 耗时及资源分配：

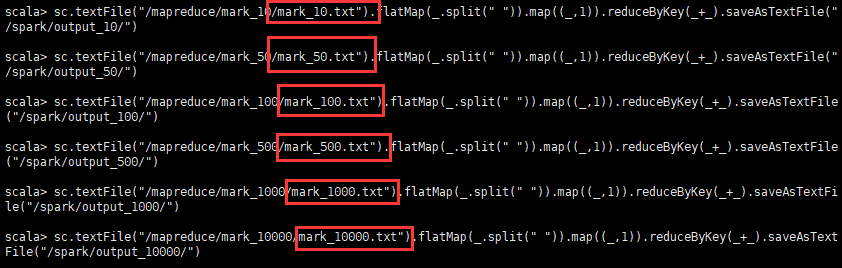


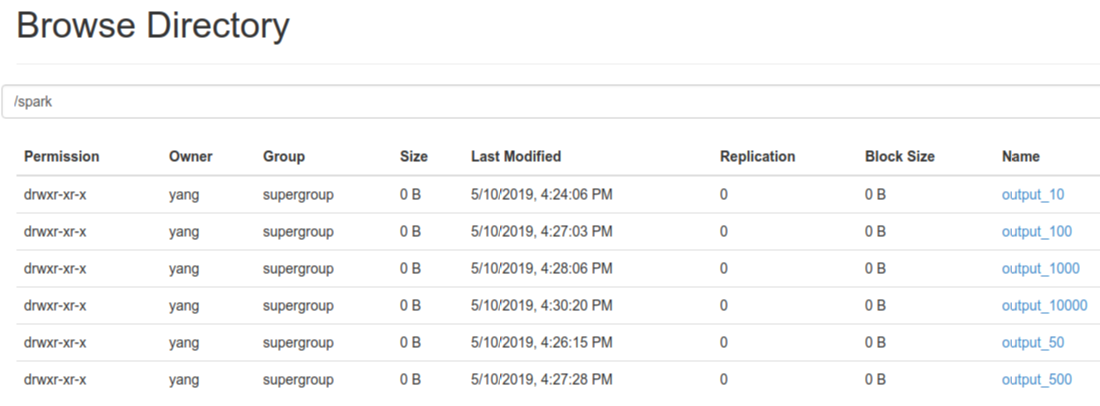
（6）10000M文本执行 MapReduce 耗时及资源分配：



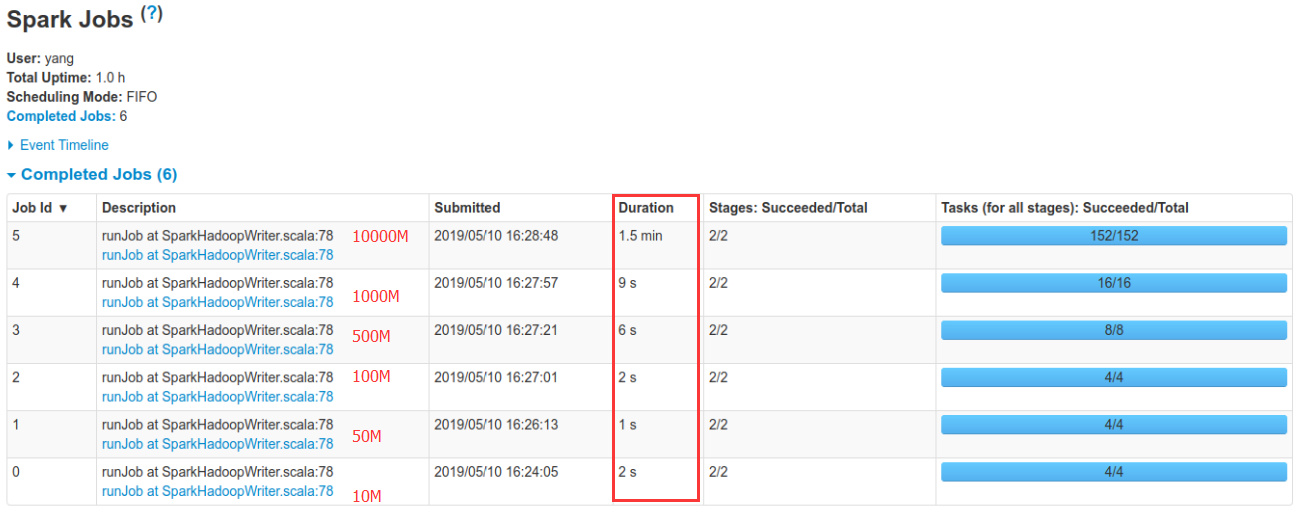
* 1. Spark执行效率
     1. 运行时间

（1）10M、50M、100M、500M、100M、10000M大小文本执行各自输出结果图：

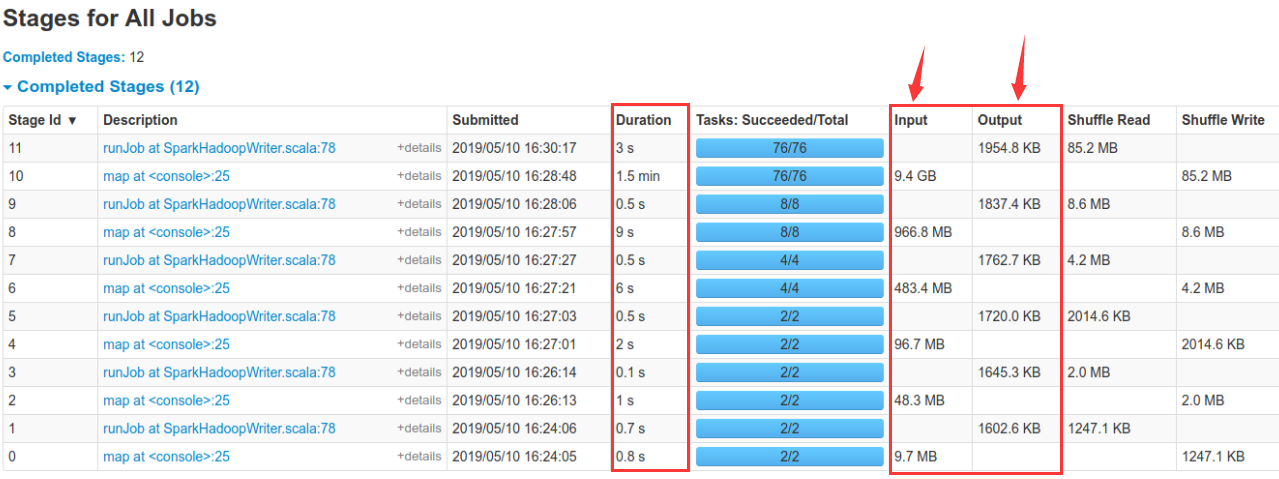




（2）10M、50M、100M、500M、100M、10000M大小文本运行各自总时耗：

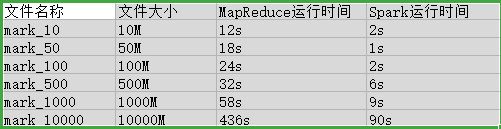


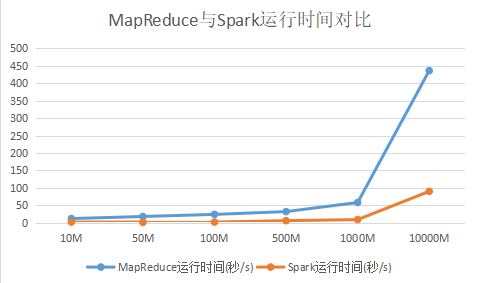
* + 1. 输出文本大小



* 1. 两者结果对比

MapReduce与Spark在10M、50M、100M、500M、100M、10000M大小文本执行各自运行消耗时间对比图：





1. 从软件体系架构角度解释分析实验结果
   1. 运行时间差异

由上小节实验可得出，MapReduce 整体的运行时间要大于Spark，当文本大小较小时（小于500M），MapReduce与Spark上的运行时间差异不明显，而随着文本大小的增大（大于500M），MapReduce与Spark上的运行时间差距越来越明显。

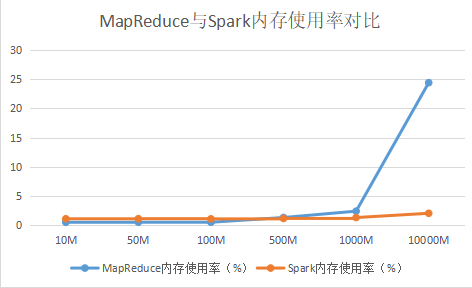
这是由于Hadoop 生态圈包含内容较多，每次任务启动时首先需要加载所需的资源包，然后缓慢地发起任务，所以导致整体计算时间长、性能差。而Spark的计算速度快是因为Spark 把中间数据放到内存中，迭代运算效率高。MapReduce 在shuffle的时候，计算结果需要保存到磁盘上，这样会影响整体速度，而Spark利用RDD技术，计算在内存中进行。且Spark 支持 DAG 图的分布式并行计算的编程框架，减少了迭代过程中数据的落地，提高了处理效率。

* 1. CPU使用率

由上小节实验得出，当文本大小在500M左右时，MapReduce占用的CPU资源基本处于饱和。而Spark的CPU使用率并没有伴随着文本文件增大而大幅度上涨。

* 1. 内存使用率

由上小节实验得出，随着文本数据的增大，MapReduce 慢慢需要占用较多的内存，而Spark 随着数据量增大而内存需求不明显。



* 1. 容错性

Spark 容错性比MapReduce 高。Spark 引进了弹性分布式数据集RDD，它是分布在一组节点中的只读对象集合，这些集合是弹性的，如果数据集一部分丢失，则可以对它们进行重建。另外在RDD 计算时可以通过 CheckPoint 来实现容错。

* 1. 通用性

MapReduce只提供了Map和Reduce两种操作，而Spark 提供的数据集操作类型有很多，大致分为转换操作（Transformations）和行动操作（Actions）两大类，Spark 更加通用。

由于Hadoop生态圈庞大，对大数据支持性很好，如果需要处理的数据量大，并且用户希望计算出的数据可以被存储和二次计算或数据挖掘，那么可以考虑MapReduce。Spark由于架构层面设计不同，所以对于 CPU、内存的使用率一直保持较低状态，它可以用于海量数据分析等用途。

1. 实验总结

MapReduce 采用了多进程模型，虽然在 server 端二者都采用了相同的并发模型，但是在任务级别上，MapReduce 的多进程模型会消耗过多的启动时间，在运行低延迟类型的任务上表现不如 Spark。

而Spark采用了多线程模型，Spark 可以用Scala编程，Scala适合并行计算，与Java相比，极大的减少代码量。Spark计算速度胜于MapReduce 还在于中间结果是缓存在内存而不是直接写入到磁盘。MapReduce 每次计算先写磁盘，下次计算先从磁盘读，计算结果再写磁盘，如此往复。Spark为每个应用程序在worker上开启一个进程，而一个Job中的Task会在同一个线程池中运行，而MapReduce的计算模型是每个Task(Mapper或者Reducer）都是一个单独的进程，启动停止进程非常昂贵，同时，进程间的数据共享也不能基于内存，只能是HDFS。Spark克服了MapReduce的这些不足，不过其带来的缺点是难以细粒度地控制每个任务的占用资源。

在数据的读取上，如果任务复杂的话，MapReduce 在读写磁盘文件时会在 IO 上花费大量的时间，而 Spark 在这一点上消除了冗余的 HDFS 读写，速度更快。MapReduce 每一次 shuffle 必将连接着一次完成 Map 和 Reduce 操作，而 Spark 基于 RDD 提供了丰富的算子操作，避免了冗余的 Map 和 Reduce 阶段。MapReduce每次启动一个任务就要启动一次基于进程的JVM，而Spark基于线程的机制，只会在启动执行器时启动一次JVM，执行任务时复用执行器中的线程。由于启动JVM时要耗费大量的时间，如果任务数一多，MapReduce频繁地启动JVM会耗费大量的时间。在上节中的实验环节，我们也可以看到这一点。

总的来说，现在的大数据应用中对更复杂的多重处理和低延迟的交互式查询需求日益增长，MapReduce计算模型在上述两类任务上因为其架构的原因，执行效率并不快，Spark则在某种程度上克服了这些困难，其基于内存的计算提升了计算速度，基于RDD结构提升了容错性能，提供了各种计算框架。不过其也有一些局限和缺点，主要体现在耗费的内存资源过大、稳定性不足、无法细粒度地进行资源分配等。

1. 实验遇到的问题
2. 问题一：Namenode，Datanode未正常启动。

解决：由于多次初始化Damenode 导致ID不对应。关闭hadoop，删除data,name,logs,tmp目录下所有文件，然后 hadoop namenode -format 重启初始化，最后开启hadoop。

1. 问题二：在hadoop安装完后，依次执行hadoop namenode -format，start-dfs.sh，start-yarn.sh之后，打开浏览器输入http://localhost:50070/无反应，查了网上很多说法都没有解决，最后发现是因为电脑连接了VPN。

解决：直到关掉VPN之后就可以了。

1. 问题三：显示错误信息，hedulerImpl: Initial job has not accepted any resources; check your cluster UI to ensure that workers are registered and have sufficient resources

19/05/07 12:56:49 WARN scheduler.TaskSchedulerImpl: Initial job has not accepted any resources; check your cluster UI to ensure that workers are registered and have sufficient resources。

解决：执行语句少了一个斜杠，即为saveAsTextFile("/spark/output/")。