实验：文档倒排索引

班级： 07112002 学号： 1120201316 姓名： 金雅各

班级： 学号： 姓名：

班级： 学号： 姓名：

班级： 学号： 姓名：

一、实验要求

文档倒排索引实验，是通过搭建Hadoop分布式集群，利用MapReduce算法，实现对海量数据文件的倒排索引，并将结果存入Hbase数据库中。

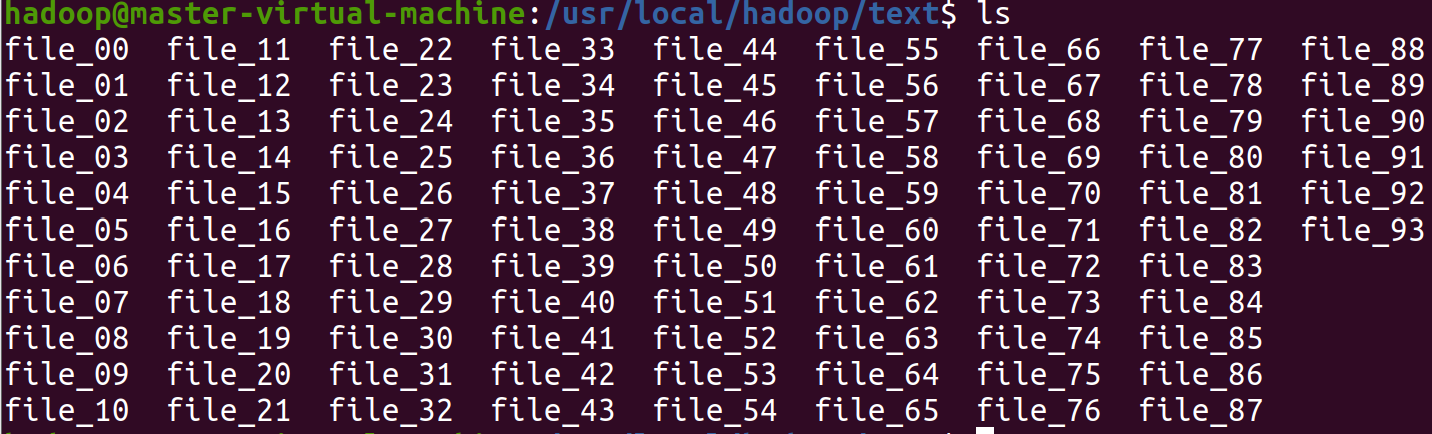
目标就是通过对Hadoop分布式框架以及Hbase架构的学习，能够在Linux操作系统环境下搭载自己的Hadoop，Hbase分布式框架，实现对海量数据的倒排索引，并将MapReduce后的数据直接插入到Hbase所建立的表中。

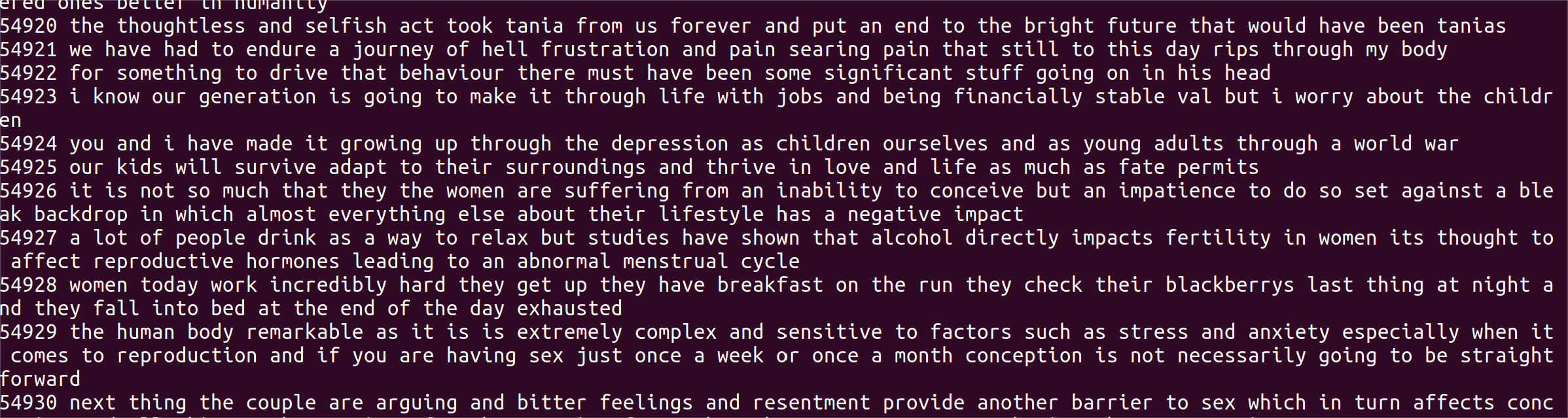
二、数据准备与说明

实验的原始文件：压缩文件500MB，解压后1.32GB

按照每100000行为一个文件进行划分，共划分出93个文件，文件名格式为file\_

划分代码为：split -l 100000 sentences.txt -d -a 2 file\_

划分出的文件为：

源数据内容为带行号的英文文本，每个单词间用空格分割：

三、环境简要说明

Linux操作系统，在VMware16pro 上运行ubuntu-20.04.5虚拟机。

共一台虚拟机hadoop

Jdk版本：java-8-openjdk-amd64

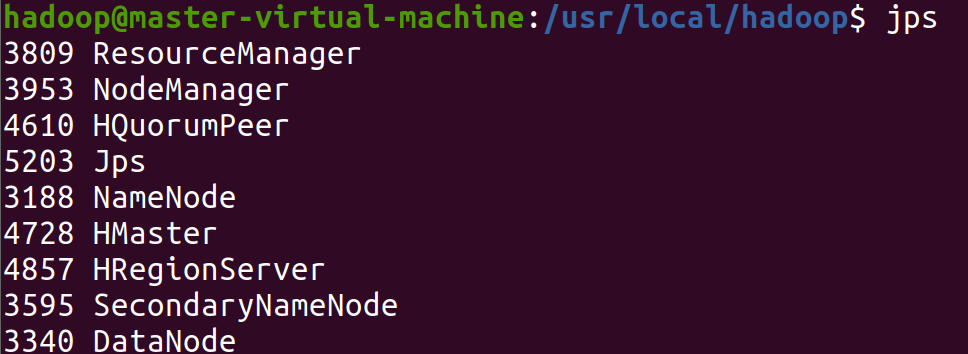
路径：/usr/lib/jvm/java-8-openjdk-amd64

(直接sudo apt-get install openjdk-8-jdk安装就好)

**Hadoop-3.3.4**

配置：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 主机名 | IP | HDFS | YARN | Hbase | Zookeeper |
| hadoop | 192.168.253.131 | NameNode  DataNode  SecondaryNameNode | NodeManager  ResourseManager | HRegionServer  Hmaster | HQuorumPeer |

****

四、安装、配置及其过程。

为虚拟机新建立一个hadoop对象: sudo useradd -m hadoop -s /bin/bash

为新用户设置登陆密码: sudo passwd hadoop

为新用户添加管理员权限: sudo adduser hadoop sudo

登录hadoop 用户后 更新系统软件：

更新软件列表：sudo apt-get update

更新软件：sudo apt-get upgrade

安装ssh-server：sudo apt-get install openssh-server

配置 ssh 无密登录：

进入ssh目录：cd ~/.ssh/

生成ssh密匙：ssh-keygen -t rsa （提示默认按回车即可）

将密匙加入授权中：cat ./id\_rsa.pub >> ./authorized\_keys

此后可以直接使用 ssh localhost 免密登录

安装 jdk8：sudo apt-get install openjdk-8-jdk

查看java是否安装成功：java -version

配置java环境：gedit ~/.bashrc

在最前面加入：export JAVA\_HOME=/usr/lib/jvm/java-8-openjdk-amd64

使环境变量生效：source ~/.bashrc

使用：java -version 查看jdk的版本 whereis java 查看jdk的路径

下来进行 Hadoop、Hbase的安装 均将安装包移动到 /usr/local/ 路径下进行解压

并改名为 hadoop、hbase

1. Hadoop

解压文件到 /usr/local/ 目录下 改名为 hadoop

然后修改文件的权限：sudo chown -R hadoop ./hadoop (hbase)

（将文件置为 hadoop 用户所有）

查看hadoop是否安装成功：

进入目录：cd hadoop

执行指令：./bin/hadoop version

进行Hadoop 伪分布式的配置

gedit ./etc/hadoop/core-site.xml

<configuration>

<property>

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

<!-- hadoop 存储路径-->

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

<description>A base for other temporary directories.</description>

</property>

<property>

<name>fs.defaultFS</name>

<!-- 告诉 NN 在那个机器，NN 使用哪个端口号接收客户端和 DN 的RPC请求-->

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

</property>

</configuration>

gedit ./etc/hadoop/hdfs-site.xml

<configuration>

<property>

<!-- 设置默认副本数量为1-->

<name>dfs.replication</name>

<value>1</value>

</property>

<property>

<!-- 主namenode 信息存储目录-->

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

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

</property>

<property>

<!-- 从namenode信息存储目录 -->

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

<value>file:/usr/local/hadoop/tmp/dfs/namesecondary</value>

</property>

<property>

<!--datanode 存储位置-->

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

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

</property>

</configuration>

gedit ./etc/hadoop/yarn-site.xml

<configuration>

<property>

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

<value>mapreduce\_shuffle</value>

</property>

</configuration>

gedit ./etc/hadoop/mapred-site.xml

<configuration>

<property>

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

<value>yarn</value>

</property>

</configuration>

执行namenode格式化：./bin/hdfs namenode -format (当提示输入Y/N时，输入Y)

开启namenode、datanode：./sbin/start-dfs.sh

查看是否开启成功：jps（4个信息JPS,SecondaryNameNode,NameNode,DataNode）

开启yarn：./sbin/start-yarn.sh

查看是否开启成功：jps (新加ResourceManager与NodeManager)

查看网页端的信息：http://localhost:9870

伪分布读取的是HDFS上的数据使用时需要上传数据到HDFS上

在HDFS上创建目录：./bin/hdfs dfs -mkdir -p /data/input （创建了data/input路径的文件夹）

上传文件到该目录下：./bin/hdfs dfs -put text/\* /data/input

查看是否上传成功：./bin/hdfs dfs -ls /data/input

HDFS 退出安全模式：./bin/hdfs dfsadmin -safemode leave

--------------------------------------------------------------------------------------------------------

②Hbase

解压改名后授权：sudo chown -R hadoop ./hbase

查看是否解压成功：./bin/hbase version

配置Hbase:

# 配置JDK安装路径

export JAVA\_HOME=/usr/lib/jvm/java-8-openjdk-amd64

# 使用自带的ZooKeeper

export HBASE\_MANAGES\_ZK=true

gedit ./conf/hbase-site.xml

<configuration>

<property>

<!-- 同hadoop配置文件core-site.xml中fs.defaultFS的值 -->

<name>hbase.rootdir</name>

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

</property>

<property>

<!-- false是单机模式，true是分布式模式 -->

<!-- 当使用外置zookeeper时必须配置为true，仅在使用hbase自带zookeeper时配置为false -->

<name>hbase.cluster.distributed</name>

<value>true</value>

</property>

<property>

<!-- zookeeper的数据目录，同zookeeper配置文件zoo.cfg中dataDir的值，启动hbase的用户需对此目录有读写权限 -->

<name>hbase.zookeeper.property.dataDir</name>

<value>/home/hadoop/zookeeper</value>

</property>

</configuration>

启动hbase：./bin/start-hbase.sh

查看是否启动成功：jps （多三个结点HRegionServer、HQuorumPeer、HMaster）

关闭hbase：./bin/stop-hbase.sh

打开shell：./bin/hbase shell

五、倒排索引及其算法

程序包含四个类InvertedDriver、InvertedMapper、 InvertedCombiner、和InvertedReducer；

其中InvertedDriver为主类，内包含初始化连接操作，创建Hbase命名空间和表的函数；

初始化Configuration=HBaseConfiguration.create();

configuration.set("hbase.rootdir","hdfs://localhost:9000/hbase");

初始化Connection用以连接Hbase以创建命名空间和表

connection = ConnectionFactory.createConnection(configuration);

获取Job Job job = Job.getInstance(configuration, "Inverted Index");

对job进行配置：

job.setJarByClass(InvertedDriver.class);

job.setMapperClass(InvertedMapper.class);

job.setMapOutputKeyClass(Text.class);

job.setMapOutputValueClass(Text.class);

job.setReducerClass(InvertedReducer.class);

FileInputFormat.addInputPath(job, new Path(args[0]));

TableMapReduceUtil.initTableReducerJob(nameSpace+":"+tableName, InvertedReducer.class,job);

job.setCombinerClass(InvertedCombiner.class);

InvertedMapper为Mapper类，继承自Mapper<Object,Text,Text,Text>

将传入的文本信息进行处理并发送给Combiner类

对接收的文本信息按照行进行处理；

先使用FileSplit得到该文本的文件名->filename

再对文本信息line使用split进行分割得到每一个单词组strings

Strings=value.split(“ ”)

初始化要写入context的键值

newKey=strings[index]+“:”+filename

写入context的value值设为默认值1标志单词在文本中出现了一次

Context.write(newKey,1)

接收的键值对<key,value>=<行偏移量，文本信息(87 this is a test)>

发送的键值对<key,value>=<letter:file\_name,1>

InvertedCombiner为优化类，继承自Reducer<Text, Text, Text, Text>

对传入的键值对按照Key值进行分组后统计，单词出现的次数

该类重载reduce函数，对传入的键值对按照key值进行分组，得到key与values(传入value的集合)

对每一个key 我们对values进行遍历，计算出key出现的次数sum

Sum=0;

For value in values:

Sum+=1;

Index<-key.find(“:”);

写入context的key值为单词，即对传入的Key取子串得到单词newKey=key.sub(0,index)

写入context的value值为filename+“:”+sum;

newValue=filename+“:”+sum

Context.write(newKey,newValue)

接收的键值对<key,value>=<letter:file\_name,{1,1,1}>

发送的键值对<key,value>=<letter,file\_name:3>

InvertedReducer为最终的Reduce类，继承自TableReducer<Text, Text, ImmutableBytesWritable>,我们需要将传入的键值对依照key值作为Hbase的rowKey,文件名作为Column，数目作为value的形式插入到Hbase中

该类重载父类reduce函数，对传入的键值对按照key值进行分组得到<key,values（key值相同的value集合）>

初始化Put对象,设置key为rowKey值

Put put =new Put(key.getBytes())

对得到的values集合，我们需要进行split以得到文件名和数目

For value in values

Index<-value.find(“:”)

Filename=value.sub(0,index)

Count=value.sub(index+1)

Put.addColumn(“File”，Filename,count)

将Put写入context中(即插入在InvertedDriver中指定的Hbase表中)

Context.write(null,put)

传入的键值对<key,value>=<letter,{file\_name:3,file\_name2:2}>

写入Hbase的信息rowKey=letter

列族为File

列与值信息为 File:file\_name 3 File:file\_name2 2

六、程序的编译、打包、运行及作业的监控或结果查看。

运行时需要打开namenode,yarn 以及hbase

将jar包和预处理过的文件一起放到放到hadoop路径下

然后执行：./bin/hdfs dfs -mkdir -p /data/input

./bin/hdfs dfs -put text/\* /data/input

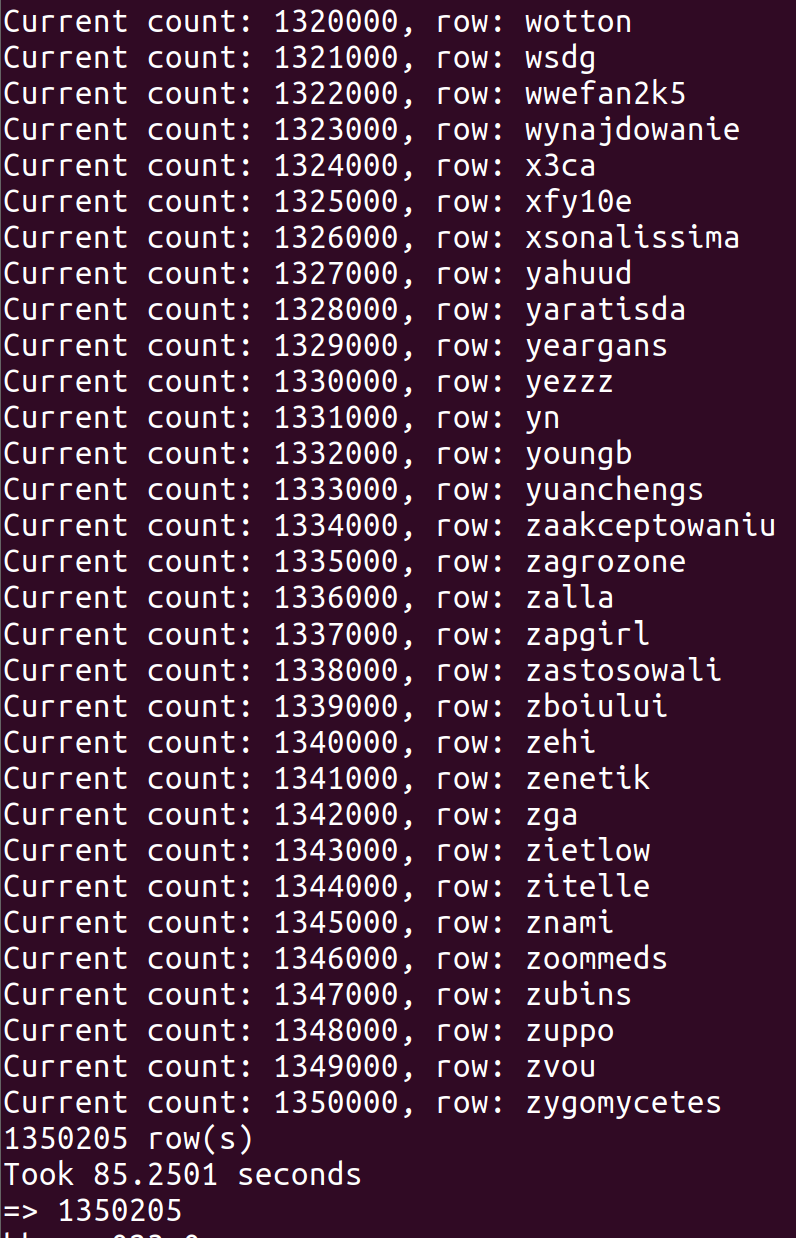
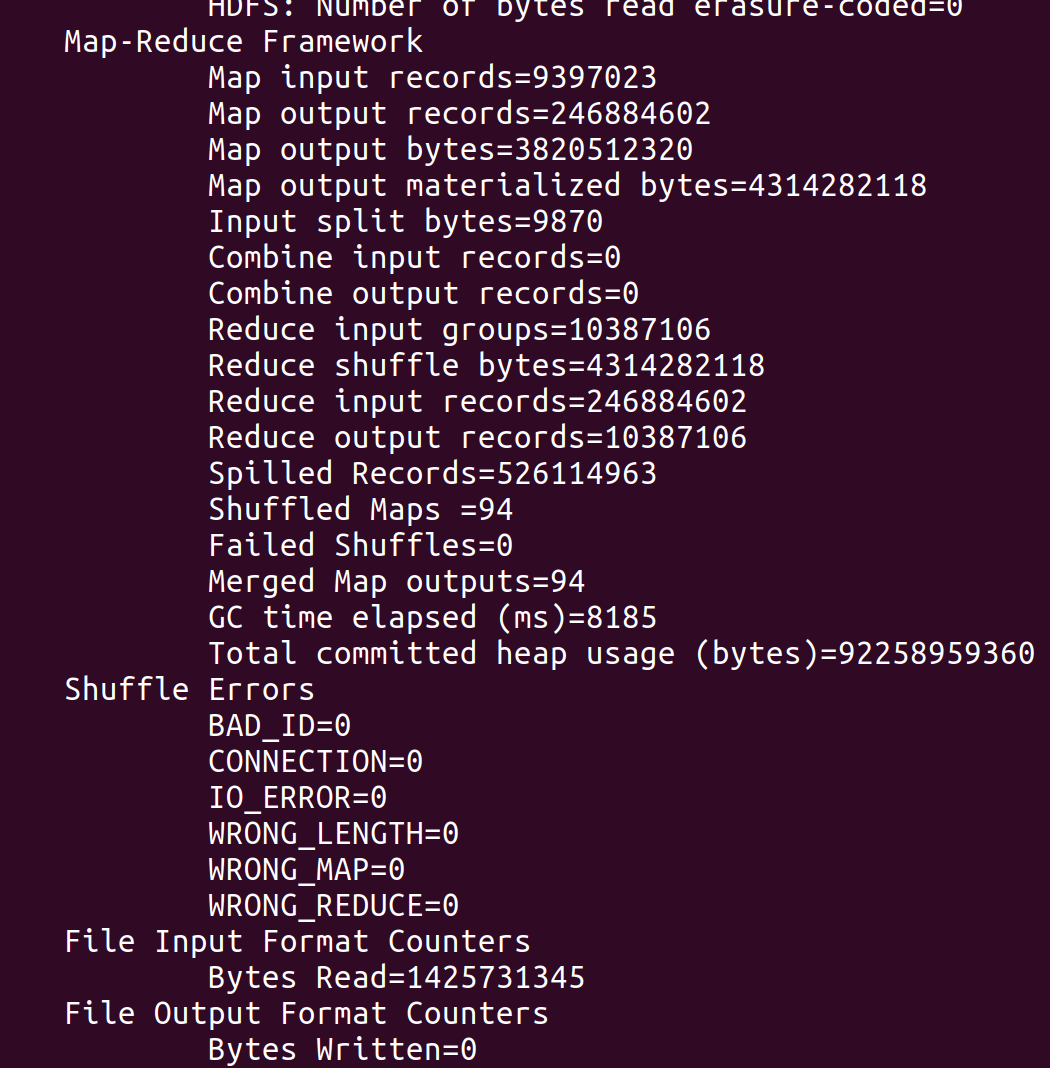
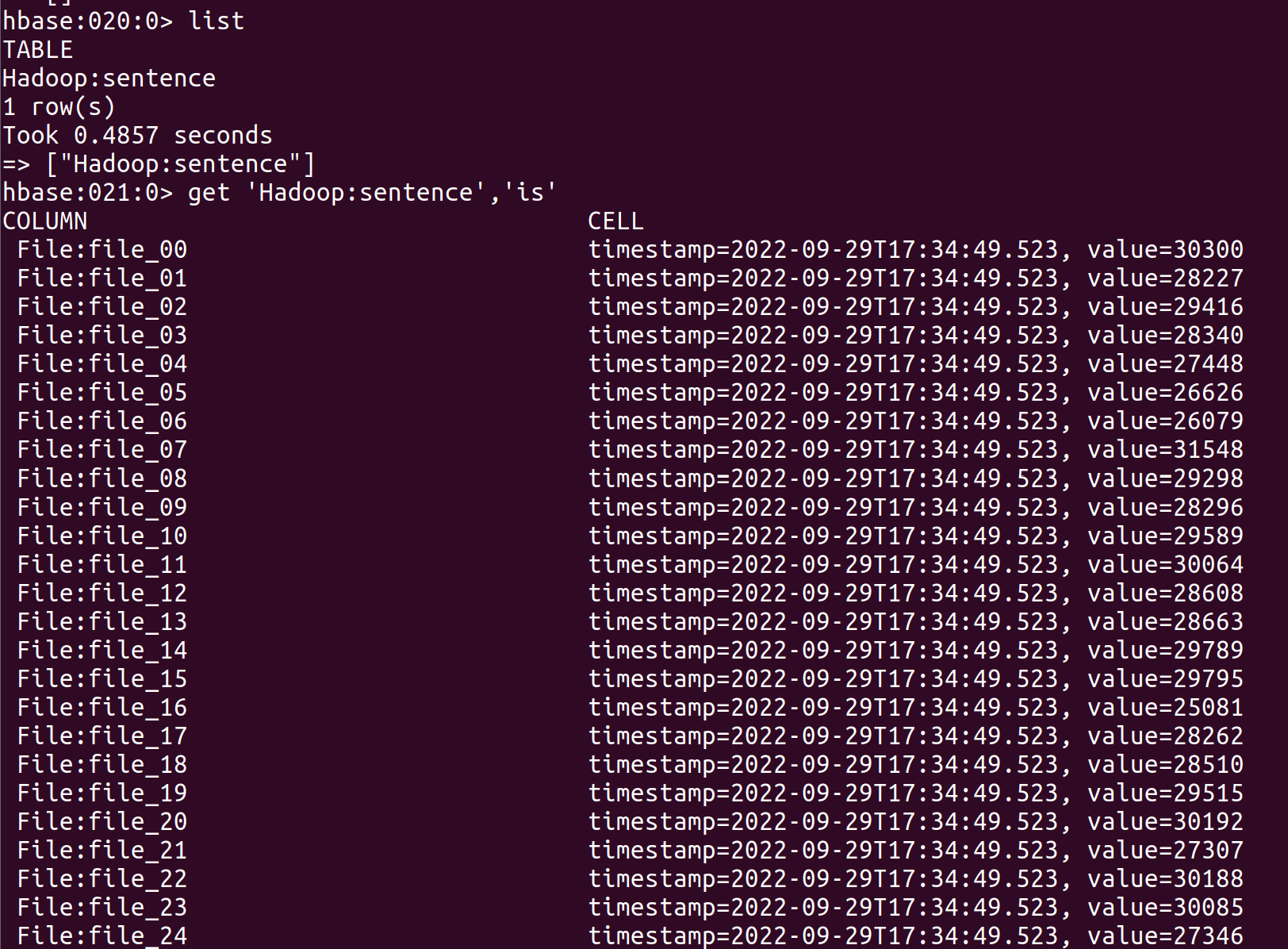
./bin/hadoop jar HadoopProject\_2.jar /data/input Hadoop:sentence

七、故障诊断。

在InvertedDriver中配置Combiner类时，由于Hbase版本的变化， TableMapReduceUtil.initTableReducerJob会检查Combiner类是否继承自TableReduce类若是继承自Reduce就会报找不到Combiner类的异常；需要把job.setCombinerClass(InvertedCombiner.class);放到TableMapReduceUtil.initTableReducerJob之后

1. 实验结果与分析。

成功将MapReduce后的文档上传到Hbase中，用时大概20min左右。



1. 总结

本此文档倒排索引基本功能都已实现，且性能还算可以（20min左右处理完毕）。但一是未能配置完全分布式环境，而是在伪分布式环境下完成本实验。而是实现代码中未能添加Combiner进行优化（原因是加入Combiner后在伪分布环境下的性能还不如没有Combiner优化的代码）。通过本次学习，我们对大数据架构有了充分的了解。文档倒排索引的实现，使我们对大数据架构搭建环境，实现算法，以及调试运行有了熟悉的认知。本次实验虽仍有遗憾，但实现的算法性能大体上还算可以。

最后，感谢老师对于我们小组实验过程遇到困难的帮助以及老师课程的教学，谢谢老师！