**大数据系统开发实践项目---实验报告**



**作 者： 黄昱欣（1120200807） 郑子帆（1120200822）**

**王天泽（1120200718） 张 驰（1120200837）**

**安禹堃（1120200860）**

**组 长： 郑子帆（1120200822）**

**所在学院： 计算机学院 计算机科学与技术专业 07112002**

**教 师： 郭贵锁**

2022年9月21日

目录

[1 实验要求 4](#_Toc115450900)

[1.1 实验内容 4](#_Toc115450901)

[1.2 实验目标 4](#_Toc115450902)

[2 数据准备与说明 5](#_Toc115450903)

[2.1 实验数据说明 5](#_Toc115450904)

[2.2 数据处理 5](#_Toc115450905)

[3 环境简要说明 6](#_Toc115450906)

[3.1 应用程序信息 6](#_Toc115450907)

[3.2 虚拟机信息 6](#_Toc115450908)

[4 安装、配置过程 8](#_Toc115450909)

[4.1 模板虚拟机配置 8](#_Toc115450910)

[4.2克隆虚拟机 9](#_Toc115450911)

[4.3 进一步配置 11](#_Toc115450912)

[4.3.1在hadoop102上安装JDK 11](#_Toc115450913)

[4.3.2在hadoop102上安装Hadoop 11](#_Toc115450914)

[4.3.3完全分布式配置准备工作 12](#_Toc115450915)

[4.3.4集群配置 12](#_Toc115450916)

[4.3.5发起集群 14](#_Toc115450917)

[4.4完全分布式zookeeper安装及配置 15](#_Toc115450918)

[4.4.1 Zookeeper集群安装 15](#_Toc115450919)

[4.5完全分布式HBase安装部署 16](#_Toc115450920)

[5 倒排索引及其算法 18](#_Toc115450921)

[5.1 原理介绍 18](#_Toc115450922)

[5.2 Mapper 18](#_Toc115450923)

[5.3 Reducer 20](#_Toc115450924)

[6 实际运行 22](#_Toc115450925)

[6.1 程序的编译、打包、运行 22](#_Toc115450926)

[6.2 作业查看(JobHistory) 25](#_Toc115450927)

[6.3 最终结果展示 27](#_Toc115450928)

[7 故障诊断 30](#_Toc115450929)

[7.1 配置过程中的问题 30](#_Toc115450930)

[7.2 程序运行中的问题 30](#_Toc115450931)

[7.3 其他问题 30](#_Toc115450932)

[8 实验结果与分析 31](#_Toc115450933)

[8.1 内容分析 31](#_Toc115450934)

[8.2 性能分析 31](#_Toc115450935)

[9 总结 32](#_Toc115450936)

# 1 实验要求

## 1.1 实验内容

倒排列表用来记录有哪些文档包含了某个单词。一般在文档集合里会有很多文档包含某个单词，每个文档会记录文档编号（DocID），单词在这个文档中出现的次数（TF）。这样与一个文档相关的信息被称做倒排索引项（Posting），包含这个单词的一系列倒排索引项形成了列表结构，这就是某个单词对应的倒排列表。下图是倒排列表的示意图。



实现文档的倒排索引，即运用MapReduce算法计算，构建一个倒排索引，将倒排索引存储在HBase中。数据在教学qq群中下载，通过MapReduce算法构建倒排索引，将其存储在HBase数据库中，实现对于一个单词可以找到它所在的文件目录和出现的次数。

## 1.2 实验目标

在实验中，根据课上老师所讲理论内容，结合实践进一步学习Hadoop、HBase的特征和功能，以及MapReduce计算框架的原理和功能。分别完成在伪分布式环境、完全分布式环境下的基于Hadoop、HBase构建的倒排索引。

# 2 数据准备与说明

## 2.1 实验数据说明

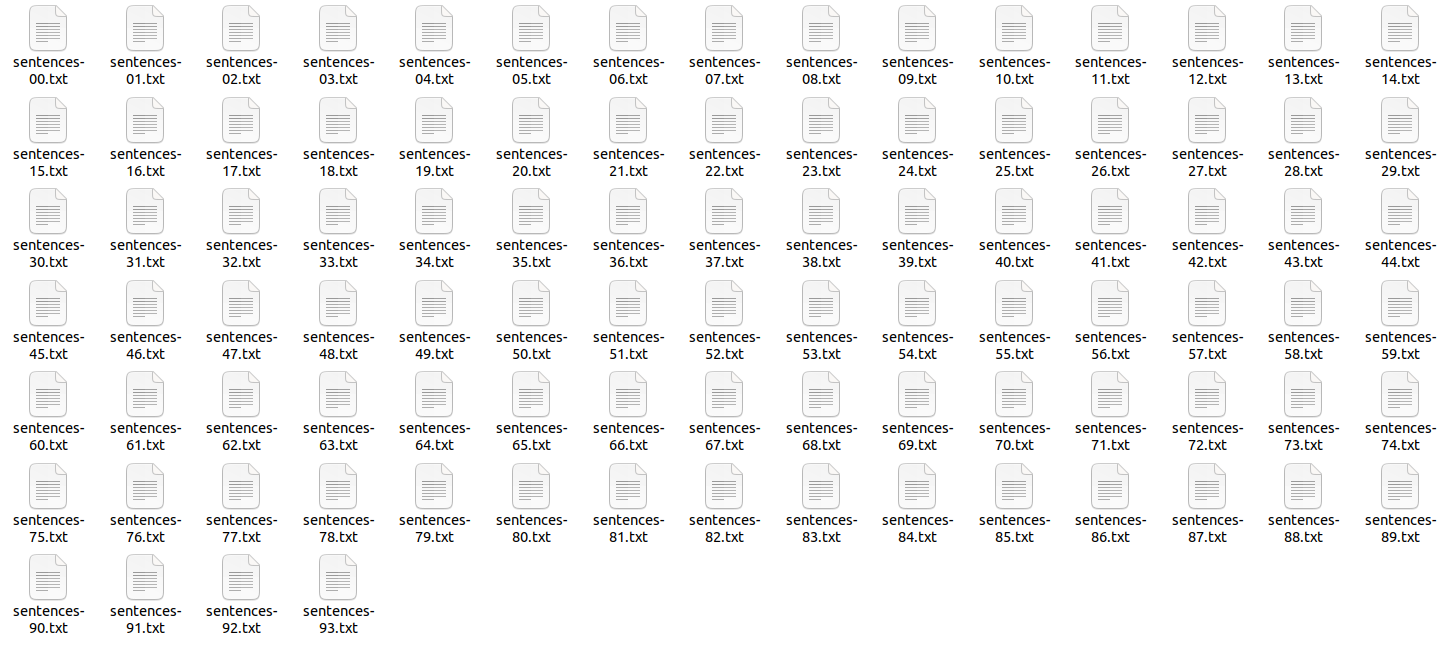
数据来自教学qq群，解压后sentences.txt文件大小1.32G。通过文本编辑器打开文件，可以看到文件一共有9397023行，每行由一个行号（0-9397022顺序排列）和一个长文本构成，相邻单词之间用空格分离。数据只由a~z二十六个小写字母和0~9十个组成，没有标点符号和特殊符号等字符。

## 2.2 数据处理

将sentences.txt文件下载至一台虚拟机的桌面。打开终端，执行以下指令，完成input文件夹的建立，并将sentences.txt文件按每10w行进行划分，分成94个文件sentences-00.txt~sentences-93.txt，放于input文件夹下。

1. mkdir input
2. cd input
3. split -d -a2 -l100000 --additional-suffix=.txt /home/sadgde/Desktop/sentences.txt sentences-
4. cd ..
5. hdfs dfs -put input

结果如下图所示：



# 3 环境简要说明

## 3.1 应用程序信息

虚拟机软件：VMware Workstation Pro 14

操作系统：CentOS 7（虚拟机）

Hadoop版本：hadoop 3.1.3

JDK版本：jdk 1.8.0\_212

HBase版本：hbase-2.4.11

远程连接软件：XShell 7

文件传输软件：Xftp 7

## 3.2 虚拟机信息

完全分布式共配置了3台Linux虚拟机 每台内存4G 硬盘50G

虚拟机1:

主机名：hadoop102

IP：192.168.10.102

作用：NameNode

DataNode

NodeManager

HMaster

HRegionServer

虚拟机2:

主机名：hadoop103

IP：192.168.10.103

作用：DataNode

ResourceManager

NodeManager

HRegionServer

虚拟机1:

主机名：hadoop104

IP：192.168.10.104

作用：SecondaryNameNode

DataNode

NodeManager

HRegionServer

# 4 安装、配置过程

## 4.1 模板虚拟机配置

* 1. 安装模板虚拟机，IP地址192.168.10.100、主机名称hadoop100、内存4G、硬盘50G

安装时注意安装桌面标准版，不要安装最小系统版，否则还需自行安装net-tool和vim

* 1. Hadoop100虚拟机配置要求

安装epel-lease

注：Extra Packages for Enterprise Linux是为“红帽系”的操作系统提供额外的软件包，适用于RHEL、CentOS和Scientific Linux。相当于是一个软件仓库，大多数rpm包在官方 repository 中是找不到的）

[root@hadoop100 ~]# yum install -y epel-release

* 1. 关闭防火墙，关闭防火墙开机自启

[root@hadoop100 ~]# systemctl stop firewalld

[root@hadoop100 ~]# systemctl disable firewalld.service

* 1. 创建用户，用户名为anyukun并设置密码
  2. 配置anyukun用户具有root权限

修改/etc/sudoers文件，在%wheel这行下面添加一行，如下所示：

## Allow root to run any commands anywhere

root ALL=(ALL) ALL

## Allows people in group wheel to run all commands

%wheel ALL=(ALL) ALL

anyukun ALL=(ALL) NOPASSWD:ALL

* 1. 在/opt目录下创建文件夹，并修改文件夹的所有者和所属组为anyukun用户

[root@hadoop100 ~]# mkdir /opt/module

[root@hadoop100 ~]# mkdir /opt/software

[root@hadoop100 ~]# chown anyukun:anyukun/opt/module

[root@hadoop100 ~]# chown anyukun:anyukun/opt/software

* 1. 卸载虚拟机自带的JDK
  2. 重启虚拟机

## 4.2克隆虚拟机

1. 利用模板机hadoop100，克隆三台虚拟机：hadoop102 hadoop103 hadoop104
2. 修改克隆机IP，以下以hadoop102为例子
   * 1. 修改克隆虚拟机的静态IP

[root@hadoop100 ~]# vim /etc/sysconfig/network-scripts/ifcfg-ens33

改成

DEVICE=ens33

TYPE=Ethernet

ONBOOT=yes

BOOTPROTO=static

NAME="ens33"

IPADDR=192.168.10.102

PREFIX=24

GATEWAY=192.168.10.2

DNS1=192.168.10.2

* + 1. 查看Linux虚拟机的虚拟网络编辑器，编辑->虚拟网络编辑器->VMnet8



* + 1. 查看Windows系统适配器VMware Network Adapter VMnet8的IP地址
    2. 保证Linux系统ifcfg-ens33文件中IP地址、虚拟网络编辑器地址和Windows系统VM8网络IP地址相同。

1. 修改克隆机主机名，以下以hadoop102举例说明
2. 修改主机名称

[root@hadoop100 ~]# vim /etc/hostname

hadoop102

1. 配置Linux克隆机主机名称映射hosts文件，打开/etc/hosts

[root@hadoop100 ~]# vim /etc/hosts

添加如下内容

192.168.10.100 hadoop100

192.168.10.101 hadoop101

192.168.10.102 hadoop102

192.168.10.103 hadoop103

192.168.10.104 hadoop104

192.168.10.105 hadoop105

192.168.10.106 hadoop106

192.168.10.107 hadoop107

192.168.10.108 hadoop108

1. 重启克隆机hadoop102
2. 修改windows的主机映像文件（hosts文件）

因为物理机操作系统为win10，所以要将文件先拷贝出来进行修改，然后再进行覆盖。（文件路径C:\Windows\System32\drivers\etc）

向host文件中添加如下内容

192.168.10.100 hadoop100

192.168.10.101 hadoop101

192.168.10.102 hadoop102

192.168.10.103 hadoop103

192.168.10.104 hadoop104

192.168.10.105 hadoop105

192.168.10.106 hadoop106

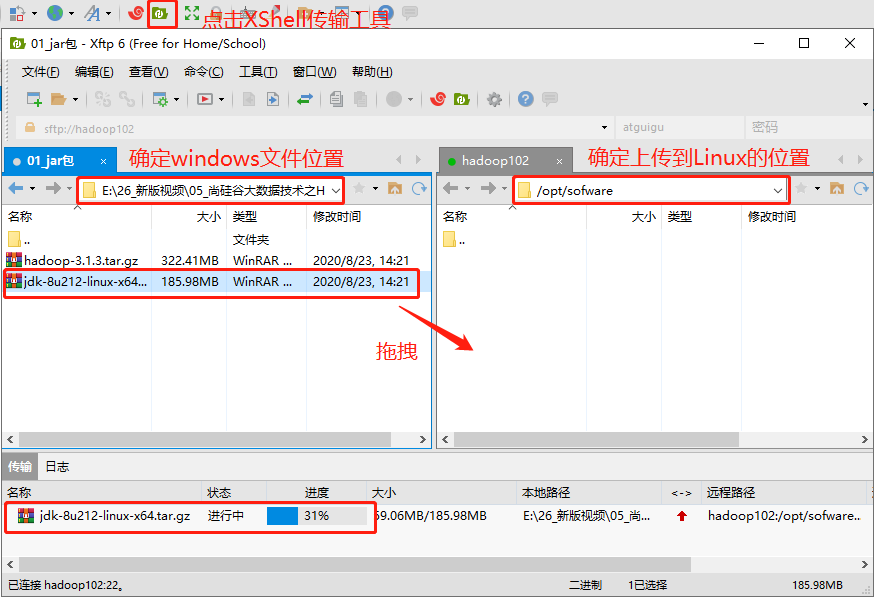
192.168.10.107 hadoop107

192.168.10.108 hadoop108

## 4.3 进一步配置

### 4.3.1在hadoop102上安装JDK

1. 用XShell传输工具将JDK导入到opt目录下面的software文件夹下面



1. 在Linux系统下解压JDK
2. 配置JDK环境变量
3. 新建/etc/profile.d/my\_env.sh文件

[anyukun@hadoop102 ~]$ sudo vim /etc/profile.d/my\_env.sh

添加如下内容

#JAVA\_HOME

export JAVA\_HOME=/opt/module/jdk1.8.0\_212

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

1. source 一下/etc/profile文件，让新的环境变量PATH生效

[anyukun@hadoop102 ~]$ source /etc/profile

### 4.3.2在hadoop102上安装Hadoop

1. 用XShell文件传输工具将hadoop-3.1.3.tar.gz导入到opt目录下面的software文件夹下面
2. 在Linux中解压hadoop-3.1.3.tar.gz到module文件夹下
3. 将Hadoop添加到环境变量
4. 修改my\_env.sh文件

[anyukun@hadoop102 hadoop-3.1.3]$ sudo vim /etc/profile.d/my\_env.sh

1. 在my\_env.sh后面添加如下内容

#HADOOP\_HOME

export HADOOP\_HOME=/opt/module/hadoop-3.1.3

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

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

1. 让修改后的文件生效

[anyukun@hadoop102 hadoop-3.1.3]$ source /etc/profile

1. 重启虚拟机

### 4.3.3完全分布式配置准备工作

1. 编写集群分发脚本xsync，该脚本能够在虚拟机之间分发并更新文件，易于操作，代码不在报告中呈现。
2. 配置ssh无密登录
3. 生成公钥和私钥

[anyukun@hadoop102 .ssh]$ pwd

/home/anyukun/.ssh

[anyukun@hadoop102 .ssh]$ ssh-keygen -t rsa

然后敲（三个回车），就会生成两个文件id\_rsa（私钥）、id\_rsa.pub（公钥）

1. 将公钥拷贝到要免密登录的目标机器上

[anyukun@hadoop102 .ssh]$ ssh-copy-id hadoop102

[anyukun@hadoop102 .ssh]$ ssh-copy-id hadoop103

[anyukun@hadoop102 .ssh]$ ssh-copy-id hadoop104

1. 在另外hadoop103和hadoop104上进行相同的操作

### 4.3.4集群配置

1. 集群部署规划

|  |  |  |  |
| --- | --- | --- | --- |
|  | hadoop102 | hadoop103 | hadoop104 |
| HDFS | NameNode  DataNode | DataNode | SecondaryNameNode  DataNode |
| YARN | NodeManager | ResourceManager  NodeManager | NodeManager |

1. 配置集群
2. 核心配置文件

配置core-site.xml

[anyukun@hadoop102 ~]$ cd $HADOOP\_HOME/etc/hadoop

[anyukun@hadoop102 hadoop]$ vim core-site.xml

文件内容如下：

<?xml version="1.0" encoding="UTF-8"?>

<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<configuration>

<!-- 指定NameNode的地址 -->

<property>

<name>fs.defaultFS</name>

<value>hdfs://hadoop102:8020</value>

</property>

<!-- 指定hadoop数据的存储目录 -->

<property>

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

<value>/opt/module/hadoop-3.1.3/data</value>

</property>

<!-- 配置HDFS网页登录使用的静态用户为anyukun -->

<property>

<name>hadoop.http.staticuser.user</name>

<value>anyukun</value>

</property>

</configuration>

1. HDFS配置文件

hdfs-site.xml文件内容如下

<?xml version="1.0" encoding="UTF-8"?>

<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<configuration>

<!-- nn web端访问地址-->

<property>

<name>dfs.namenode.http-address</name>

<value>hadoop102:9870</value>

</property>

<!-- 2nn web端访问地址-->

<property>

<name>dfs.namenode.secondary.http-address</name>

<value>hadoop104:9868</value>

</property>

</configuration>

1. YARN配置文件

yarn-site.xml文件内容如下

<?xml version="1.0" encoding="UTF-8"?>

<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<configuration>

<!-- 指定MR走shuffle -->

<property>

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

<value>mapreduce\_shuffle</value>

</property>

<!-- 指定ResourceManager的地址-->

<property>

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

<value>hadoop103</value>

</property>

<!-- 环境变量的继承 -->

<property>

<name>yarn.nodemanager.env-whitelist</name>

<value>JAVA\_HOME,HADOOP\_COMMON\_HOME,HADOOP\_HDFS\_HOME,HADOOP\_CONF\_DIR,CLASSPATH\_PREPEND\_DISTCACHE,HADOOP\_YARN\_HOME,HADOOP\_MAPRED\_HOME</value>

</property>

</configuration>

1. MapReduce配置文件

mapred-site.xml文件内容如下

<?xml version="1.0" encoding="UTF-8"?>

<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<configuration>

<!-- 指定MapReduce程序运行在Yarn上 -->

<property>

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

<value>yarn</value>

</property>

</configuration>

1. 在集群上分发配置好的Hadoop配置文件

[anyukun@hadoop102 hadoop]$ xsync /opt/module/hadoop-3.1.3/etc/hadoop/

### 4.3.5发起集群

1. 配置workers

[anyukun@hadoop102 hadoop]$ vim /opt/module/hadoop-3.1.3/etc/hadoop/workers

在该文件中增加如下内容：

hadoop102

hadoop103

hadoop104

注意文件中不能有空格和空行

保存后同步到所有点

1. 启动集群
2. 第一次启动集群需要进行格式化（只有第一次需要，后面再次启动时如果进行初始化会导致产生新的集群id）

[anyukun@hadoop102 hadoop-3.1.3]$ hdfs namenode -format

1. 启动hdfs

[anyukun@hadoop102 hadoop-3.1.3]$ sbin/start-dfs.sh

1. 在配置了ResourceManager的节点（hadoop103）启动YARN

[anyukun@hadoop103 hadoop-3.1.3]$ sbin/start-yarn.sh

## 4.4完全分布式zookeeper安装及配置

### 4.4.1 Zookeeper集群安装

* 1. 集群规划

在hadoop102、hadoop103 和 hadoop104 三个节点上都部署 Zookeeper。

* 1. 解压安装

[anyukun@hadoop102 software]$ tar -zxvf apache-zookeeper-3.5.7- bin.tar.gz -C /opt/module/

* + 1. 在 hadoop102 解压Zookeeper 安装包到/opt/module/目录下
    2. 修改apache-zookeeper-3.5.7-bin 名称为 zookeeper-3.5.7
  1. 配置服务器编号
     1. 在/opt/module/zookeeper-3.5.7/这个目录下创建 zkData
     2. 在/opt/module/zookeeper-3.5.7/zkData 目录下创建一个 myid 的文件

[anyukun@hadoop102 zkData]$ vim myid

在文件中添加与server对应的编号（上下不要有空行，左右不要有空格）

* + 1. 将配置好的zookeeper用xsync脚本分发到其他机器上
  1. 配置zoo.cfg文件
     1. 重命名/opt/module/zookeeper-3.5.7/conf 这个目录下的 zoo\_sample.cfg 为 zoo.cfg

[anyukun@hadoop102 conf]$ mv zoo\_sample.cfg zoo.cfg

* + 1. 打开zoo.cfg文件
    2. 修改数据存储路径配置

dataDir=/opt/module/zookeeper-3.5.7/zkData

#增加如下配置

#######################cluster##########################

server.2=hadoop102:2888:3888 server.3=hadoop103:2888:3888 server.4=hadoop104:2888:3888

* + 1. 将配置好的zoo.cfg文件分发给别的虚拟机
  1. 集群操作

分别启动Zookeeper

[anyukun@hadoop103 zookeeper-3.5.7]$ bin/zkServer.sh start [anyukun@hadoop104 zookeeper-3.5.7]$ bin/zkServer.sh start

[anyukun@hadoop102 zookeeper-3.5.7]$ bin/zkServer.sh start

## 4.5完全分布式HBase安装部署

1. Zookeeper正常部署

首先保证Zookeeper集群正常部署并启动

1. Hadoop正常部署

Hadoop集群正常部署并启动

1. HBase解压
   1. 解压Hbase到module目录下
   2. 配置环境变量

#HBASE\_HOME

export HBASE\_HOME=/opt/module/hbase

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

* 1. 使用source让配置的环境变量生效

1. HBase的配置文件
   1. hbase-env.sh修改内容

export HBASE\_MANAGES\_ZK=false

* 1. hbase-site.xml

<?xml version="1.0"?>

<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<configuration>

<property>

<name>hbase.zookeeper.quorum</name>

<value>hadoop102,hadoop103,hadoop104</value>

<description>The directory shared by RegionServers.

</description>

</property>

<!-- <property>-->

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

<!-- <value>/export/zookeeper</value>-->

<!-- <description> 记得修改 ZK 的配置文件 -->

<!-- ZK 的信息不能保存到临时文件夹-->

<!-- </description>-->

<!-- </property>-->

<property>

<name>hbase.rootdir</name>

<value>hdfs://hadoop102:8020/hbase</value>

<description>The directory shared by RegionServers.

</description>

</property>

<property>

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

<value>true</value>

</property>

</configuration>

* 1. regionservers

hadoop102

hadoop103

hadoop104

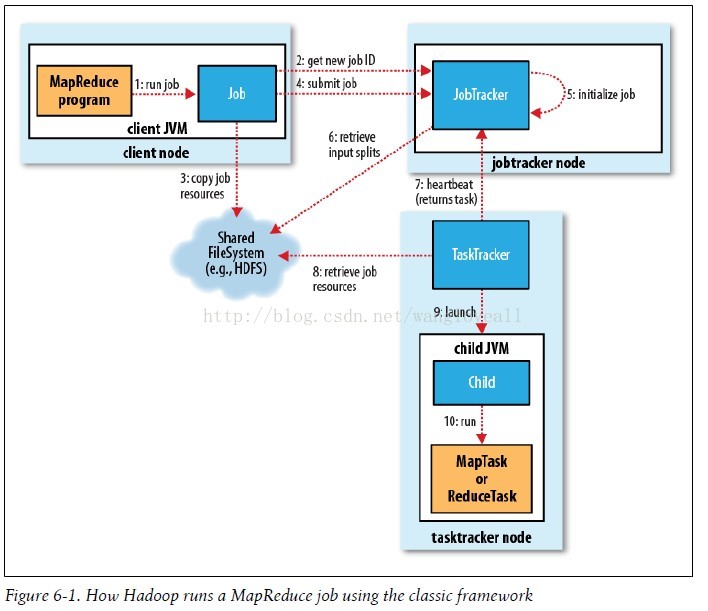
1. HBase利用xsync脚本远程发送到其它虚拟机

# 5 倒排索引及其算法

## 5.1 原理介绍

如下图所示，client提交MapReduce作业，JobTracker将作业分成多个任务Task，交由多个TaskTracker并行执行任务。对于每个任务，分为MapTask和ReduceTask，MapTask将每个文件中的句子分为多个键值对<k, v>。map后得到的键值对进行排序，将相同key的键值对存储在相同的块中。ReduceTask将键值对进行合并操作，并存储到HBase中。

对于Map任务的数量，取决于文件的大小，在环形缓存区溢写后即分配一个map任务。在分配map任务时，要考虑数据本地化（Data Local），即分配给含有该map任务的数据块中的TaskTracker中。对于Reduce任务的数量，可以在配置文件或程序中进行设置，ReduceTaskTracker根据key的HashCode对ReducerNum取余数的值分配，一个reducetask对应一个partition（分区）。



在本次实验中，我们分别继承了Mapper类和Reducer类并重写了map、reduce函数，统计了每个单词在每个文件中出现的次数，具体算法过程如下。

## 5.2 Mapper

对于环形缓存区溢出的数据，我们将每个单词及其所在的文件名用’$’符号分割拼接在一起作为key，value为1，得到<k2, v2>。例如，文件file01.txt中”i love mapreduce”在map函数处理后，得到<i$file01.txt, 1>，<love$file01.txt, 1>，<mapreduce$file01.txt, 1>。

在具体实现中，我们的InvertedIndexMapper类继承自Mapper类，K1类型为LongWritable类，V1类型为Text类，输出的K2类型为Text类，V2类型为IntWritable类。其中，LongWritable，IntWritable，Text类均为Hadoop提供的IO接口。

Mapper.java源代码如下：

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.lib.input.FileSplit;

import java.io.IOException;

import java.util.StringTokenizer;

public class InvertedIndexMapper extends Mapper<LongWritable, Text, Text, IntWritable> {

    @Override

    protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {

        String path = ((FileSplit)context.getInputSplit()).getPath().toString();    // 得到<k, v>所属的file的路径

        String fileName = path.substring(path.lastIndexOf('/') + 1);    // 得到文件名

        StringTokenizer tokenizer = new StringTokenizer(value.toString());

        tokenizer.nextToken();

        final IntWritable ONE = new IntWritable(1);

        while (tokenizer.hasMoreTokens()) {

            context.write(new Text(fileName + "$" + tokenizer.nextToken()), ONE);   // <k, v> = <文件名$单词, 1（词频）>

        }

    }

}

## 5.3 Reducer

对于<k2, v2>键值对，我们通过‘$’标识符将k2进行split操作，得到单词和其所在的文件名。count变量对v2进行累加。输出为某个单词在某个文件中的出现次数，记录单词、文件名和出现次数。

在具体实现中，我们的InvertedIndexReducer类继承自TableReducer类，该类是Reducer类的派生类，输出键值对<k3, v3>中的v3类型默认为ImmutableBytesWritable。在输出中，新建以单词为行键的put类，将文件名和单词出现的次数添加到hbase事先建立的表word的列族count中。

Reducer.java源代码如下：

import org.apache.hadoop.hbase.client.Put;

import org.apache.hadoop.hbase.io.ImmutableBytesWritable;

import org.apache.hadoop.hbase.mapreduce.TableReducer;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import java.io.IOException;

public class InvertedIndexReducer extends TableReducer<Text, IntWritable, ImmutableBytesWritable> { // 继承自TableReducer，K3无意义

    @Override

    protected void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {

        String[] fileNameAndWord = key.toString().split("\\$");     // 将文件名和单词split

        int count = 0;

        for (IntWritable value : values) {

            count += value.get();           // 该文件下的词频统计

        }

        Put put = new Put(fileNameAndWord[1].getBytes());

        put.addColumn(Main.FAMILY.getBytes(), fileNameAndWord[0].getBytes(), String.valueOf(count).getBytes()); // 添加文件名和该文件下该单词的词频

        context.write(null, put);

    }

}

# 6 实际运行

## 6.1 程序的编译、打包、运行

Main.java源代码如下：

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.hbase.HBaseConfiguration;

import org.apache.hadoop.hbase.mapreduce.TableMapReduceUtil;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import java.io.IOException;

public class Main {

    public static final String TABLE = "word";

    public static final String FAMILY = "count";

    public static void main(String[] args) throws IOException, InterruptedException, ClassNotFoundException {

        Configuration config = HBaseConfiguration.create();

        config.addResource(new Path(System.getenv("HBASE\_CONF\_DIR"), "hbase-site.xml"));    // 添加hbase-stie.xml配置

        config.addResource(new Path(System.getenv("HADOOP\_CONF\_DIR"), "core-site.xml"));    // 添加core-site.xml配置

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

        job.setJarByClass(Main.class);

        job.setMapperClass(InvertedIndexMapper.class);      // 指定自定义的Mapper类

        job.setReducerClass(InvertedIndexReducer.class);    // 指定自定义的Reducer类

        job.setMapOutputKeyClass(Text.class);               // 指定map输出的k的类型

        job.setMapOutputValueClass(IntWritable.class);      // 指定map输出的v的类型

        FileInputFormat.addInputPath(job, new Path(args[0]));   // 指定输入目录

        TableMapReduceUtil.initTableReducerJob(TABLE, InvertedIndexReducer.class, job); // 创建Reduce任务

        System.exit(job.waitForCompletion(true) ? 0 : 1);

    }

}

打包时依次选择IntelliJ IDEA的File->Project Structure->+->JAR->From modules with dependencies，Module选择InvertedIndex，Main Class选择InvertedIndex.Main，JAR files from libraries选择extract to the target JAR并设置MF文件夹得路径。在配置好hadoop、hbase环境之后，可以将相应的包导出并生成MF文件夹。之后build project再build artifact即可得到打包好的jar包。

在进行分布式环境配置之前，先在一台电脑上进行伪分布式环境下的运行测试。具体配置方法详见[Hadoop官网](https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/SingleCluster.html#Pseudo-Distributed_Operation)和[Hbase官网](https://hbase.apache.org/book.html#quickstart)。将94个文件上传至hdfs中，通过指令hdfs dfs -ls input可以查看是否成功导入。

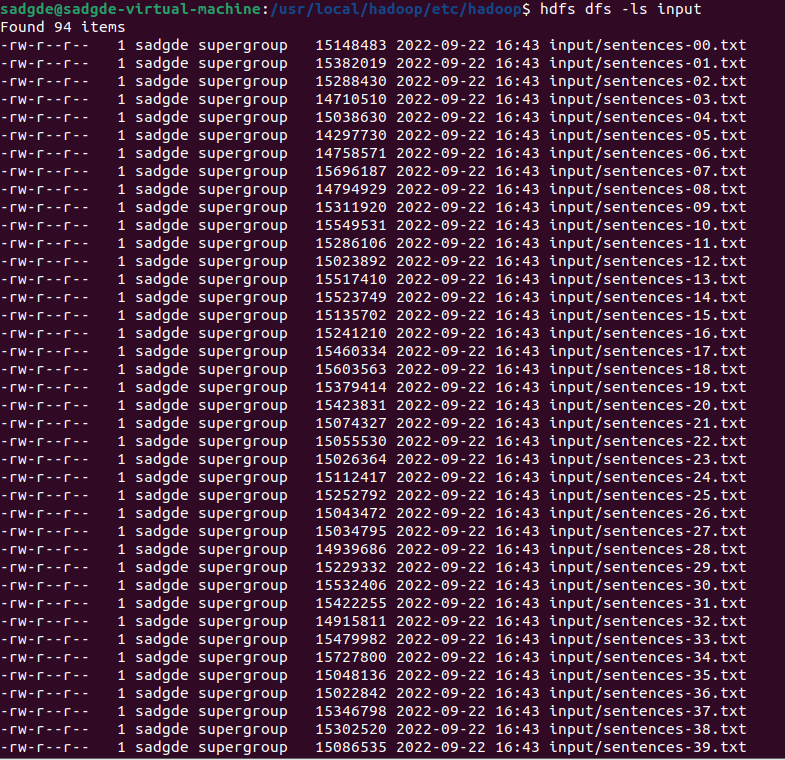


图 查看dfs文件信息

具体查看Hbase数据内容如下：

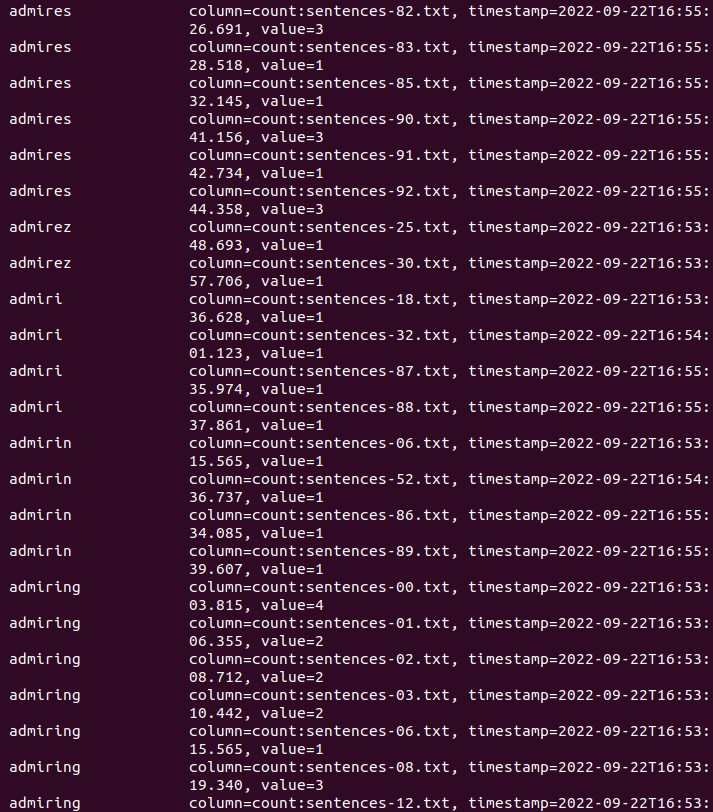


图 scan 'word'节选

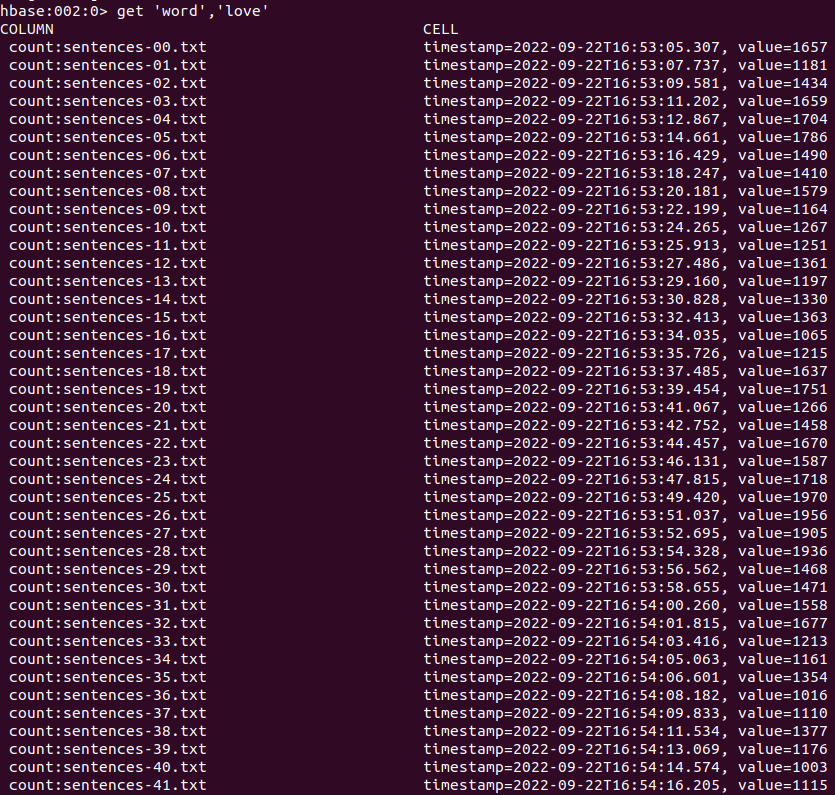
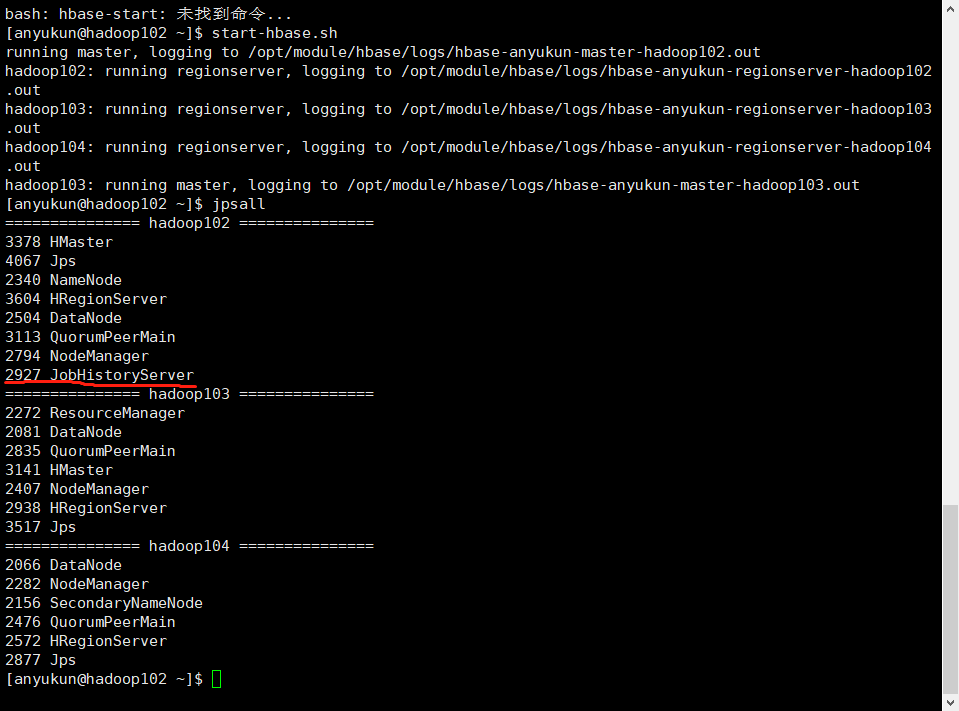
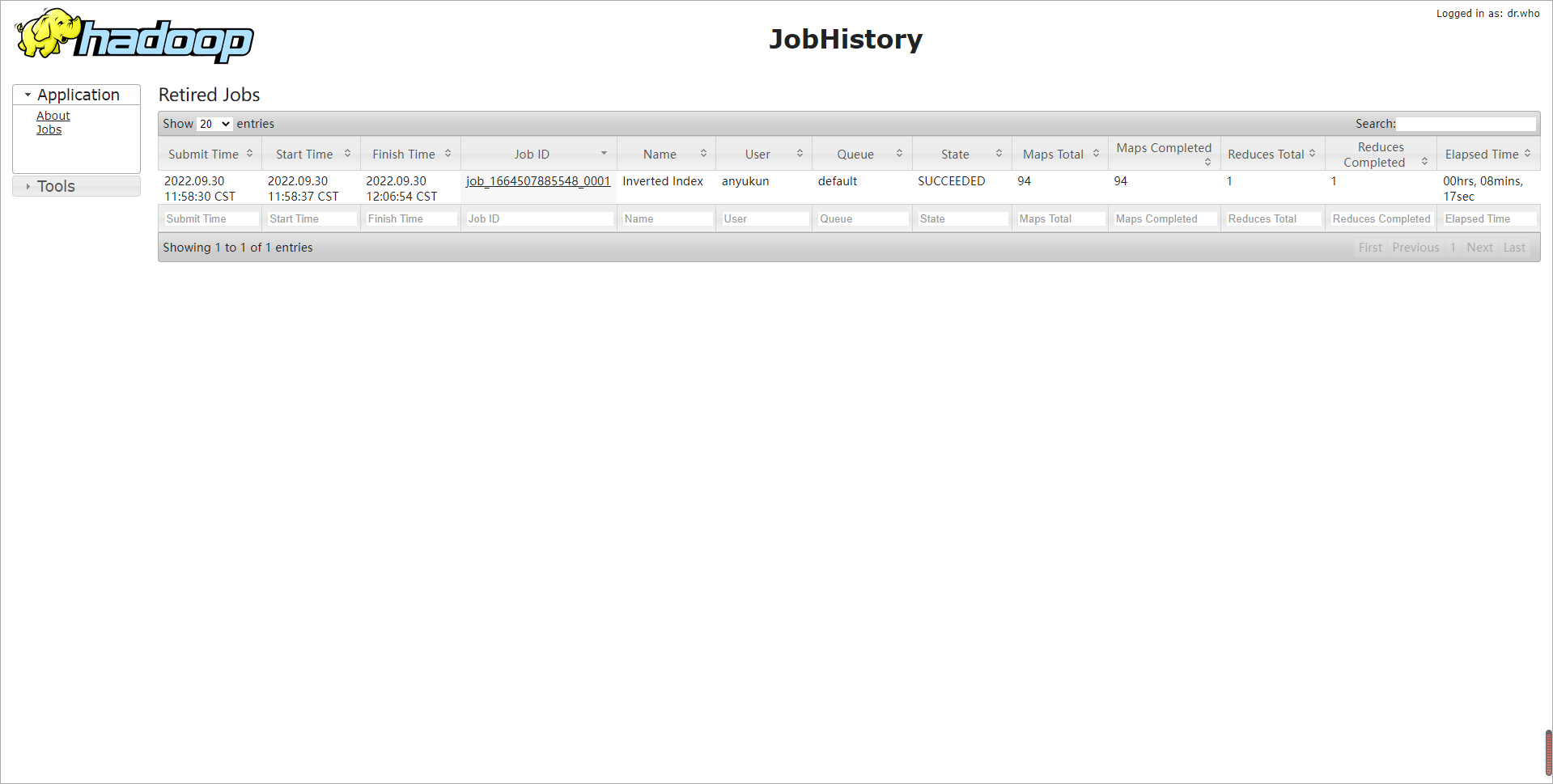


图 get 'word','love'节选

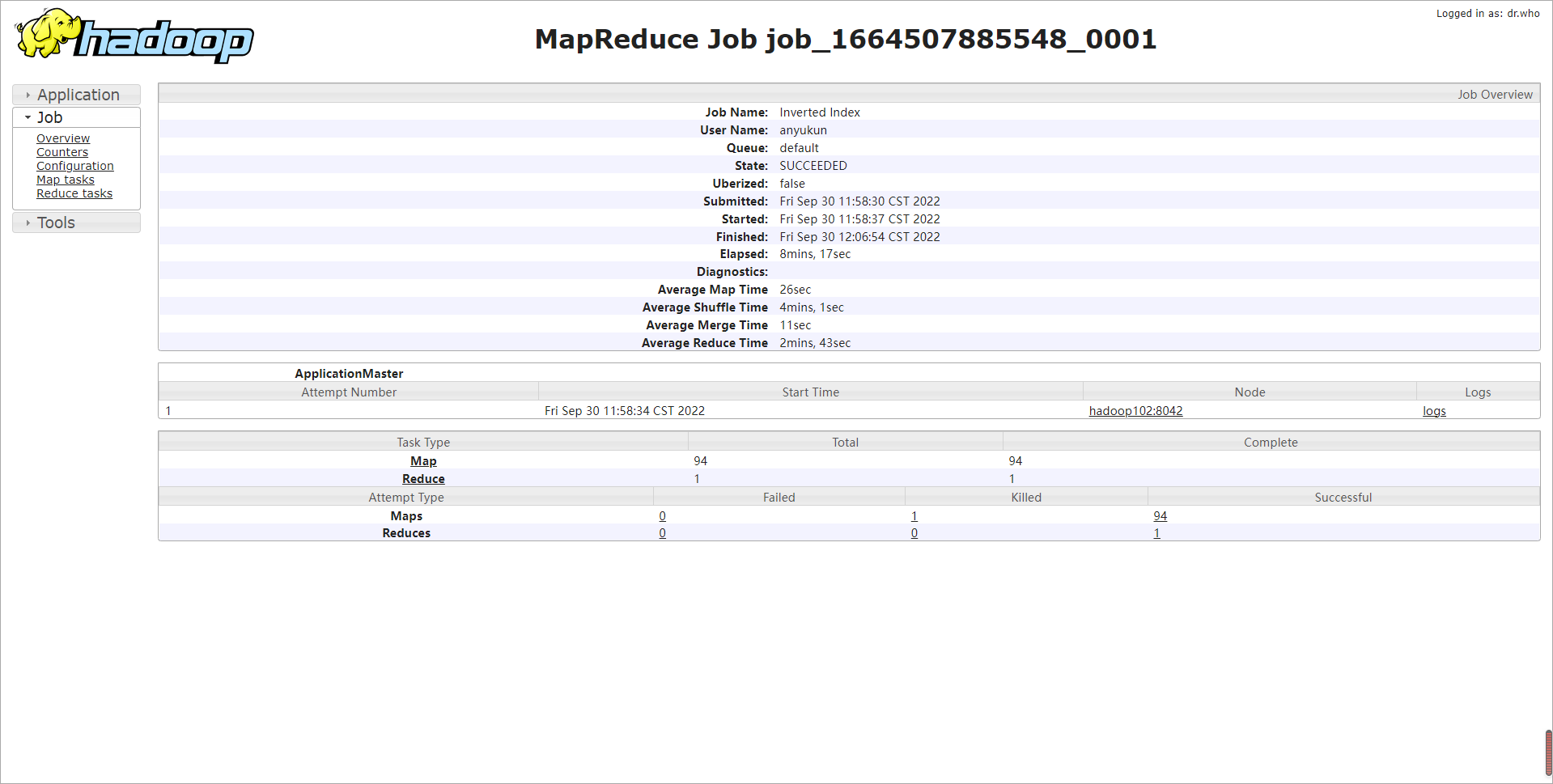
## 6.2 作业查看(JobHistory)



手动配置好历史服务器JobHistoryServer后，可以通过jps命令查看JobHistoryServer正在运行。

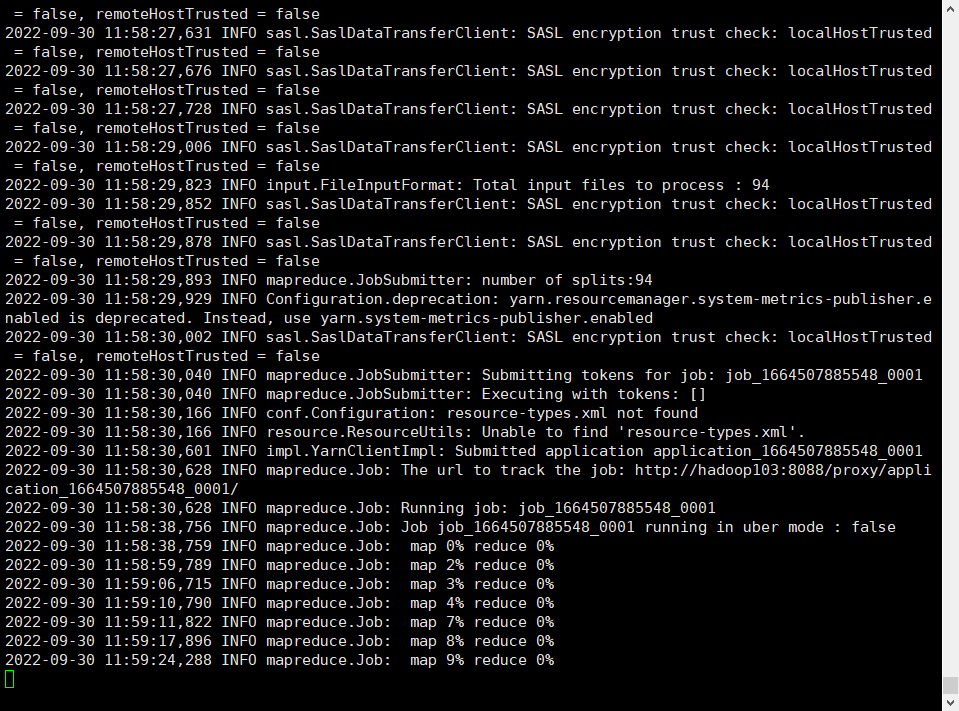


在web端可以访问<http://hadoop102:19888/>来查看历史服务器

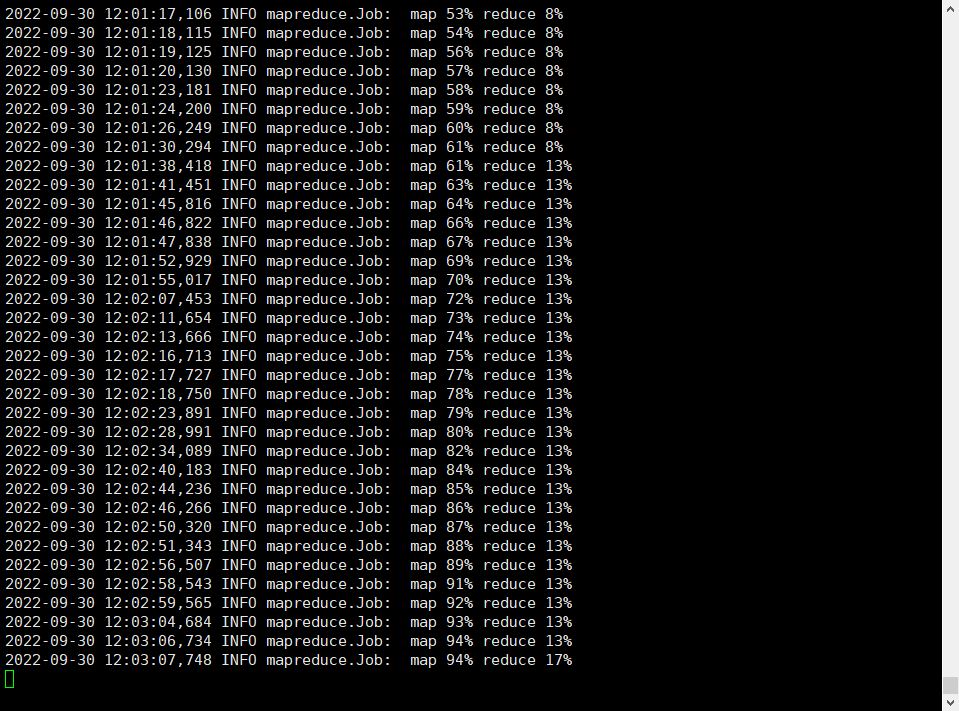


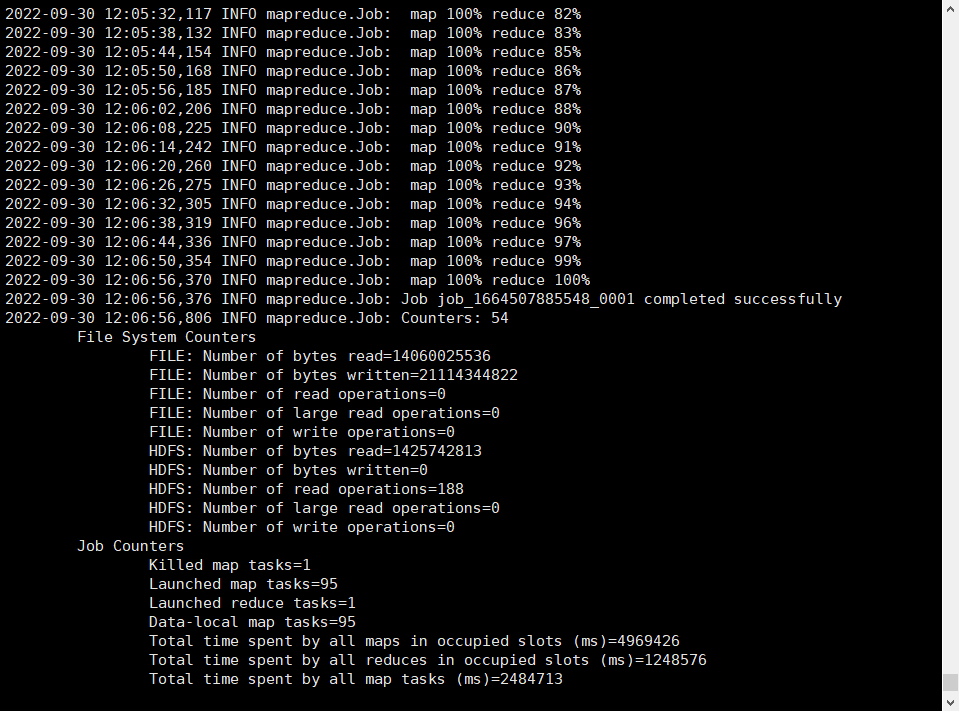
点击历史任务可以查看任务的执行详情，上图显示的是本次实现倒排索引任务的执行详情，包括执行时间、map任务数量、reduce任务数量等等。

## 6.3 最终结果展示

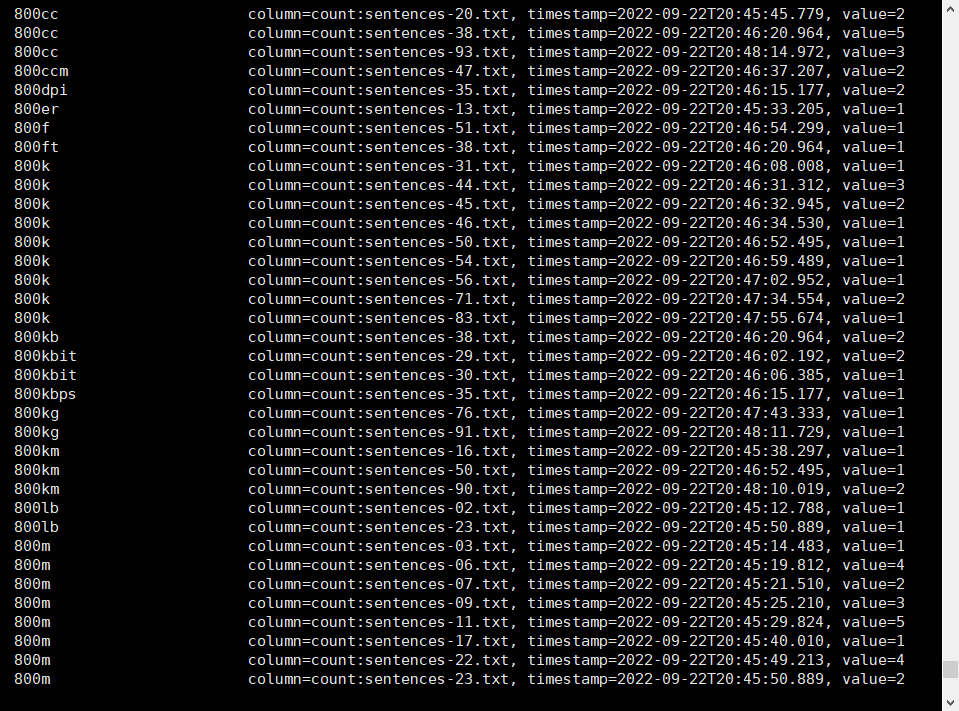


通过运行编写好的代码进行MapReduce任务





MapReduce任务完成



利用scan ‘word’查看表格中统计好的词频

# 7 故障诊断

## 7.1 配置过程中的问题

完全分布式：

按照上面写的过程进行配置时并没与遇到很大的问题

## 7.2 程序运行中的问题

在运行程序的过程中，一开始出现了错误：

Current usage: 87.0 MB of 1 GB physical memory used; 2.4 GB of 2.1 GB virtual memory used

导致所有MapReduce任务都失败了。

在网上查阅了相关资料后，在yarn-site.xml中添加下列代码：

<property>

<name>yarn.nodemanager.vmem-check-enabled</name>

<value>false</value>

</property>

在mapred-site.xml中添加下列代码：

<property>

　　<name>mapreduce.map.memory.mb</name>

　　<value>1536</value>

</property>

<property>

　　<name>mapreduce.map.java.opts</name>

　　<value>-Xmx1024M</value>

</property>

<property>

　　<name>mapreduce.reduce.memory.mb</name>

　　<value>3072</value>

</property>

<property>

　　<name>mapreduce.reduce.java.opts</name>

　　<value>-Xmx2560M</value>

</property>

对上述两个文件在所有虚拟机上进行修改并保存后，再次运行代码，问题得到了解决。

# 8 实验结果与分析

## 8.1 内容分析

从内容上，本次实验即简单完成对于多个文件中的单词词频的统计，结果截图展示详见6 程序运行。比如在HBase Shell中输入get ‘word’,‘love’可以得到列族count中的内容，第一行的sentence-00.txt代表love这个单词在sentence-00.txt这个文件出现过，value=1657表示love在这个文件里出现了1657次。这也映证了HBase的存储方式，每一个key对应一个或多个列族，并且标有timestamp（时间戳）用来记录相同key中不同版本的值。

## 8.2 性能分析

本次实验分别在伪分布环境和完全分布式环境下进行了测试，其中伪分布环境为一台机器完成namenode和datanode的任务；完全分布式环境为一台机器为namenode和datanode，另两台机器为datanode。在两种环境下，分别对其运行mapreduce计算程序的时间进行了记录，伪分布式总运行时间为4min35s，而完全分布式总运行时间为3min左右。由于数据量还不是特别大，所以二者运行时间差异并不是很明显，但是也可以看出完全分布式环境下运行效率更高一些，性能更好。所以可以得出，在分布式环境下，多台机器可以对任务进行分割完成，可以提高性能与效率。

# 9 总结

日落跌入迢迢星野，人间忽晚，山河已秋。从暑假中期的炎热到如今的秋意渐浓，历时一个月的项目终于接近尾声。

在这整个过程中，我们所取得的收获远超最初的预期，既对大数据的整体架构有了充分了解、亲自动手体会其中的奥秘，又在郭老师针砭时弊的讲述中对大学生活有了更深一层的认识。

通过本门课程和本次实验，我们第一次接触到了大数据技术和hadoop架构。通过对课程的学习，我们对于大数据技术的特征和应用场景有了初步的了解，也了解了hadoop分布式架构的优势和其原理、MapReduce工作原理等等。本次实验让我们自己动手，真正做到学以致用，从自己搭建环境开始，在虚拟机上搭建了伪分布式和完全分布式两种架构，自行编写MapReduce代码。在整个实验过程中我们对hadoop分布式架构和hbase数据库有了更深一步的理解。虽然在实验过程中也遇到了各种各样的问题，但是我们都能够自己通过查找资料进行解决，大大提升了我们的团队合作和解决问题的能力。

除此之外，本次课程设计的经历带给我们的最大感受便是“道阻且长，行则将至”。从最初时对大数据的一无所知，到最后设计出了一个一开始看起来那么遥不可及的项目，这其中不乏郭老师的精心讲授与小组成员在遇到困难时的相互支持。在这整个过程中，我们同样明白了什么叫做“积跬步以致千里”，只有努力做好眼前的事，耐下心来努力写好每一段代码、解决每一个bug，才能最终达到目标。

最后，感谢本学期中郭老师对课程内容的精心设计和在小学期期间的认真讲解，也祝愿郭老师在接下来的工作、生活中一切顺利。