**成绩**



**大数据系统开发实践项目**

**实验报告**

**题　　目： 大数据系统开发实践**

**学 院： 计算机学院**

**专业名称： 计算机科学与技术**

**小组成员：**

**任课教师： 郭贵锁**

目录

[小组成员 2](#_Toc145537857)

[搜索引擎实现的技术方案 2](#_Toc145537858)

[功能描述： 2](#_Toc145537859)

[技术选型： 2](#_Toc145537860)

[功能实现： 2](#_Toc145537861)

[1. 数据预处理： 2](#_Toc145537862)

[2. 构建倒排索引： 2](#_Toc145537863)

[3. 索引存储： 3](#_Toc145537864)

[4. 用户查询处理： 3](#_Toc145537865)

[5. 排序和结果呈现： 3](#_Toc145537866)

[6. 实例分析 3](#_Toc145537867)

[工作计划： 3](#_Toc145537868)

[组织架构： 3](#_Toc145537869)

[软件功能： 3](#_Toc145537870)

[1. 建立搜索引擎库： 3](#_Toc145537871)

[2. 在索引数据库中检索关键内容： 4](#_Toc145537872)

[软件质量保证： 4](#_Toc145537873)

[1. 功能性测试： 4](#_Toc145537874)

[2. 性能测试： 4](#_Toc145537875)

[3. 软件的非功能性保证： 4](#_Toc145537876)

[文档的倒排索引与多台电脑的真分布式实现方案 5](#_Toc145537877)

[实验要求 5](#_Toc145537878)

[数据准备 5](#_Toc145537879)

[Hadoop集群环境搭建与配置 5](#_Toc145537880)

[1. 下载软件 5](#_Toc145537881)

[2. 修改主机名 5](#_Toc145537882)

[3. 网络环境配置 5](#_Toc145537883)

[4. 修改每台电脑IP映射 6](#_Toc145537884)

[5. 免密登录 6](#_Toc145537885)

[6. 关闭各节点防火墙 6](#_Toc145537886)

[7. 测试各节点互相可以ping通 6](#_Toc145537887)

[8. 修改环境变量 6](#_Toc145537888)

[9. 修改Master配置文件 6](#_Toc145537889)

[10. 将搭建好的环境分发到各节点 10](#_Toc145537890)

[11. 启动Hadoop+zookeeper+hbase 10](#_Toc145537891)

[12. 实验中发现的坑 10](#_Toc145537892)

[13. 搭建好的环境展示（创新点1：三台电脑真分布式） 10](#_Toc145537893)

[14. 单机伪分布式展示 11](#_Toc145537894)

[算法及实现方案 12](#_Toc145537895)

[1. 拆分文件 12](#_Toc145537896)

[2. 倒排索引的原理与代码实现： 13](#_Toc145537897)

[运行结果与分析 17](#_Toc145537898)

[1. 分割源txt文件 17](#_Toc145537899)

[2. 上传文件 17](#_Toc145537900)

[3. 在HBase中建立空InvertedIndexTable 18](#_Toc145537901)

[4. 运行倒排索引代码 18](#_Toc145537902)

[5. 查看结果 21](#_Toc145537903)

[心得体会 21](#_Toc145537904)

# 小组成员

| 组员姓名 | 学号 | 教学班 |
| --- | --- | --- |
| XX1 |  |  |
| XX2 |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

# 搜索引擎实现的技术方案

## 功能描述：

实现最基础的搜索功能，可以在搜索框中输入文字，通过集群进行查询并最终返回结果。同时也可以处理非结构化的数据如图片，地图等等。

## 技术选型：

MapReduce框架可以用于大规模文本数据的分布式处理。可以编写Map和Reduce任务来执行文本清理、分词并构建倒排索引等操作。

Hadoop分布式文件系统（HDFS）用于存储大规模文本数据，以便进行处理。倒排索引的构建通常涉及大量文档，因此HDFS是一个理想的存储解决方案。

对于大规模的倒排索引，使用分布式存储系统，如HBase，来存储索引数据以支持高并发查询。HBase提供了高度可扩展的表格存储，并且适合存储大规模的索引数据。

编写一个搜索引擎需要处理互联网上大量的信息。鉴于其海量数据，我们选择采用Hadoop组件中的HDFS和MapReduce组件来解决。

ElasticSearch 是一个基于 Lucene 的搜索和分析引擎，使用它来提供搜索引擎的接口。它的主要特点是实时、分布式、易于扩展且支持多租户。与 Hadoop 集成后，ElasticSearch 可以充分利用 Hadoop 的存储和计算能力，为大规模数据提供快速的搜索和分析功能。

## 功能实现：

### 数据预处理：

对当前数据集进行清理和预处理，包括去除HTML标签、分词、词干化等。

将预处理后的数据存储在HDFS中，以便进一步处理。

### 构建倒排索引：

使用Hadoop MapReduce来构建倒排索引。倒排索引是一个单词到文档的映射，用于快速检索文档。

Map阶段：将文档分成单词，并将每个单词与文档ID关联。

Reduce阶段：将单词与文档ID列表关联起来，构建倒排索引。

### 索引存储：

存储构建好的倒排索引，可以选择将其存储在HDFS中，或者使用分布式数据库（如HBase）进行存储。

### 用户查询处理：

接收用户的搜索查询，并将查询词分词、词干化等预处理。

使用倒排索引快速定位包含查询词的文档。

### 排序和结果呈现：

根据查询词的匹配程度和其他相关性指标，对搜索结果进行排序。

将排序后的结果呈现给用户。

### 实例分析

首先，获取原始数据。对于本实验而言，我们将使用python编写爬虫脚本，通过大量爬取网页，以获得原始数据。

一旦获得原始数据，我们将每个文件分解成单词，并使用TFIDF分析工具计算每个单词在其出现的网页中的重要性。最后，生成的内容包括每个单词、它所出现的第一个文件及其重要性、它所出现的第二个文件……这些信息构成了倒排索引文件。

当用户输入搜索内容后，搜索引擎会将搜索内容分解成单词，并查找它们在倒排索引文件中的位置。然后，它们将被提取出来，并对于在同一文件中出现的单词，计算它们的重要性之和，并按降序排列，然后将这些文件按从大到小的顺序呈现给用户。

搜索引擎的工作完成。

## 工作计划：

上课（9.4-9.8）

搜索资料并学习Hadoop的使用。（9.8-9.14）

数据处理（数据集的拆分、数据清洗）。（9.15-9.18）

集群分布式实现倒排索引并编写搜索引擎技术文档。（9.19-9.23）

## 组织架构：

环境搭建与代码执行：邹子艺、韩芳暖、朱思齐

文档整理编写：尹艺璇、汪馨怡

## 软件功能：

### 建立搜索引擎库：

利用Hadoop作为底层的存储和访问工具，结合网络爬虫技术，采用深度和广度优先搜索策略以及Pagerank和OPIC等技术，实现对用户查询相关的网页的爬取，并将其中的页面内容进行提取和相关信息的存储。

一旦获取了保存在HDFS中的网页的文本信息，可以利用MapReduce算法来计算，从而构建一个倒排索引。这倒排索引可以存储在HBase中，形成一个快速、可扩展的索引数据库。

这个系统的工作流程如下：

利用MapReduce算法，将数据集进行处理，构建倒排索引。在倒排索引中，每个单词将与它出现在哪些网页中以及在这些网页中的位置等信息相关联。

构建的倒排索引可以存储在HBase中，作为一个高效的索引数据库。这样，用户查询时可以快速地定位相关的网页，并提供相关信息。

通过这个系统，用户可以进行高效的信息检索，而Hadoop、HDFS、MapReduce、HBase、ElasticSearch等技术的结合为构建大规模、高性能的搜索引擎提供了可行的基础。

### 在索引数据库中检索关键内容：

用户输入搜索关键词并点击搜索按钮后，系统的检索程序启动。接着，通过倒排索引，从网页索引数据库中检索包含用户搜索内容的所有相关网页。最后，系统对搜索结果进行排序处理，并将处理完毕的文档呈现给用户在客户端上浏览。

## 软件质量保证：

### 功能性测试：

确保软件按照规格说明书中的要求执行其基本功能。

验证每个功能是否按预期工作，包括输入验证、输出正确性等。

### 性能测试：

测试软件在正常和极端负载下的性能，包括响应时间、吞吐量和资源利用率等。

评估软件的性能是否满足预期的需求。

### 软件的非功能性保证：

搜索引擎需要满足快速和便捷的要求，以及对用户的打开和检索请求作出及时响应。这需要高效的检索算法和快速的读取速度。在测试过程中，我们发现搜索引擎的检索性能基本满足要求。此外，通过对Hadoop集群性能的调优和定期更新索引，我们可以进一步有效提高搜索引擎的响应速度。

此外，搜索引擎的Web应用程序也需要满足简洁和美观的需求。在确保响应速度的同时，我们将着重改善Web页面的设计，以提高其美观度，从而提升用户体验。

# 文档的倒排索引与多台电脑的真分布式实现方案

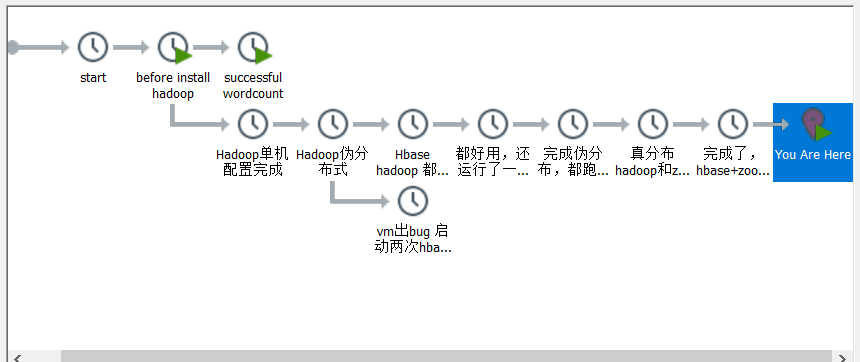
## 实验要求

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。

## 数据准备

将压缩文件sentences.txt.zip以一万为单位拆分数据集，具体实现见下文。

## Hadoop集群环境搭建与配置



### 下载软件

VMware：本次实验使用的是17.0.0 build-20800274

CentOS Linux release 7.9.2009 (Core)

Java：本次实验的Java版本是oracle jdk 1.8\_341

Hadoop：本次实验使用的是Hadoop-3.3.5

ZooKeeper：ZooKeeper-3.8.2

HBase：HBase-2.5.4（HBase版本与Hadoop版本有冲突，但是通过修改配置文件解决）

### 修改主机名

修改为Hadoop01，Hadoop02，Hadoop03.

| 主机 | 角色 |
| --- | --- |
| Hadoop01 | 主节点 |
| Hadoop02 | 从节点 |
| Hadoop03 | 从节点 |

### 网络环境配置

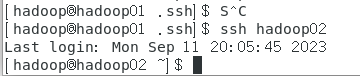
本实验需要将网络连接方式改为桥接模式以实现多个节点互连。

### 修改每台电脑IP映射

在host中修改IP映射。

### 免密登录

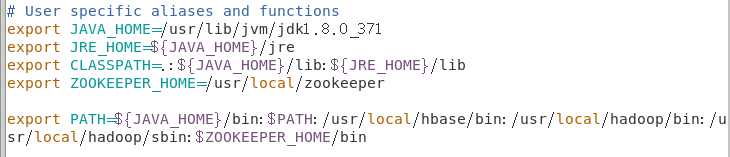
下图为主节点成功免密登录各从节点



### 关闭各节点防火墙

### 测试各节点互相可以ping通

### 修改环境变量



### 修改Master配置文件

1. zookeeper:



1. HBase：

hbase-env.sh：

export HBASE\_DISABLE\_HADOOP\_CLASSPATH\_LOOKUP="true"

# The java implementation to use. Java 1.8+ required.

export JAVA\_HOME=/usr/lib/jvm/jdk1.8.0\_371

export HBASE\_MANAGES\_ZK=false

export HBASE\_CLASSPATH=/usr/local/hadoop/etc/hadoop

hbase-site.xml：

<configuration>

<property>

<name>hbase.rootdir</name>

<value>hdfs://hadoop01:9000/hbase</value>

</property>

<property>

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

<value>true</value>

</property>

<property>

<!-- 指定 zookeeper 的地址-->

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

<value>hadoop01:2181,hadoop02:2181,hadoop03:2181</value>

</property>

</configuration>

1. Hadoop：

core-site.xml:

<configuration>

<property>

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

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

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

</property>

<property>

<name>fs.defaultFS</name>

<value>hdfs://hadoop01:9000</value>

</property>

</configuration>

hdfs-site.xml:

<configuration>

<property>

<name>dfs.replication</name>

<value>2</value>

</property>

<property>

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

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

</property>

<property>

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

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

</property>

<property>

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

<value>hadoop01:50090</value>

</property>

</configuration>

yarn-site.xml：

<configuration>

<property>

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

<value>hadoop01</value>

</property>

<property>

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

<value>mapreduce\_shuffle</value>

</property>

</configuration>

mapred-site.xml：

<configuration>

<property>

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

<value>yarn</value>

</property>

<property>

<name>mapreduce.jobhistory.address</name>

<value>hadoop01:10020</value>

</property>

<property>

<name>mapreduce.jobhistory.webapp.address</name>

<value>hadoop01:19888</value>

</property>

<property>

<name>yarn.app.mapreduce.am.env</name>

<value>HADOOP\_MAPRED\_HOME=/usr/local/hadoop</value>

</property>

<property>

<name>mapreduce.map.env</name>

<value>HADOOP\_MAPRED\_HOME=/usr/local/hadoop</value>

</property>

<property>

<name>mapreduce.reduce.env</name>

<value>HADOOP\_MAPRED\_HOME=/usr/local/hadoop</value>

</property>

</configuration>

### 将搭建好的环境分发到各节点

使用SCP将搭建好的环境分发到各节点。

### 启动Hadoop+zookeeper+hbase

先启动zookeeper，在各个节点上运行以下命令：

zkServer.sh start

接着启动hadoop，只需在master节点上运行一次即可。

# 启动dfs服务

start-dfs.sh

# 启动yarn服务

start-yarn.sh

最后启动hbase，注意一定要等hadoop启动成功后30s以上再启动hbase，否则会造成java1.8线程池异常。

start-hbase.sh

### 实验中发现的坑

1. vmware虚拟机在开两个相同名字的虚拟机的同时，删除旧的快照，会使得vm的文件系统崩溃，然后把正在运行的虚拟机强制闪退。

2023-09-07T14:12:02,893 ERROR [main] regionserver.HRegionServer: Failed construction RegionServer

java.lang.UnsupportedOperationException: Constructor threw an exception for org.apache.hadoop.hbase.ipc.NettyRpcServer

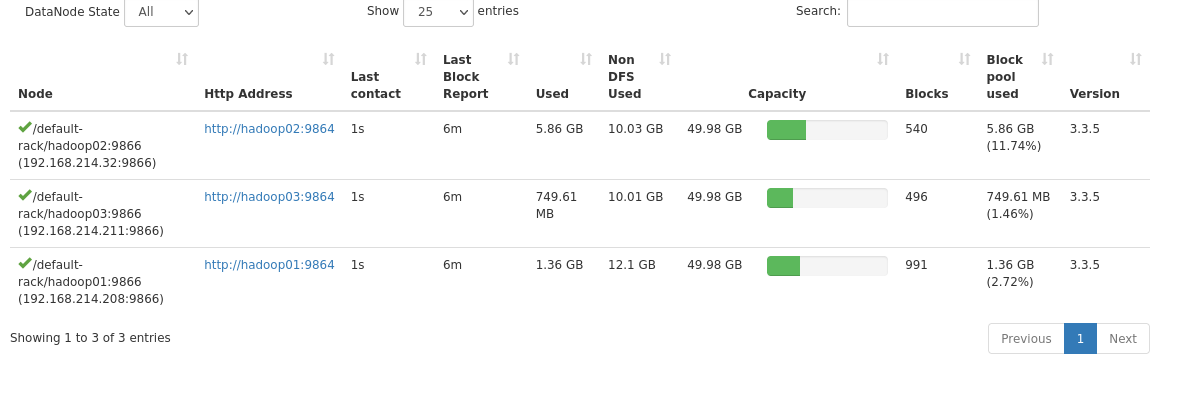
如上所示：然后虚拟机内java8的线程池就会崩溃。这样一来，当启动hbase的shell时，就会导致Master节点崩溃，进而导致zookeeper崩溃。最终在shell中就会出现无法连接的状态。这时只要重新再启动一次hbase就可以解决这个问题。至少在2.5.4这个问题是存在的。

2. 解压hbase时，如果被解压文件正在进行文件复制，就会将不完整的文件解压到对应位置，而且会报错，如果不彻底清除临时文件就再次进行解压，就会造成部分文件解压为空文件但是不报错。如果在此基础上配置hbase，会造成hbase运行异常。

### 搭建好的环境展示（创新点1：三台电脑真分布式）

以上是本次实验过程中所有的配置流程，我们使用了三台电脑实现了真正的分布式，与以往的伪分布和在一台电脑上开启多台虚拟机不同。本来我们预期和一台电脑多开基本没什么区别，但是实践中遇到了多个问题。1.校园网不给虚拟机的mac分配地址，所以需要手机开热点才可以正常ping通。2.vm17.002的软件的桥接模式似乎有问题，经过多次尝试最终通过安装最新版的vmware解决了问题。相比于单机运行，三台电脑总共能提供24核和24g的内存进行运行，速度相当的快。

Hadoop集群配置完成图：

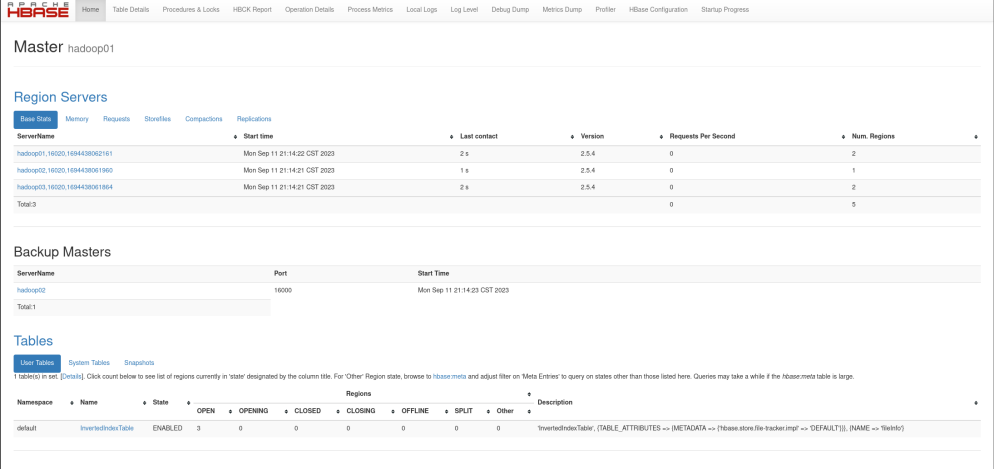


多台电脑的真分布实现：

图片的内容是三台电脑分别打开hadoop后台，hbase后台，yarn后台的展示，说明我们确实将其部署到了三台不同的电脑上。



HBase后台界面：



### 单机伪分布式展示

我们同时也搭建了单机伪分布式的环境，便于开发调试。从jps可以看出hadoop+zookeeper+hbase都以伪分布式的方式在运行。在实际的开发环境中，应该也是先在伪分布式上运行成功，然后再到真分布的生产环境。

图形用户界面, 文本, 应用程序

描述已自动生成

## 算法及实现方案

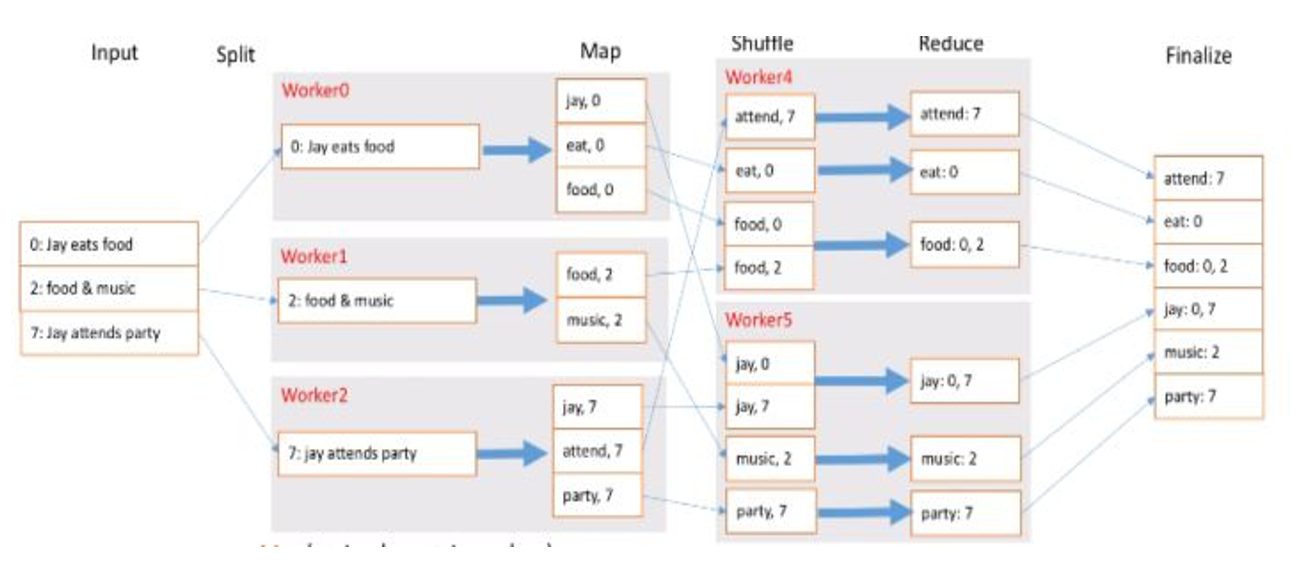
### 拆分文件

从实验要求可以得到，每个文件应该包含10000句话。为此我们使用java进行处理。以下是该部分的代码，尽管只使用一次，我们还是把错误处理加上去了。因为代码中没有使用string进行中转，所以代码效率比较高，很快就处理完了，共940个文件：

import java.io.BufferedReader;  
import java.io.BufferedWriter;  
import java.io.FileReader;  
import java.io.FileWriter;  
import java.io.IOException;  
  
public class FileSplitter {  
 public static void main(String[] args) {  
 // 源文件路径  
 String sourceFilePath = "sentences.txt";  
 // 目标文件路径前缀  
 String targetFilePrefix = "split\_";  
 // 每个文件的最大行数  
 int maxLinesPerFile = 10000;  
  
 try (BufferedReader reader = new BufferedReader(new FileReader(sourceFilePath))) {  
 String line;  
 int lineCount = 0;  
 int fileCount = 0;  
 BufferedWriter writer = new BufferedWriter(new FileWriter(targetFilePrefix + fileCount + ".txt"));  
  
 while ((line = reader.readLine()) != null) {  
 writer.write(line);  
 writer.newLine();  
 lineCount++;  
  
 if (lineCount >= maxLinesPerFile) {  
 writer.close();  
 lineCount = 0;  
 fileCount++;  
 writer = new BufferedWriter(new FileWriter(targetFilePrefix + fileCount + ".txt"));  
 }  
 }  
  
 writer.close();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
}

### 倒排索引的原理与代码实现：

mapreduce过程示意图：



#### map阶段

当输入为”0 Jay eats food”时，map函数以空格为界，将该句中所有的单词分离，那么第一个单词就是该行文档标题的编号，所以从第二个单词开始，单词为key，编号为value进行输出。通过调试跟踪运行，我发现map是按照行来运行的。不管是不是同一个文件。

#### reduce阶段

reduce阶段将key相同的键值对进行累加合并，key不变，value值以分号为分割符放在一起，最终写入Hbase。

#### 倒排索引的代码实现：

以下的代码直接把结果写进了hbase中

**import** java.io.IOException;

**import** java.util.StringTokenizer;

**import** org.apache.hadoop.conf.Configuration;

**import** org.apache.hadoop.fs.Path;

**import** org.apache.hadoop.hbase.HBaseConfiguration;

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

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

**import** org.apache.hadoop.hbase.mapreduce.TableOutputFormat;

**import** org.apache.hadoop.hbase.util.Bytes;

**import** org.apache.hadoop.mapreduce.Job;

**import** org.apache.hadoop.mapreduce.Mapper;

**import** org.apache.hadoop.mapreduce.Reducer;

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

**import** org.apache.hadoop.util.GenericOptionsParser;

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

**import** org.apache.hadoop.io.Text;

**public** **class** InvertedIndex {

**public** **static** **class** Map **extends** Mapper<Object, Text, Text, Text> {

**private** Text keyInfo = **new** Text();

**private** Text valueInfo = **new** Text();

**private** FileSplit split;

**public** **void** map(Object key, Text value, Context context) **throws** IOException, InterruptedException {

split = (FileSplit) context.getInputSplit();

StringTokenizer itr = **new** StringTokenizer(value.toString());

**while** (itr.hasMoreTokens()) {//按行map，然后放文件名，如果相对路径名就太长了

String fileName = split.getPath().getName();

keyInfo.set(itr.nextToken() + ":" + fileName);

valueInfo.set("1");

context.write(keyInfo, valueInfo);

}

}

}

**public** **static** **class** Combine **extends** Reducer<Text, Text, Text, Text> {

**private** Text info = **new** Text();

**public** **void** reduce(Text key, Iterable<Text> values, Context context) **throws** IOException, InterruptedException {//这个是combine步骤

**int** sum = 0;

**for** (Text value : values) {

sum += Integer.*parseInt*(value.toString());

}

**int** splitIndex = key.toString().indexOf(":");

info.set(key.toString().substring(splitIndex + 1) + ":" + sum);

key.set(key.toString().substring(0, splitIndex));

context.write(key, info);

}

}

**public** **static** **class** Reduce **extends** Reducer<Text, Text, ImmutableBytesWritable, Put> {

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

//开始写入数据库中。

String fileList = "";

**for** (Text value : values) {

fileList += value.toString() + ";";

}

Put put = **new** Put(Bytes.*toBytes*(key.toString()));

put.addColumn(Bytes.*toBytes*("fileInfo"), Bytes.*toBytes*("fileList"), Bytes.*toBytes*(fileList));

context.write(**new** ImmutableBytesWritable(key.getBytes()), put);

}

}

**public** **static** **void** main(String[] args) **throws** Exception {

Configuration conf = HBaseConfiguration.*create*();

String[] ioArgs = **new** String[]{"sentence\_split", "output"};

String[] otherArgs = **new** GenericOptionsParser(conf, ioArgs).getRemainingArgs();

**if** (otherArgs.length != 2) {

System.***err***.println("Usage: Inverted Index <in> <out>");

System.*exit*(2);

}

//现在就是把job建起来，然后启动

Job job = **new** ~~Job~~(conf, "Inverted Index");

job.setJarByClass(InvertedIndex.**class**);

job.setMapperClass(Map.**class**);

job.setCombinerClass(Combine.**class**);

job.setReducerClass(Reduce.**class**);

job.setMapOutputKeyClass(Text.**class**);

job.setMapOutputValueClass(Text.**class**);

job.setOutputKeyClass(ImmutableBytesWritable.**class**);

job.setOutputValueClass(Put.**class**);

//向hbase中写入InvertedIndexTable是已经建立好的表

job.setOutputFormatClass(TableOutputFormat.**class**);

job.getConfiguration().set(TableOutputFormat.***OUTPUT\_TABLE***, "InvertedIndexTable");

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

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

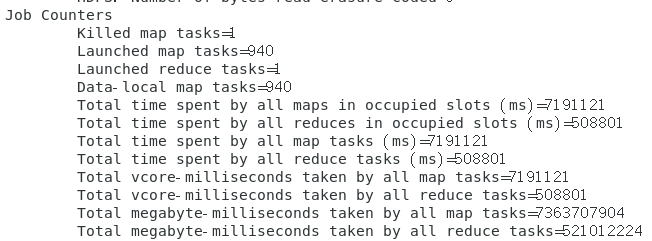
}

}

mapreduce运行过程中：

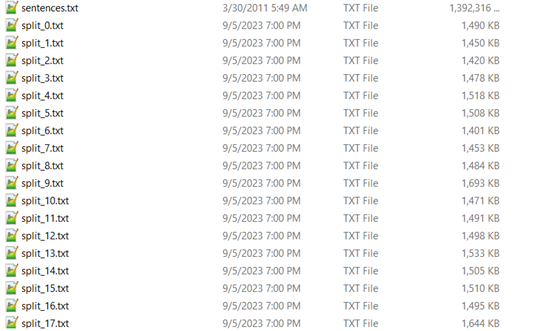


程序运行结束，可以看出速度相当的快。

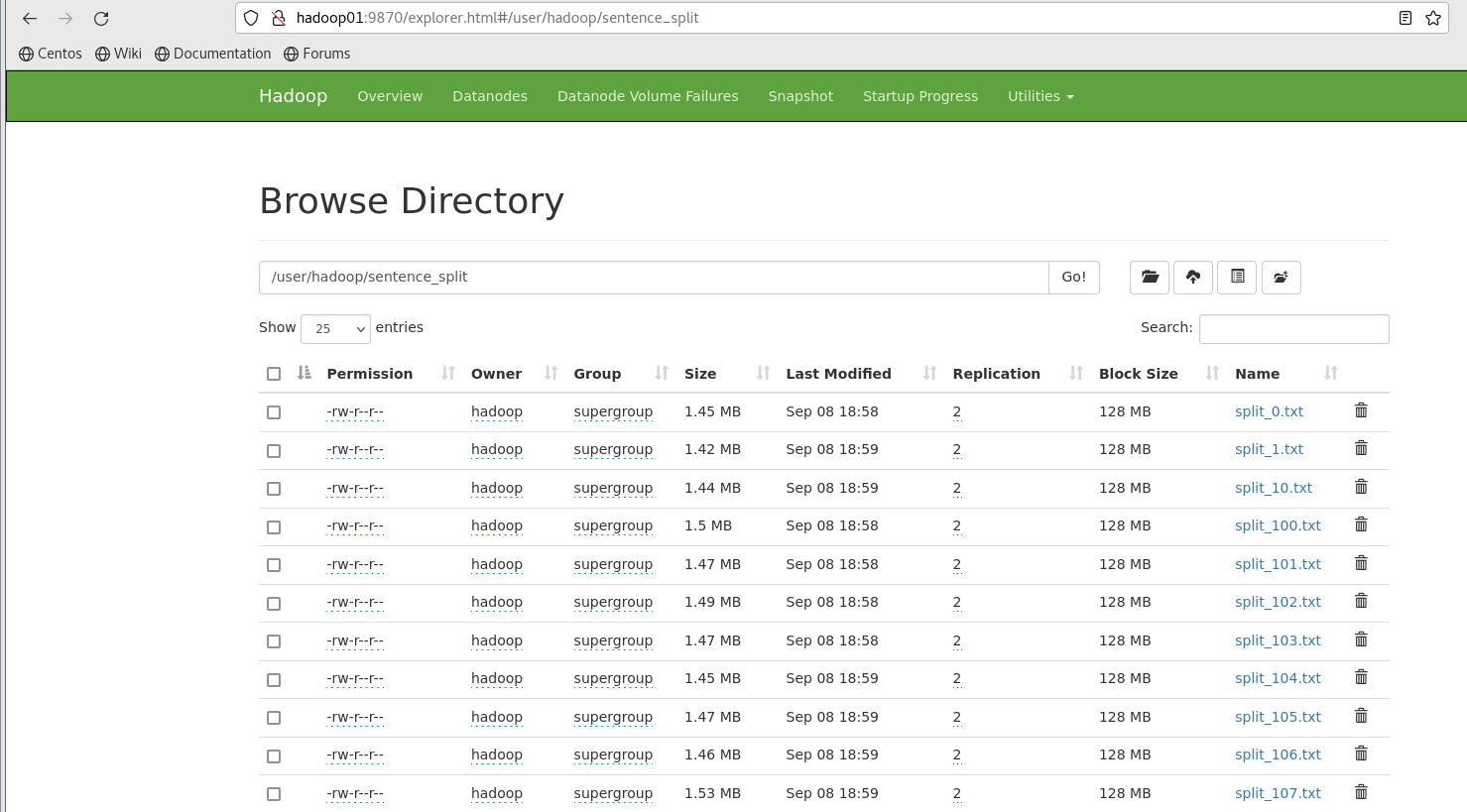


## 运行结果与分析

### 分割源txt文件

* 

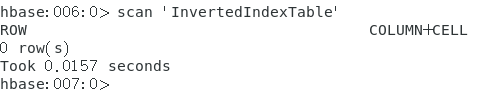
### 上传文件

* 

### 在HBase中建立空InvertedIndexTable

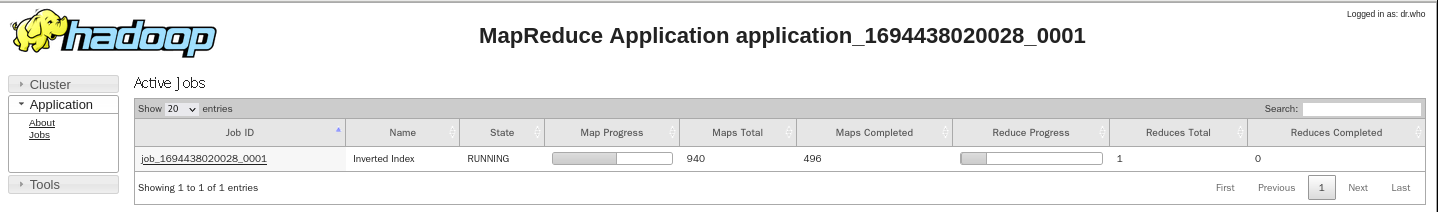
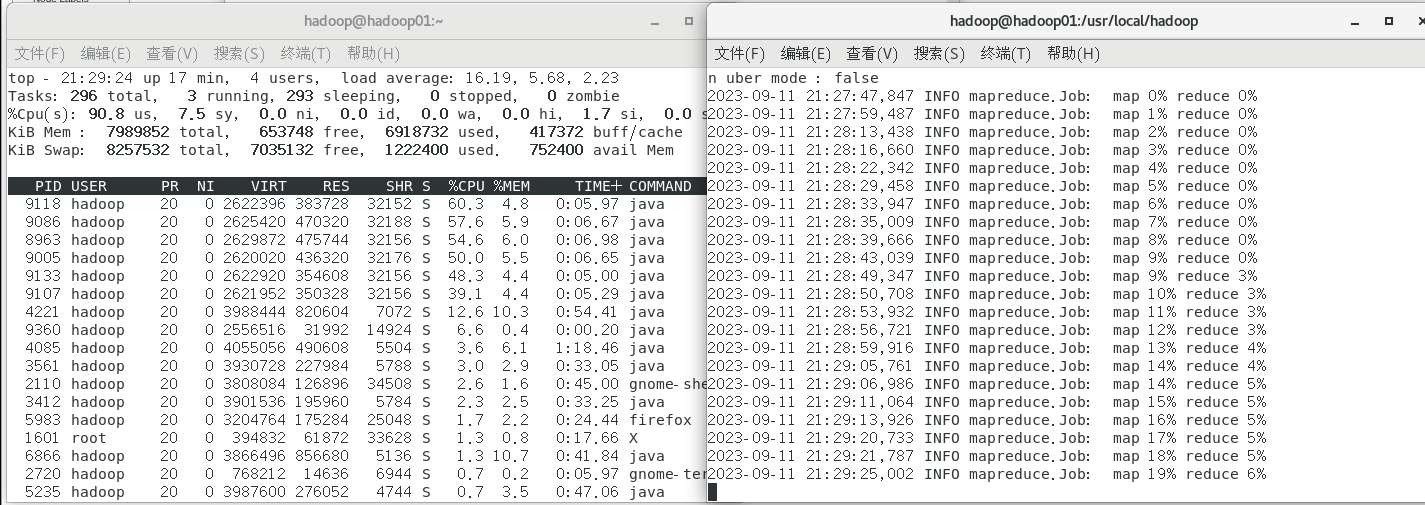
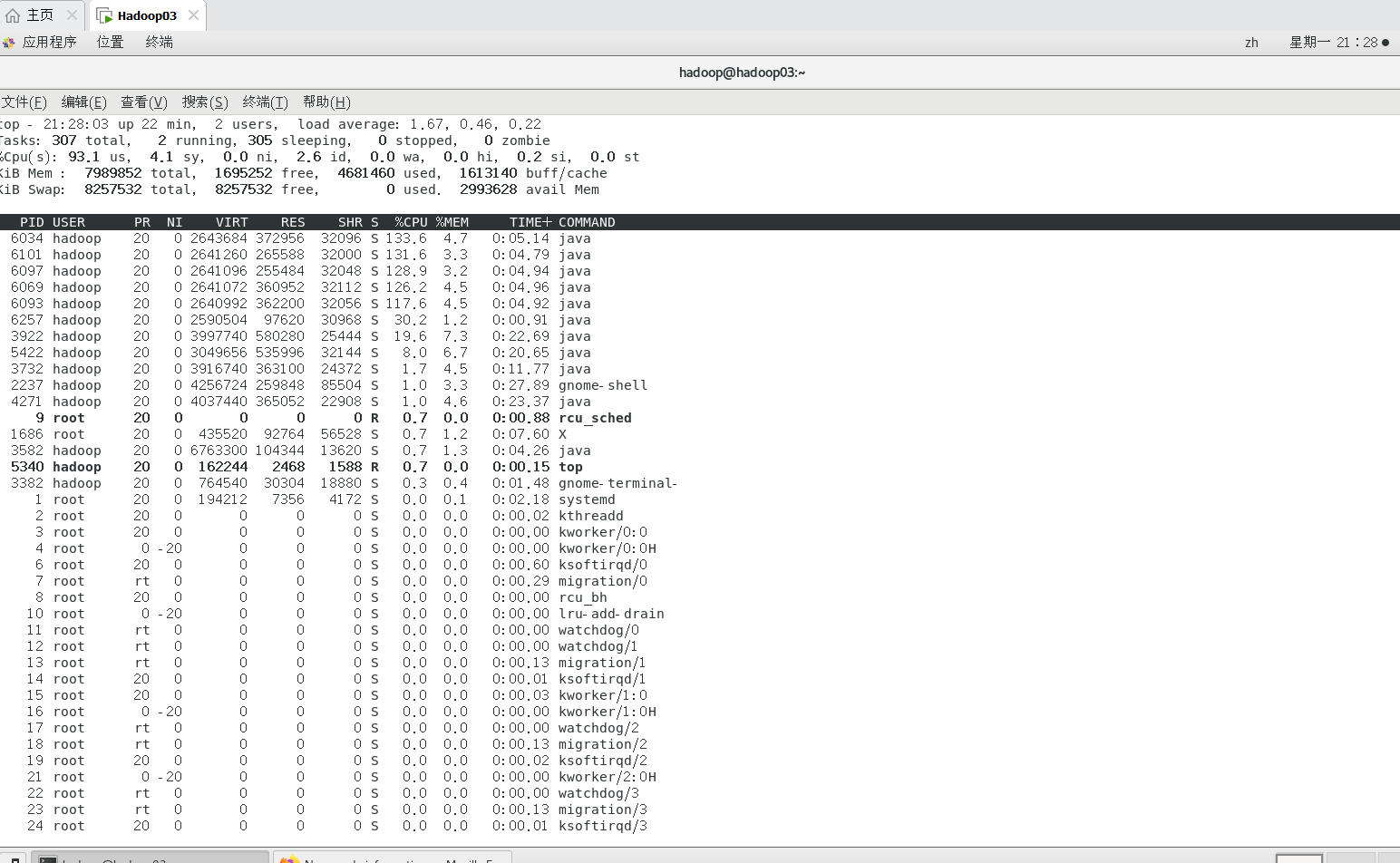
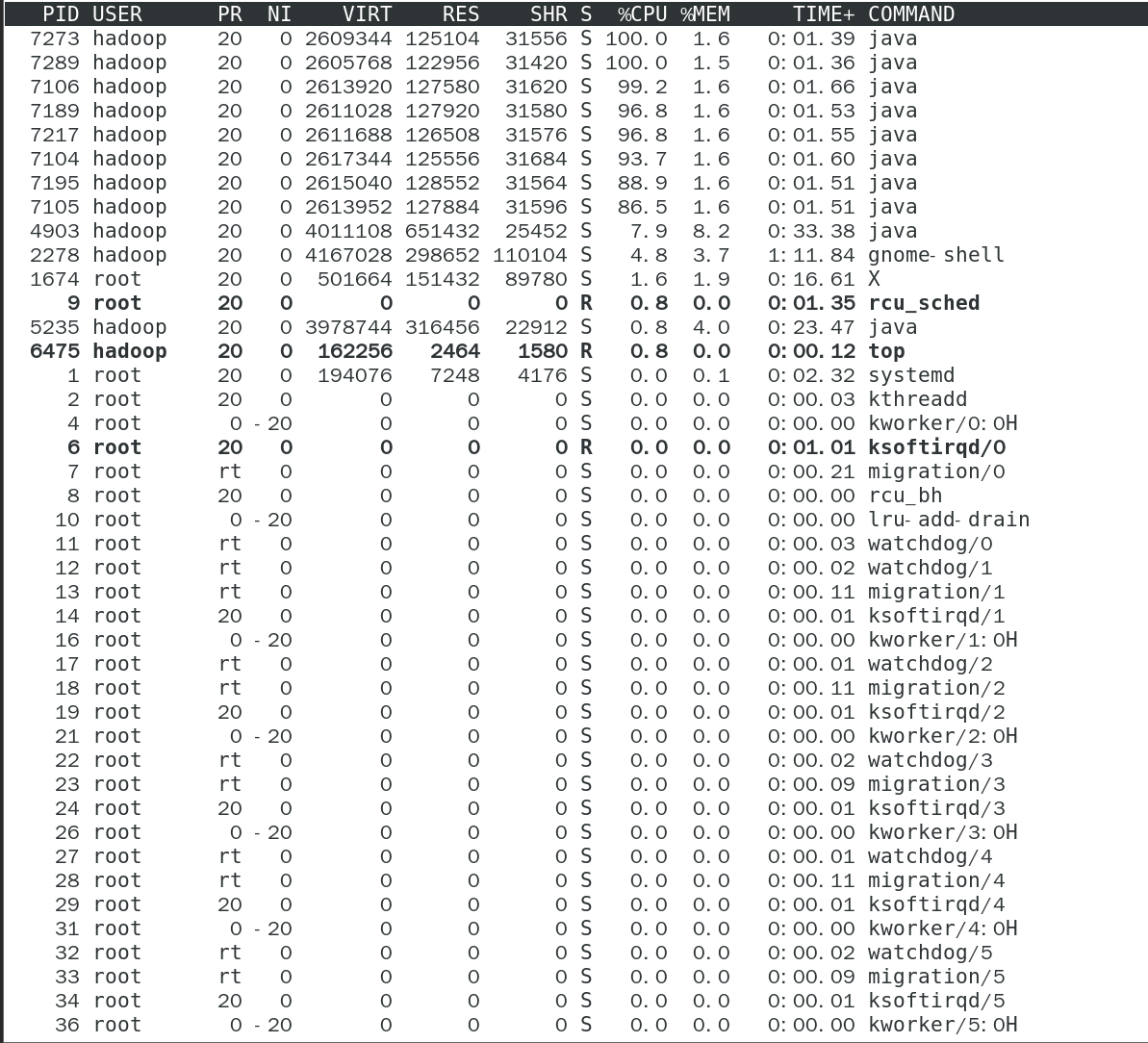
create 'InvertedIndexTable', 'fileInfo'

查看建立好的表格：



### 运行倒排索引代码

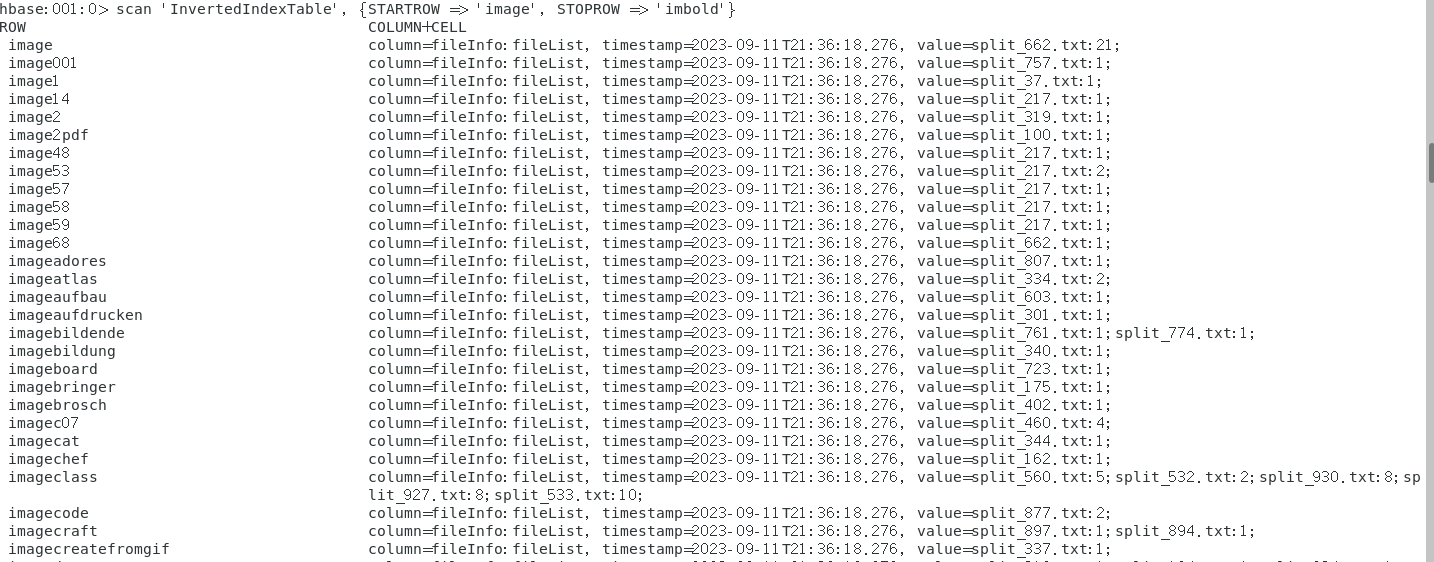
本来在ide中也能运行，但是为了贴近生产环境，我们将其打包成jar文件运行。

* hadoop jar ./myapp/full\_inverted\_index\_sentence\_split.jar 
* 节点Hadoop01运行倒排索引的过程进度：
* 节点Hadoop02运行倒排索引的过程：
* 
* 节点Hadoop03运行倒排索引的过程：
* 
* 可以看出三台电脑都在运行MapReduce

### 查看结果

由于本次实验是按照文件划分的，所以900万多的句子每一行都会有序号，导致hbase中很大一部分是 “行号-文件名”形式。为了体现出倒排索引，运行以下的代码，将单词image

到单词'imbold'的倒排索引展示出来。

scan 'InvertedIndexTable', {STARTROW => 'image', STOPROW => 'imbold'}

可以看出确实实现了倒排索引。

# 心得体会

在本次的大数据系统开发课程中，我学到了很多知识，尤其是关于如何将多个主机的性能充分发挥。hadoop是一个比较成熟的生态圈了，我在网上看到很多资料基本都是基于hadoop来实现的。只是可惜MapReduce似乎不是很适合分布式编译，这样我手里空下来的8台云服务器还是不能发挥很大的作用。

本次实验过程中我遇到了很多问题，主要是之前没有在linux上搭建过这么复杂的配置，所以也算是走过了一些弯路，比方说我开始为了图省事打算在windows上测试搭建hadoop，结果发现hadoop对windows的支持实在是太差了，就算搭建成功接下来的实验也没法做。于是我开始在linux上搭建。我在网上找到的教程开始是错的，后来发现厦门大学的林子雨教授的教程很好用，同时这次经历极大的锻炼了我看log的能力，通过看log我发现出现IllegalArgumentException原因是：HBase的jar包和Hadoop的jar包有冲突，导致服务没有起来，最后通过调整配置文件解决了。然后是使用vmware虚拟机，没想到虚拟机居然桥接模式有问题，经过一系列重置网络，重启，重装等操作，发现解决方案是安装最新版本的vmware。还有一些其他的问题我记录在安装过程中了。

hbase安装也比较麻烦，个人感觉比hadoop还要麻烦，但是因为有了hadoop的经验，所以安装快很多。有趣的是hbase运行的速度比我想象的快多了，本来看着hdfs那运行速度，我还以为写入hbase要写好久，没想到写入hbase和写入txt相比没有差多少。看来工业界的代码就是快。安装完成后我就在想，hbase，zookeeper，hadoop毕竟也是经过无数人部署了，对于新手来说还是很麻烦。如果没有docker这种容器方便进行批量管理和环境搭建，恐怕很多开源项目连发展起来的机会都没有。

最后成功在三台笔记本运行成功时我是相当的激动，毕竟干了这么久终于有结果了，而且我之前就有一个把我手里的云服务器物尽其用的想法，尽管hadoop+zookeeper+hbase只是小小的一步，但是也为我之后搭建分布式编译系统提供了宝贵的经验。这次上课学到的知识和通过实验提升的能力我想对以后的学习工作也有帮助。