大数据处理综合实验

实验二

倒排索引

小组编号：01

组长姓名：张玲 201220052

2023年5月1日

Part 1:文档倒排索引算法

1.实验目的

用MapReduce算法实现一个“带词频属性的倒排索引算法”，并用该算法对所提供的数据《哈利波特》全集进行测试，上传至HDFS集群。

2.Map和Reduce的设计思路

代码实现了一个基本的倒排索引功能，对于每个词语输出一个<Key,Value>对（包含词语、平均出现次数、在所出现文档中的词频）。

Map函数将输入的文本数据划分为单词并与文档名称组合成一个键值对，其中Key是单词和文档名称的组合而成的字符串，用于标识该单词在哪个文件中出现了。具体来说，Key的格式为”[单词]:文件名称"，Value则是固定为"1"，表示该单词在当前文档中出现了1次。

Reduce函数将相同单词的键值对进行合并，并计算每个单词在不同文档中的出现次数，生成一个文档列表并计算每个单词在每个文档中的平均出现次数。Key是单词，Value是包含该单词的文档列表和对应的词频信息。具体来说，Value是一个包含多个元素的字符串，每个元素代表一个文档，格式为”文件名:词频”，多个元素之间用分号";"分隔。其中，文件名包含了完整的文件路径，词频代表了该单词在该文件中出现的次数。第一个元素为平均出现次数。在Reduce函数中，生成的文档列表字符串格式为"平均出现次数,文档名称1:单词出现次数;文档名称2:单词出现次数;....."，其中平均出现次数保留两位小数。

3.Map和Reduce的伪代码

Map部分的伪代码：

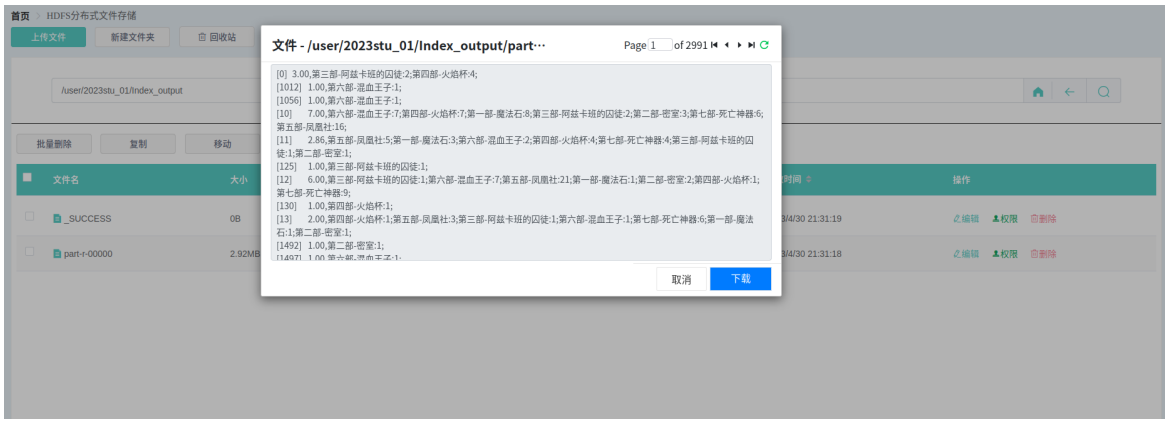
```
// Map函数
function map(key, value):
    split = getSplitObject()  // 获取当前键值对所属的Split对象
    for word in splitWords(value):
        // 构造Key，将单词和文件路径组合起来
        //
            Key格式为"[单词]:文件路径"，其中"
            [单词]"是为了区分不同的Key而添加的标识符，"文件路径"是该单词所在的文件的完整路径
        keyInfo = "[word]:" + split.getPath().toString()
        valueInfo = "1"  // 该单词在当前文档中出现1次
        emit(keyInfo, valueInfo)

// 分割文本，返回单词列表
function splitWords(text):
    return list of words in text
```

Reduce部分的伪代码：

```
// Reduce函数
function reduce(key, values):
    fileList = ""  // 用于存储文档列表的字符串
    filecnt = 0    // 统计文档数量
    sum = 0        // 统计词频总和
    for value in values:
        // 从Value中解析出文件路径和词频
        index = value.indexOf(":")
        filePath = value.substring(0, index)
        freq = value.substring(index + 1)
        fileList += value + ";"   // 将该文件信息添加到文档列表中
        filecnt += 1
        sum += freq
    aveFreq = sum / filecnt    // 计算平均词频
    fileList = String.format("%.2f", aveFreq) + "," + fileList  // 将平均词频和文档列表拼接成字符串
    emit(key, fileList)
```

4.输出结果文件的部分截图



输出结果文件在HDFS上路径： /user/2023stu_01/Index_output

5.两个单词的输出结果

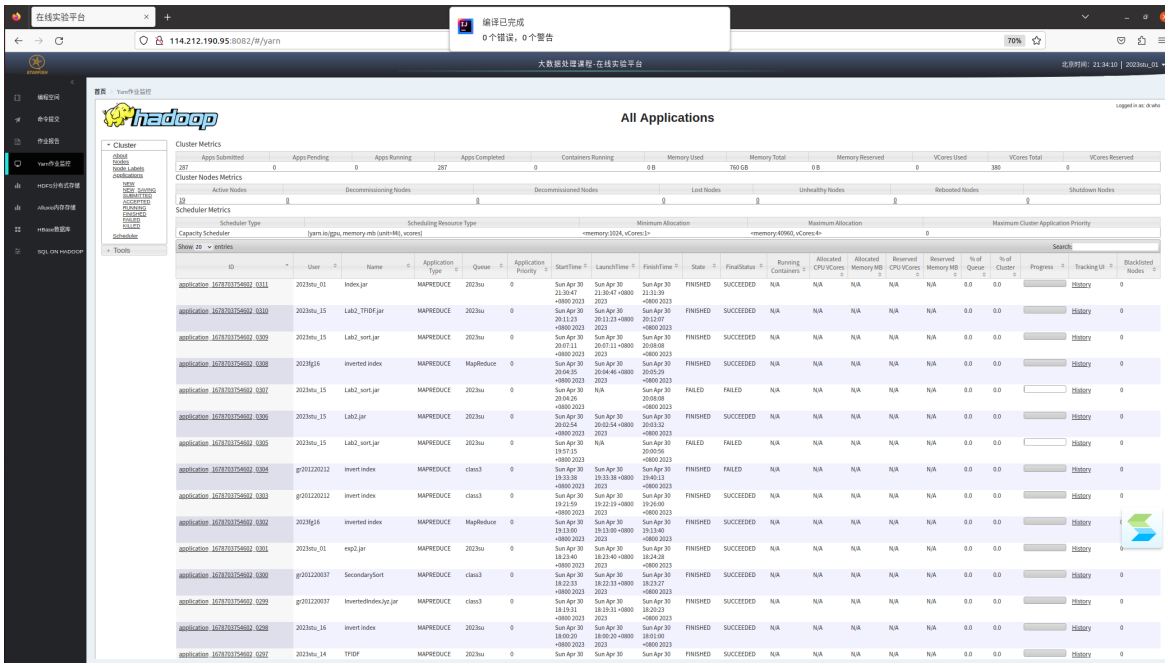
“我们”输出结果：

```
17366 [我交]  1.00,第三部-阿兹卡班的囚徒:1;
17367 [我们]  711.43,第三部-阿兹卡班的囚徒:555;第七部-死亡神器:1044;第二部-密室:364;第四部-火焰杯:677;第一部-魔法石:361;第五部-凤凰社:1323;第六部-混血王子:656;
17368 [我会]  26.29,第六部-混血王子:28;第二部-密室:11;第三部-阿兹卡班的囚徒:34;第七部-死亡神器:27;第五部-凤凰社:50;第四部-火焰杯:23;第一部-魔法石:11;
```

“什么”输出结果：

```
3465 [人鱼们]  1.00,第四部-火焰杯:1;
3466 [什]  4.00,第四部-火焰杯:1;第三部-阿兹卡班的囚徒:2;第七部-死亡神器:1;第二部-密室:7;第六部-混血王子:2;第五部-凤凰社:11;
3467 [什么]  568.00,第五部-凤凰社:1002;第七部-死亡神器:582;第四部-火焰杯:678;第三部-阿兹卡班的囚徒:506;第六部-混血王子:588;第一部-魔法石:331;第二部-密室:289;
3468 [什么样]  7.43,第二部-密室:4;第四部-火焰杯:10;第五部-凤凰社:15;第七部-死亡神器:7;第一部-魔法石:4;第三部-阿兹卡班的囚徒:8;第六部-混血王子:4;
```

6.WebUI执行报告


在线实验平台
114.212.190.95:8082/#/yarn
编译已完成
0 个错误，0 个警告
大数据处理课程-在线实验平台
All Applications
Cluster Metrics
Cluster Nodes Metrics
Scheduler Metrics

Part 2:全局排序算法（选做）

1.实验目的

写一个MapReduce Job实现对每个词语出现的平均次数进行全局排序，并输出排序结果。测试数据为《哈利波特》全集，并将运行结果上传至HDFS集群。

2.Map和Reduce的设计思路

代码实现的是对输入文件中每个单词的平均数进行排序。Map函数将每个单词的平均数和单词本身作为键值对输出，Reduce函数直接将键值对输出即可。

Map函数的设计思路：

输入类型为Object，Text；输出类型为SortUnit，NullWritable（SortUnit类封装了单词和平均数两个属性，并且实现了WritableComparable接口，用于在排序过程中比较两个SortUnit对象的大小）。读取输入文件中的每一行，将单词和平均数分别提取出来。将单词和平均数封装为SortUnit对象，作为键值对的Key。将NullWritable对象作为键值对的Value，输出到Reduce函数。

Reduce函数的设计思路：

输入类型为SortUnit，NullWritable；输出类型：SortUnit，NullWritable。直接将键值对输出即可，因为已经按照Key进行了排序。Key表示一个单词及其平均值的数据项，其中SortUnit实现了WritableComparable接口，包含了单词名称wordname和平均值aveg两个属性，可以用于排序。Value为NullWritable表示输出值为空，不包含任何信息。最后是降序排序。

3.Map和Reduce的伪代码

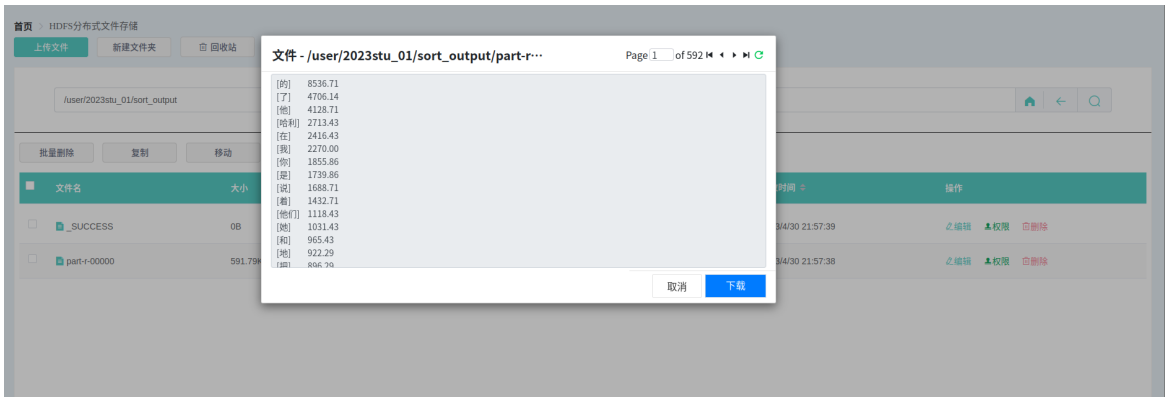
Map部分的伪代码：

```
function map(key, value):
    fields = split value by tab
    wordname = fields[0]
    index = find index of comma in fields[1]
    aveg = convert substring of fields[1] from start to index to double
    create SortUnit object with wordname and aveg
    emit SortUnit object as key with NullWritable as value
```

Reduce部分的伪代码：

```
function reduce(key, values):
    emit key as key with NullWritable as value
```

4.输出结果文件的部分截图



输出结果文件在HDFS上路径：/user/2023stu_01/sort_output

5.两个单词的输出结果

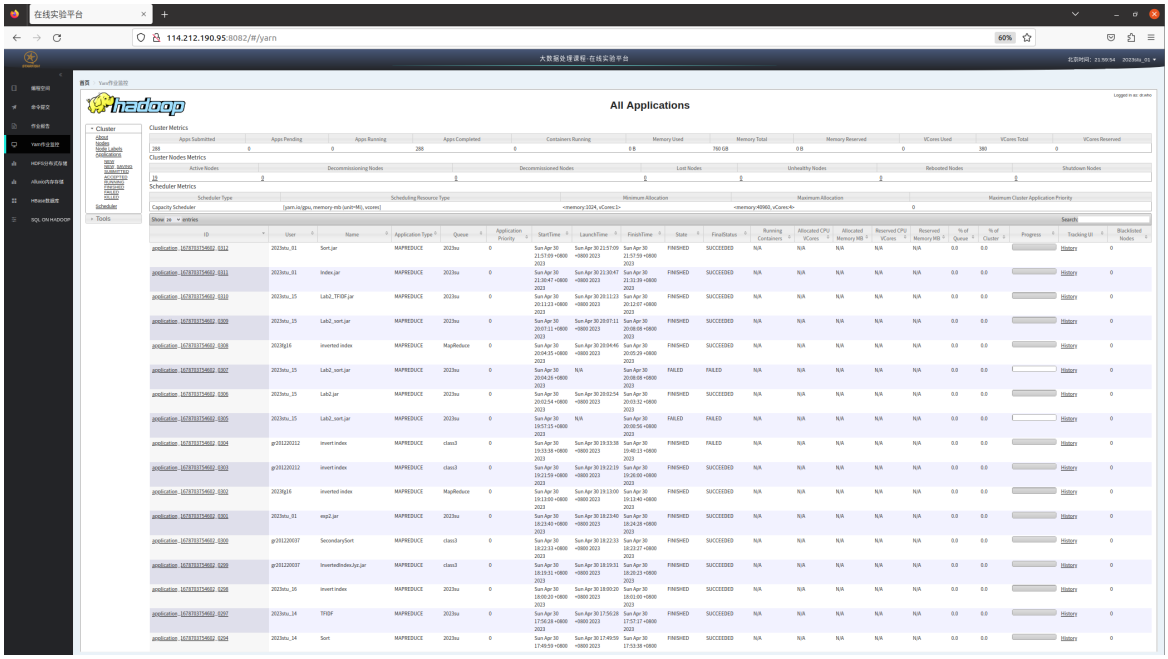
“我们”输出结果：

```
16 [就]    851.57
17 [不]    791.43
18 [赫敏]  742.71
19 [都]    717.14
20 [我们]  711.43
21 [上]    706.00
22 [罗恩]  676.00
```

“什么”输出结果：

```
26 [里]    608.57
27 [那]    578.57
28 [也]    569.14
29 [什么]  568.00
30 [有]    563.00
31 [到]    557.43
32 [知道]  532.43
33 [人]    525.00
```

6.WebUI执行报告



Part 3 词语的TF-IDF计算（选做）

1.实验目的

设计一个MapReduce Job计算文档内每个词语的TF-IDF，其中TF定义为词语在作品中出现的次数之和，IDF定义为 $\text{IDF}(\text{词语}) = \log\left(\frac{\text{语料库文档总数}}{\text{包含该词的文档数}+1}\right)$

测试数据为《哈利波特》全集，并将运行结果上传至HDFS集群。

2.Map和Reduce的设计思路

在Map阶段，将每个单词和它所在的文件名（或文件路径）组成一个Key，词频（即出现次数）组成Value，作为Mapper的输出。这里将Key和Value都定义为Text类型。

在Combine阶段，将Map阶段输出的所有<key,value>对按照key进行合并，统计每个单词在每个文件中的总词频，重新组成新的<key,value>对。这里也将Key和Value都定义为Text类型。

在Reduce阶段，将Combine阶段输出的<key,value>对按照单词进行合并，统计每个单词在所有文件中出现的次数，计算每个单词在每个文件中的TF-IDF值，并将结果以“文件名,单词,TF-IDF值”为格式输出，作为Reducer的最终输出。这里将Key定义为Text类型，Value定义为NullWritable类型，因为Value本身不需要输出。

3.Map和Reduce的伪代码

```
Mapper(Object key, Text value, Context context):
    split = context.getInputSplit()  # 获取当前输入记录所属的 FileSplit 对象
    for word in value:
        # 每个单词和当前文档所属的文件名组成一个 key，以及出现次数 1 组成一个 value，写入到 Context 中
        key = word + ":" + split.getPath().toString()
        value = "1"
        context.write(key, value)
```

Map部分的伪代码：

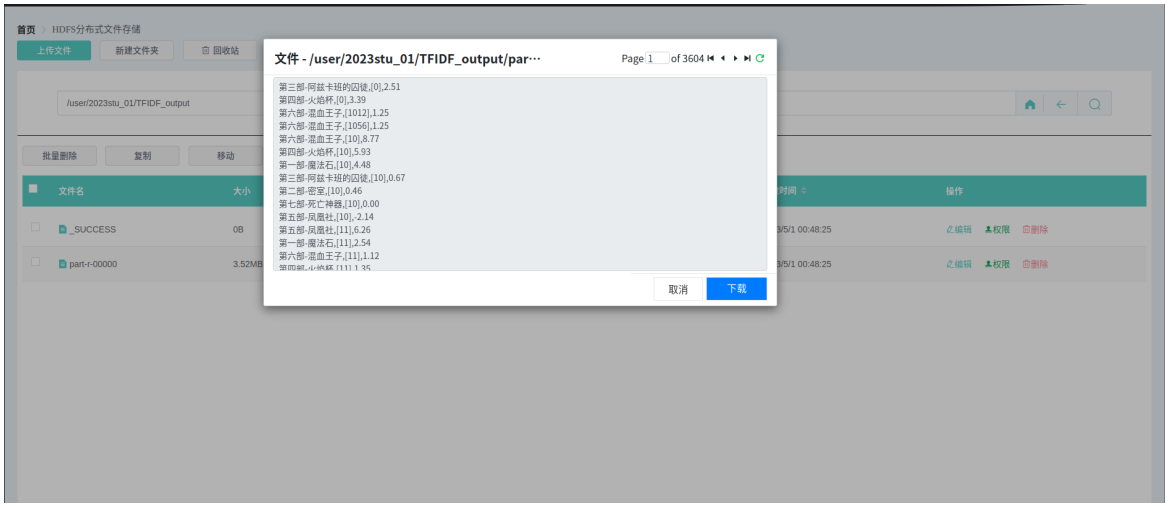
Combine部分的伪代码：

```
Reducer(Text key, Iterable<Text> values, Context context):
    int sum = 0
    for value in values:
        sum += int(value)  # 统计同一个单词在当前文档中出现的次数
    # 把 URL 和该单词在该文档中的出现次数组合成一个新的 value，以单词作为 key，写入到 Context 中
    context.write(key, sum)
```

Reduce部分的伪代码：

```
Reducer(Text key, Iterable<Text> values, Context context):
    docList = ""
    fileCnt = 0
    for value in values:
        fileCnt += 1  # 统计包含当前单词的文档数
        tf = int(value.split(":")[1])
        idf = log(n / (fileCnt + 1))  # 计算逆文档频率
        tf_idf = tf * idf  # 计算 tf-idf 值
        docList += value.split(":")[0] + ","  # 把文档名添加到 docList 中
        result = value.split(":")[0] + "," + key + "," + str(tf_idf)  # 构造输出结果
        context.write(result, NullWritable.get())
```

4.输出结果文件的部分截图



输出结果文件在HDFS上路径：/user/2023stu_01/TFIDF_output

5.两个单词的输出结果

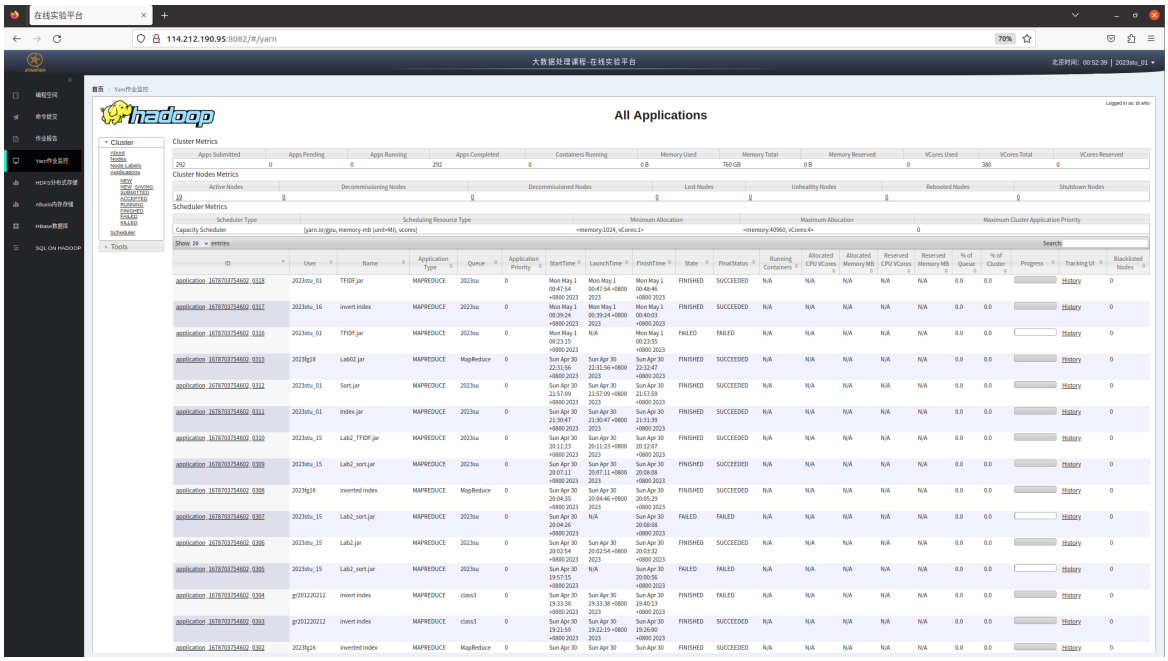
“我们”输出结果：

```
44207 第三部-阿兹卡班的囚徒,[我交],1.25
44208 第三部-阿兹卡班的囚徒,[我们],695.28
44209 第七部-死亡神器,[我们],884.58
44210 第二部-密室,[我们],203.70
44211 第四部-火焰杯,[我们],227.79
44212 第一部-魔法石,[我们],55.65
44213 第五部-凤凰社,[我们],0.00
44214 第六部-混血王子,[我们],-87.60
44215 第六部-混血王子,[我会],35.08
```

“什么”输出结果：

```
8977 第六部-混血王子,[什],0.31
8978 第五部-凤凰社,[什],0.00
8979 第五部-凤凰社,[什么],1255.27
8980 第七部-死亡神器,[什么],493.13
8981 第四部-火焰杯,[什么],379.42
8982 第三部-阿兹卡班的囚徒,[什么],170.25
8983 第六部-混血王子,[什么],90.64
8984 第一部-魔法石,[什么],0.00
8985 第二部-密室,[什么],-38.59
8986 第二部-密室,[什么样],5.01
```

6.WebUI执行报告


在线实验平台
114.212.190.95:8082/#/yarn
大数据处理课程-在线实验平台
All Applications
Cluster Metrics
Cluster Nodes Metrics
Scheduler Metrics