

大数据处理综合实验-实验2实验报告

小组15

一、JAR包执行方式

说明：本次实验我们把所有功能打包在一个jar包内，通过指定主类选择功能。假定jar包的位置在lab2/lab2.jar位置。

注意：选做功能1（排序）在执行时需要以基础功能的输出作为输入，请确保执行前已经执行过倒排索引。

基础功能（倒排索引）：

`hadoop jar lab2/lab2.jar InvertedIndexer input_path output_dir_1`

选做功能1（排序）：

`hadoop jar lab2/lab2.jar SortedCounter output_dir_1 output_dir_2`

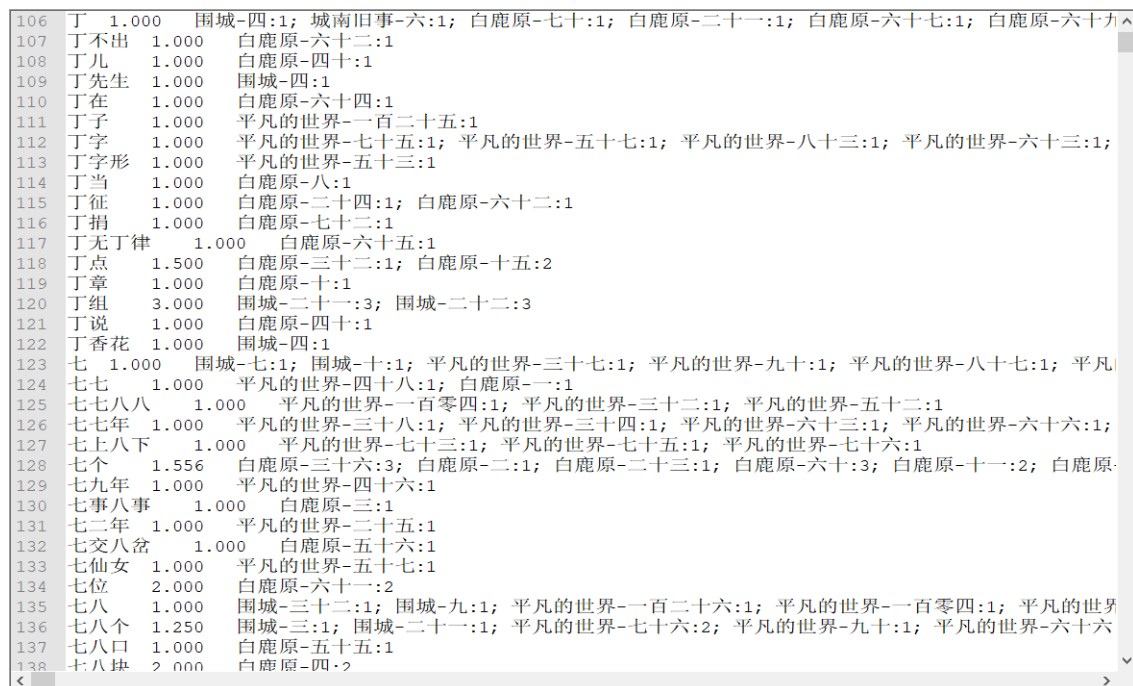
选做功能2（TF-IDF）：

`hadoop jar lab2/lab2.jar TFIDF input_path output_dir_3`

二、实验结果及输出文件路径

1. 基础功能

输出文件路径：/user/2021sg15/lab2/out1

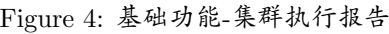


106	丁	1.000	围城-四:1; 城南旧事-六:1; 白鹿原-七十:1; 白鹿原-二十一:1; 白鹿原-六十七:1; 白鹿原-六十九:1
107	丁不出	1.000	白鹿原-六十二:1
108	丁儿	1.000	白鹿原-四十:1
109	丁先生	1.000	围城-四:1
110	丁在	1.000	白鹿原-六十四:1
111	丁子	1.000	平凡的世界-一百二十五:1
112	丁字	1.000	平凡的世界-七十五:1; 平凡的世界-五十七:1; 平凡的世界-八十三:1; 平凡的世界-六十三:1;
113	丁字形	1.000	平凡的世界-五十三:1
114	丁当	1.000	白鹿原-八:1
115	丁征	1.000	白鹿原-二十四:1; 白鹿原-六十二:1
116	丁捐	1.000	白鹿原-七十二:1
117	丁无丁仲	1.000	白鹿原-六十五:1
118	丁点	1.500	白鹿原-三十二:1; 白鹿原-十五:2
119	丁章	1.000	白鹿原-十:1
120	丁组	3.000	围城-二十一:3; 围城-二十二:3
121	丁说	1.000	白鹿原-四十:1
122	丁香花	1.000	围城-四:1
123	七	1.000	围城-七:1; 围城-十:1; 平凡的世界-三十七:1; 平凡的世界-九十:1; 平凡的世界-八十七:1; 平凡
124	七七	1.000	平凡的世界-四十八:1; 白鹿原-一:1
125	七七八八	1.000	平凡的世界-一百零四:1; 平凡的世界-三十二:1; 平凡的世界-五十二:1
126	七七年	1.000	平凡的世界-三十八:1; 平凡的世界-三十四:1; 平凡的世界-六十三:1; 平凡的世界-六十六:1;
127	七上八下	1.000	平凡的世界-七十三:1; 平凡的世界-七十五:1; 平凡的世界-七十六:1
128	七个	1.556	白鹿原-三十六:3; 白鹿原-二:1; 白鹿原-二十三:1; 白鹿原-六十:3; 白鹿原-十一:2; 白鹿原-
129	七九年	1.000	平凡的世界-四十六:1
130	七事八事	1.000	白鹿原-三:1
131	七二年	1.000	平凡的世界-二十五:1
132	七交八岔	1.000	白鹿原-五十六:1
133	七仙女	1.000	平凡的世界-五十七:1
134	七位	2.000	白鹿原-六十一:2
135	七八	1.000	围城-三十二:1; 围城-九:1; 平凡的世界-一百二十六:1; 平凡的世界-一百零四:1; 平凡的世界
136	七八个	1.250	围城-三:1; 围城-二十一:1; 平凡的世界-七十六:2; 平凡的世界-九十:1; 平凡的世界-六十六
137	七八口	1.000	白鹿原-五十五:1
138	七八块	2.000	白鹿原-四:2

Figure 1: 基础功能-结果文件部分截图

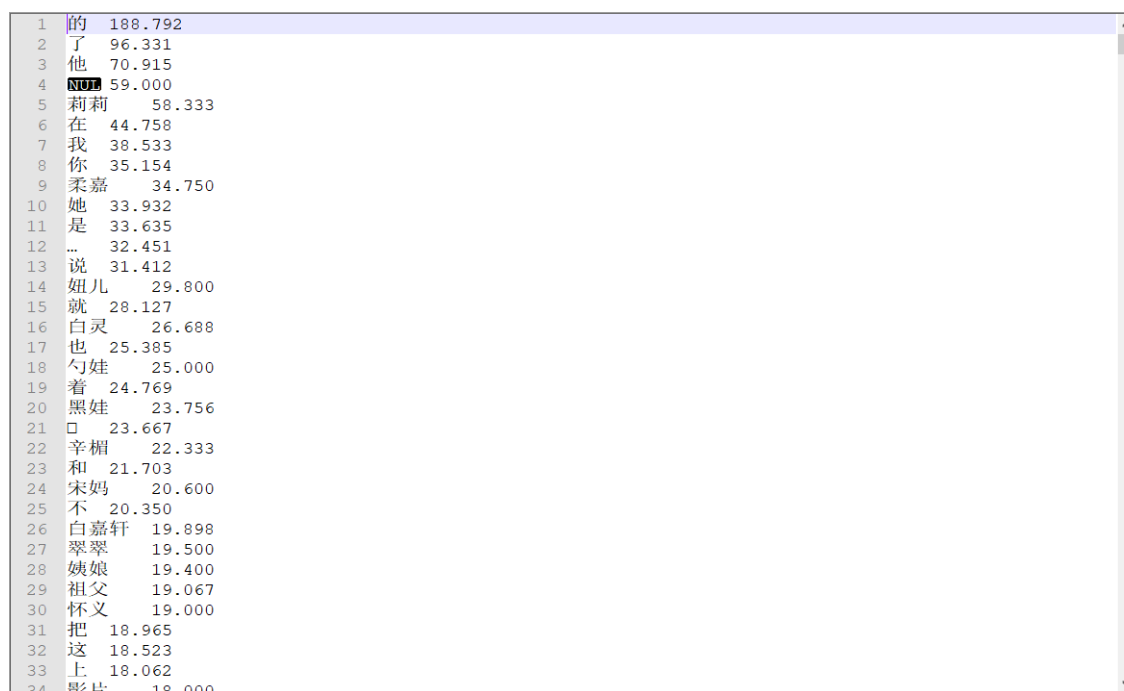
Figure 2: 基础功能-结果文件-“我们”

Figure 3: 基础功能-结果文件-“什么”



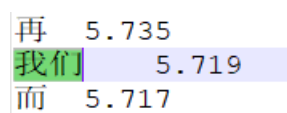
2

输出文件路径: /user/2021sg15/lab2/out2



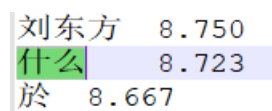
1	的	188.792
2	了	96.331
3	他	70.915
4	NOT	59.000
5	莉莉	58.333
6	在	44.758
7	我	38.533
8	你	35.154
9	柔嘉	34.750
10	她	33.932
11	是	33.635
12	...	32.451
13	说	31.412
14	妞儿	29.800
15	就	28.127
16	白灵	26.688
17	也	25.385
18	勺娃	25.000
19	着	24.769
20	黑娃	23.756
21	口	23.667
22	辛楣	22.333
23	和	21.703
24	宋妈	20.600
25	不	20.350
26	白嘉轩	19.898
27	翠翠	19.500
28	姨娘	19.400
29	祖父	19.067
30	怀义	19.000
31	把	18.965
32	这	18.523
33	上	18.062
34	影片	18.000

Figure 5: 选做1 (排序) -结果文件部分截图



再	5.735
我们	5.719
而	5.717

Figure 6: 选做1 (排序) -结果文件-“我们”



刘东方	8.750
什么	8.723
於	8.667

Figure 7: 选做1 (排序) -结果文件-“什么”



Logged in as: drwho

Application application_1626070675586_10052

Cluster

About

Nodes

Node Labels

Applications

NEW

NEW SAVING

SUBMITTED

ACCEPTED

RUNNING

FINISHED

FAILED

KILLED

Scheduler

Tools

Kill Application

Application Overview

User: 2021sg15

Name: sort count

Application Type: MAPREDUCE

Application Tags:

YarnApplicationState: FINISHED

Queue: root.2021s

FinalStatus Reported by AM: SUCCEEDED

Started: Sat Apr 16 15:40:17 +0800 2022

Elapsed: 19sec

Tracking URL: History

Diagnostics:

Application Metrics

Total Resource Preempted: <memory:0, vCores:0>

Total Number of Non-AM Containers Preempted: 0

Total Number of AM Containers Preempted: 0

Resource Preempted from Current Attempt: <memory:0, vCores:0>

Number of Non-AM Containers Preempted from Current Attempt: 0

Aggregate Resource Allocation: 135055 MB-seconds, 34 vcore-seconds

Show 20 entries

Attempt ID

Started

Node

Logs

Search:

Blacklisted Nodes

appatempt_1626070675586_10052_000001

Sat Apr 16 15:40:17 +0800 2022

http://slave007:8042

Logs

N/A

Showing 1 to 1 of 1 entries

First Previous 1 Next Last

Figure 8: 选做1（排序）-集群执行报告

3. 选做2（TF-IDF）

输出文件路径：/user/2021sg15/lab2/out3

```
1 平凡的世界,100,287.185
2 城南旧事,(,4.174
3 白鹿原,(,8.349
4 城南旧事,),4.174
5 白鹿原,),8.349
6 平凡的世界,*,4.868
7 城南旧事,/ ,4.174
8 边城,/ ,4.174
9 平凡的世界,/ ,4.174
10 围城,0,4.868
11 城南旧事,00,4.868
12 平凡的世界,1,11.307
13 白鹿原,1,18.845
14 城南旧事,10,3.769
15 边城,10,3.769
16 平凡的世界,10,11.307
17 白鹿原,100,9.735
18 平凡的世界,11,12.523
19 白鹿原,1112,4.868
20 平凡的世界,12,12.523
21 白鹿原,1200,4.868
22 白鹿原,1211,4.868
23 平凡的世界,13,12.523
24 平凡的世界,14,12.523
25 白鹿原,1400,4.868
26 平凡的世界,15,11.854
27 白鹿原,15,11.854
28 平凡的世界,16,12.523
29 平凡的世界,17,12.523
30 平凡的世界,18,12.523
31 平凡的世界,19,12.523
32 城南旧事,1960,9.735
33 白鹿原,1978,4.868
34 城南旧事,1979,4.868
```

Figure 9: 选做2（TF-IDF）-结果文件部分截图

```

围城,我们,44.631
城南旧事,我们,47.542
边城,我们,6.792
平凡的世界,我们,153.784
白鹿原,我们,28.865

```

Figure 10: 选做2 (TF-IDF) -结果文件-“我们”

```

围城,什么,8.475
城南旧事,什么,4.319
边城,什么,2.522
平凡的世界,什么,30.959
白鹿原,什么,5.253

```

Figure 11: 选做2 (TF-IDF) -结果文件-“什么”

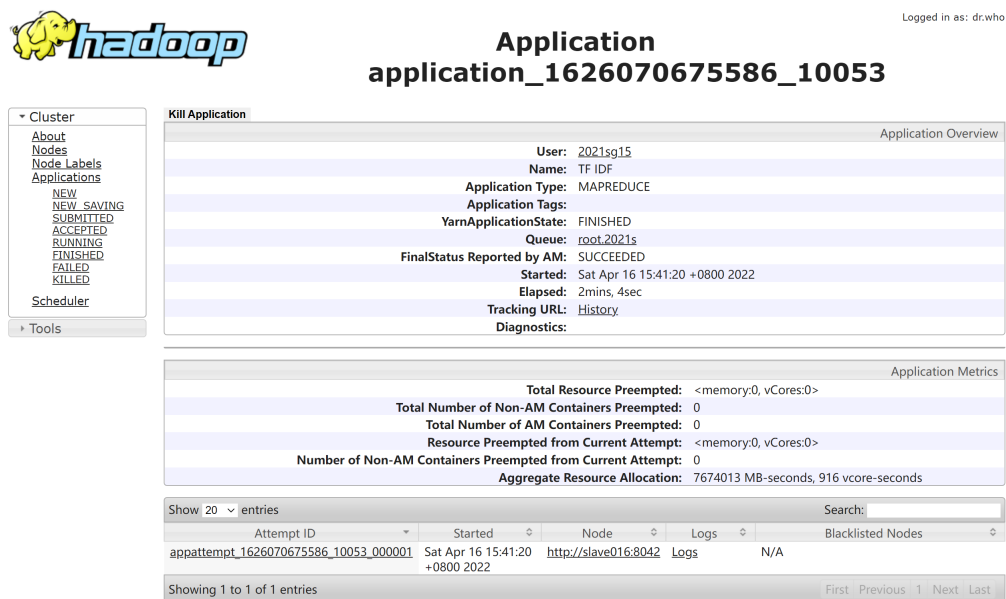


Figure 12: 选做2 (TF-IDF) -集群执行报告

三、基础功能

设计思路

Mapper类接受和发送的Key, Value类型: <Object, Text, Text, IntWritable>. 发送的key的含义是单词+文档名, 发送的value的含义是key中单词在该文档的出现次数.这里需要重写Partitioner类, 按照key中的单词作为划分标准.

Reducer类接受和发送的Key, Value类型: <Text, IntWritable, Text, Text>. 包含当前单词的文档数的统计方法为: 使用一个类成员变量`fileNumber`记录, 在当前单词没有发生变化时自增1, 否则在完成均值和倒排索引的计算后置0. 当前单词的总词频的计算方法为: 在当前单词没有发生变化时, 累加`values`中所有的值, 同时在这一过程中使用StringBuiler记录当前文档的倒排索引及其词频.在当前词语发生变化时及完成所有项的统计后, 计算均值并写出结果.

伪代码

Algorithm 1 Mapper for InvertedIndexer

```
1: method Map(docid  $dn$ , doc  $d$ )
2: Let  $F$  be a new AssociatedArray.
3: for each  $t \in d$  do
4:    $F\{t\} = F\{t\} + 1$ ;
5: end for
6: for each  $t \in F$  do
7:    $Emit(< t, docid >, F\{t\})$ ;
8: end for
9: return
```

Algorithm 2 Reducer for InvertedIndexer

```
1: method Setup(docid  $dn$ , doc  $d$ )
2:  $fileNumber := 0$ ;
3:  $frequency := 0$ ;
4:  $sumFrequency := 0$ ;
5:  $t_{prev} := \emptyset$ ;
6:  $P := \text{new } PostingsList$ ;
7:
8: method Reduce( $< t, docid >, tf[]$ )
9: if  $t \neq t_{prev} \wedge t_{prev} \neq \emptyset$  then
10:    $average := sumFrequency / fileNumber$ ;
11:    $emit(prev, < average, P >)$ ;
12:    $P.reset()$ ;
13:    $fileNumber = 0$ ;
14:    $sumFrequency = 0$ ;
15: end if
16:  $t_{prev} = t$ ;
17:  $fileNumber = fileNumber + 1$ ;
18:  $frequency = 0$ ;
19: for each  $f \in tf$  do
20:    $frequency = frequency + f$ ;
21: end for
22:  $sumFrequency = sumFrequency + frequency$ ;
23:  $P.add(< docid, frequency >)$ ;
24:
25: method Close()
26:  $average := sumFrequency / fileNumber$ ;
27:  $emit(prev, < average, P >)$ ;
28:
29: function TF_IDF( $PostingsList < docid, frequency > P, IDF, word$ )
30:  $res := \text{new } List$ ;
31:  $wordfile\_freq\_map := \text{new } AssociatedArray$ ;
32: Let All elements in  $wordfile\_freq\_map$  has default value 0.
33: for each  $< docid, frequency > \in P$  do
34:    $fileName := docid.substr(0, docid.indexOf('-'))$ ;
35:    $wordfile\_freq\_map[fileName] = wordfile\_freq\_map[fileName] + frequency$ ;
36: end for
37: for each  $< fileName, frequency > \in wordfile\_freq\_map$  do
38:    $res.add(< fileName, word, IDF * frequency >)$ ;
39: end for
40: return  $res$ ;
```

四、选做内容

A. 排序出现次数的排序

设计思路

以基础功能的输出为输入：

Mapper类接受和发送的Key, Value类型： $\langle \text{Object}, \text{Text}, \text{Text}, \text{Text} \rangle$ 。发送的value的含义是单词，发送的key的含义是该单词平均出现次数。重写WritableComparator的compare方法，以单词平均出现次数的降序进行排列。

Reducer类接受和发送的Key, Value类型： $\langle \text{Text}, \text{Text}, \text{Text}, \text{Text} \rangle$ 。将得到的 $\langle \text{value}$ （单词）和 key （平均出现次数） \rangle 作为输出。

伪代码

Algorithm 3 Mapper for SortedCounter

```
1: method Map(docid dn, doc d)
2: Let  $F$  be a new AssociatedArray.
3: for each  $line \langle word, averageFrequency, postingList \langle docid, frequency \rangle \rangle \in d$  do
4:   Emit( $averageFrequency, word$ );
5: end for
6: return
```

Algorithm 4 WritableComparator for SortedCounter

```
1: method Compare(WritableComparable a, WritableComparable b)
2: Let double  $m$  be the value of  $a$ .
3: Let double  $n$  be the value of  $b$ .
4: return b.compareTo( $a$ );
```

Algorithm 5 Reducer for SortedCounter

```
1: method Reduce( $frequency, words[]$ )
2: for each  $word \in words$  do
3:   Emit( $word, frequency$ );
4: end for
```

B. 每个作品的每个词语的TF-IDF

设计思路

Mapper类接受和发送的Key, Value类型： $\langle \text{Object}, \text{Text}, \text{Text}, \text{IntWritable} \rangle$ 。发送的key的含义是单词+文档名，发送的value的含义是key中单词在该文档的出现次数。这里需要重写Partitioner类，按照key中的单词作为划分标准。

Reducer类接受和发送的Key, Value类型： $\langle \text{Text}, \text{IntWritable}, \text{Text}, \text{Text} \rangle$ 。语料库文档总数在main方法中由FileSystem获取，并存储在Configuration中。在Reducer中通过上下文Context获取该项的值。包含当前单词的文档数的统计方法和基础功能中的实现相类似，这里不再赘述。得到了单词word对应的所有[文档-词频]以及该单词的IDF，遍历每个文档-词频计算其TF-IDF即可。注意对同一作品不同文档的词频求和才能得到正确的TF-IDF。

伪代码

Algorithm 6 Mapper for TF-IDF

```
1: method Map(docid dn, doc d)
2: Let F be a new AssociatedArray.
3: for each t ∈ d do
4:   F{t} = F{t} + 1;
5: end for
6: for each t ∈ F do
7:   Emit(< t, docid >, F{t});
8: end for
9: return
```

Algorithm 7 Reducer for TF-IDF

```
1: method Setup(docid dn, doc d)
2: Let srcFileCnt be the number of input files.
3: fileNumber := 0;
4: frequency := 0;
5: tprev := ∅;
6: P := new PostingsList;
7: wordfile_freq_map := new AssociatedArray;
8: Let All elements in wordfile_freq_map has default value 0.
9:
10: method Reduce(< t, docid >, tf[])
11: if t ≠ tprev ∧ tprev ≠ ∅ then
12:   IDF := log( $\frac{srcFileCnt}{fileNumber+1}$ );
13:   wordfile_freq_map.clear();
14:   for each < docid, frequency > ∈ P do
15:     fileName := docid.substr(0, docid.indexOf('-'));
16:     wordfile_freq_map[fileName] = wordfile_freq_map[fileName] + frequency;
17:   end for
18:   for each < fileName, frequency > ∈ wordfile_freq_map do
19:     Emit(< fileName, word >, IDF * frequency >);
20:   end for
21:   P.reset();
22:   fileNumber = 0;
23: end if
24: tprev = t;
25: fileNumber = fileNumber + 1;
26: frequency = 0;
27: for each f ∈ tf do
28:   frequency = frequency + f;
29: end for
30:
31: P.add(< docid, frequency >);
32:
33: method Close()
34: wordfile_freq_map.clear();
35: for each < docid, frequency > ∈ P do
36:   fileName := docid.substr(0, docid.indexOf('-'));
37:   wordfile_freq_map[fileName] = wordfile_freq_map[fileName] + frequency;
38: end for
39: for each < fileName, frequency > ∈ wordfile_freq_map do
40:   Emit(< fileName, word >, IDF * frequency >);
41: end for
```
