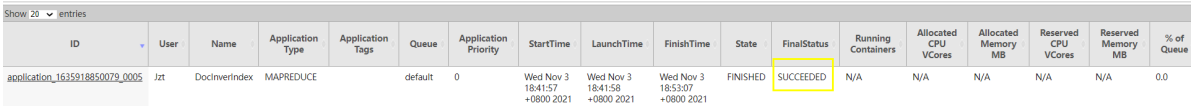
《金融大数据处理技术》作业6报告

191870068 嵇泽同

一、实验结果

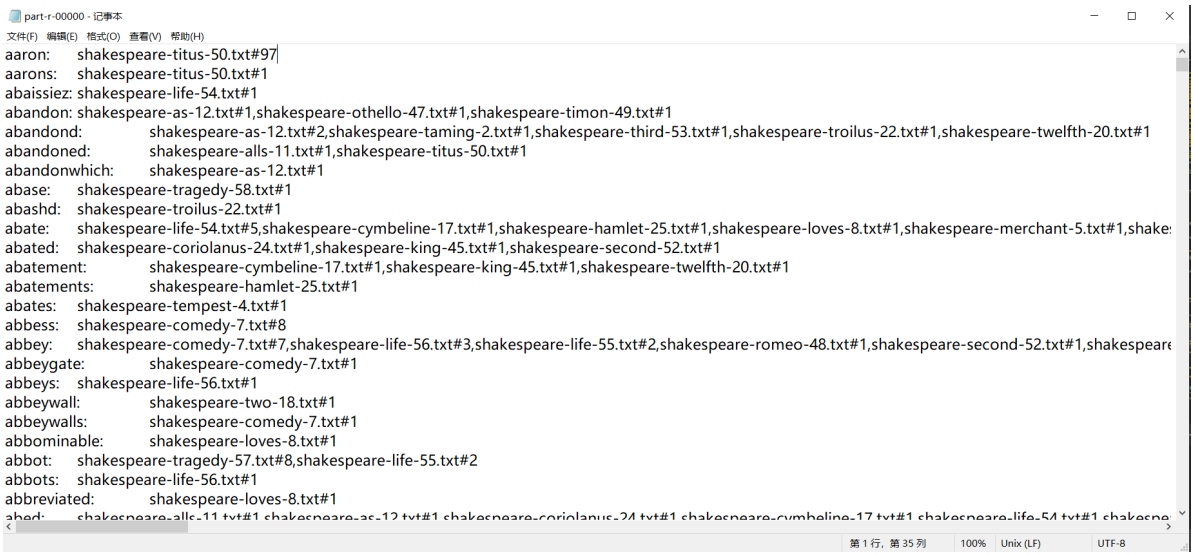
yarn web截图：



Show 20 entries

ID	User	Name	Application Type	Application Tags	Queue	Application Priority	StartTime	LaunchTime	FinishTime	State	FinalStatus	Running Containers	Allocated CPU VCores	Allocated Memory MB	Reserved CPU VCores	Reserved Memory MB	% of Queue
application_1635918850079_0005	Jzt	DocInverIndex	MAPREDUCE		default	0	Wed Nov 3 18:41:57 +0800 2021	Wed Nov 3 18:41:58 +0800 2021	Wed Nov 3 18:53:07 +0800 2021	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0

输出结果展示：



part-r-00000 - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
aaron:      shakespeare-titus-50.txt#97
aarons:     shakespeare-titus-50.txt#1
abaissiez: shakespeare-life-54.txt#1
abandon: shakespeare-as-12.txt#1,shakespeare-othello-47.txt#1,shakespeare-timon-49.txt#1
abandond:           shakespeare-as-12.txt#2,shakespeare-taming-2.txt#1,shakespeare-third-53.txt#1,shakespeare-troilus-22.txt#1,shakespeare-twelfth-20.txt#1
abandoned:          shakespeare-alls-11.txt#1,shakespeare-titus-50.txt#1
abandonwhich:       shakespeare-as-12.txt#1
abase:      shakespeare-tragedy-58.txt#1
abashd:     shakespeare-troilus-22.txt#1
abate:      shakespeare-life-54.txt#5,shakespeare-cymbeline-17.txt#1,shakespeare-hamlet-25.txt#1,shakespeare-loves-8.txt#1,shakespeare-merchant-5.txt#1,shake
abated:     shakespeare-coriolanus-24.txt#1,shakespeare-king-45.txt#1,shakespeare-second-52.txt#1
abatement:          shakespeare-cymbeline-17.txt#1,shakespeare-king-45.txt#1,shakespeare-twelfth-20.txt#1
abatements:         shakespeare-hamlet-25.txt#1
abates:     shakespeare-tempest-4.txt#1
abbess:     shakespeare-comedy-7.txt#8
abbey:      shakespeare-comedy-7.txt#7,shakespeare-life-56.txt#3,shakespeare-life-55.txt#2,shakespeare-romeo-48.txt#1,shakespeare-second-52.txt#1,shakespeare
abbeygate:          shakespeare-comedy-7.txt#1
abbeys:     shakespeare-life-56.txt#1
abbeywall:          shakespeare-two-18.txt#1
abbeywalls:         shakespeare-comedy-7.txt#1
abbominable:        shakespeare-loves-8.txt#1
abbot:      shakespeare-tragedy-57.txt#8,shakespeare-life-55.txt#2
abbots:     shakespeare-life-56.txt#1
abbreviated:        shakespeare-loves-8.txt#1
abed:       shakespeare-alls-11.txt#1 shakespeare-as-12.txt#1 shakespeare-coriolanus-24.txt#1 shakespeare-cymbeline-17.txt#1 shakespeare-life-54.txt#1 shakespe
```

第 1 行，第 35 列　100%　Unix (LF)　UTF-8

二、设计思路

本次实现的基础是课上介绍的带词频属性的文档倒排算法，在此基础上实现的主要额外功能就是对于每个单词，单词的索引按照单词在该文档中出现的次数从大到小排序（输出按字典序排序可以按课上说的把单词加入key实现；单词忽略大小写、标点等可以按照作业5中的方式实现）。因此下面主要说明单词索引按出现次数排序的设计思路。

要想将出现次数排序，一个很自然的想法就是把出现次数放到key里面。但是什么时候放到key里面是一个问题。此次作业我采用的是Chain方法，第一个Mapper起到分词作用，输出的值一般固定为1；要想将出现次数放到key里面，首先就需要统计（sum）出出现次数，这一统计过程一般可以在combine或者reduce过程中实现。如果在reduce中统计出现次数，那么即使将统计出来的次数放在key中，并且在其后再新填一个Mapper（Chain中只能存在一个Reducer，而可以有多个Mapper），那么由于程序按照key自动排序是在shuffle中进行的，而只有在Mapper到Reducer时才会有shuffle过程，Reducer后再添加Mapper，那么Mapper接收到的数据顺序就是和Reducer的输出顺序一致的，因此即

使在reduce中将出现次数加入key中，对于排序也没有用。而如果在combiner中进行统计，那么需要注意一点，combiner的输出不仅要和reducer的输入格式一致，同时也要和mapper的输出格式一致，这也是为什么combiner常常被用来进行一个如求和、求均值等不改变数据性质的工作的原因。如果要在combiner中进行统计并且将结果放到key中输出，那么combiner的key和mapper的key就很有可能在格式上不同从而不满足要求，除非特意专门自定义某种key类，那么理论上是可行的，但是也往往会导致传输数据的冗余。创建多个job的方法自然也是可行的，但是也会造成多次IO降低性能，此次我采用了Chain还是希望能够在一个job中完成。

最终，在本次作业中我采用的方法是在reduce中手动设计算法进行排序，而非依靠shuffle过程中的自动排序。这样可能导致的问题是如果数据量过大则可能会导致内存不足。但是考虑到需要排序的是单词在每个文档中的出现次数，文档次数是有限的，一般不会过大，因此本次作业中采用这种方法没什么问题，如果以后遇到特殊情况的话就要再重新考虑设计了。注意到在将出现次数进行排序的同时还要将出现次数和对应的文档绑定起来，对此我采用的方法是自定义类存储文档名和对应出现次数，然后自定义针对该类的Comparator，并运用ArrayList的sort()方法进行排序。

除此之外，由于此次作业比较简单，再加上经历了作业5后又积累了一定的经验，因此也没有什么别的特别需要提及的了。

三、作业过程中遇到的主要困难

在作业5的报告中我提到当时就考虑用Chain，但是由于参数始终不对的问题未能成功，受限于时间而临时采用了链式job的实现方法。此次作业，我重新开始探索Chain的用法。

以下是报错信息：

```
ChainMapper.addMapper(jobConf,TokenizerMapper.class,Object.class,Text.class,WordDocid.class,IntWritable.class,
        byValue: true,map1Conf);
```

	Required type	Provided
klass:	Class<? extends Mapper<K1, V1, K2, V2>>	Class<TokenizerMapper>
inputKeyClass:	Class<? extends K1>	Class<Object>
inputValueClass:	Class<? extends V1>	Class<Text>
outputKeyClass:	Class<? extends K2>	Class<WordDocid>
outputValueClass:	Class<? extends V2>	Class<IntWritable>

reason: no instance(s) of type variable(s) K1, K2, V1, V2 exist so that TokenizerMapper conforms to Mapper<K1, V1, K2, V2>

```
public static class DocInverIndex.TokenizerMapper
extends Mapper<Object, Text, DocInverIndex.WordDocid, IntWritable>
```

怎么看都不知道问题出在哪里，参数的类型完全一致，而网上关于Chain的资料也不多。反复仔细观察参数要求后，最终发现问题所在——要求继承的Mapper是来自mapred包：

```
addMapper(@NotNull org.apache.hadoop.mapred.JobConf job,
          Class<? extends org.apache.hadoop.mapred.Mapper<K1, V1, K2, V2>> klass,
          Class<? extends K1> inputKeyClass,
          Class<? extends V1> inputValueClass,
          Class<? extends K2> outputKeyClass,
          Class<? extends V2> outputValueClass,
          boolean byValue,
          @Nullable org.apache.hadoop.mapred.JobConf mapperConf)
```

而我实现的Mapper类是继承的下面这个，来自mapreduce包：

```java
package org.apache.hadoop.mapreduce;

import ...

@Public
@Stable
public class Mapper<KEYIN, VALUEIN, KEYOUT, VALUEOUT> {
```

而非下面这个来自mapred包的Mapper：

```java
package org.apache.hadoop.mapred;

import ...

@Public
@Stable
public interface Mapper<K1, V1, K2, V2> extends JobConfigurable, Closeable {
```

而之所以会要求用mapred包的Mapper而非mapreduce包的Mapper，原因在于我在代码中引入的ChainMapper也是来自mapred包（当时引用的时候忘了从哪里复制的引用代码，自己也没注意）：

```java
import org.apache.hadoop.mapred.lib.ChainMapper;
```

看起来似乎mapred包和mapreduce包里有不少看起来“重复”的内容，而且还不兼容，所以我就上网查找了相关内容：

> 结果令小菜很失望，就找到了一个符合理想的帖子。但是通过这个帖子，小菜知道了，**mapred代表的是hadoop旧API，而mapreduce代表的是hadoop新的API。**
>
> OK，小菜在google输入框中输入“hadoop新旧API的区别”，结果很多。看了之后，又结合权威指南归结如下：
>
> 1. 首先第一条，也是小菜今天碰到这些问题的原因，新旧API不兼容。所以，以前用旧API写的hadoop程序，如果旧API不可用之后需要重写，也就是上面我的程序需要重写，如果旧API不能用的话，如果真不能用，这个有点儿小遗憾！
>
> 2. 新的API倾向于使用抽象类，而不是接口，使用抽象类更容易扩展。例如，我们可以向一个抽象类中添加一个方法(用默认的实现)而不用修改类之前的实现方法。因此，在新的API中，Mapper和Reducer是抽象类。
>
> 3. 新的API广泛使用context object(上下文对象)，并允许用户代码与MapReduce系统进行通信。例如，在新的API中，MapContext基本上充当着JobConf的OutputCollector和Reporter的角色。
>
> 4. 新的API同时支持"推"和"拉"式的迭代。在这两个新老API中，键/值记录对被推mapper中，但除此之外，新的API允许把记录从map()方法中拉出，这也适用于reducer。分批处理记录是应用"拉"式的一个例子。
>
> 5. 新的API统一了配置。旧的API有一个特殊的JobConf对象用于作业配置，这是一个对于Hadoop通常的Configuration对象的扩展。在新的API中，这种区别没有了，所以作业配置通过Configuration来完成。作业控制的执行由Job类来负责，而不是JobClient，并且JobConf和JobClient在新的API中已经荡然无存。这就是上面提到的，为什么只有在mapred中才有Jobconf的原因。
>
> 6. 输出文件的命名也略有不同，map的输出命名为part-m-nnnnn，而reduce的输出命名为part-r-nnnnn，这里nnnnn指的是从0开始的部分编号。
>
> 这样了解了二者的区别就可以通过程序的引用包来判别新旧API编写的程序了。小菜建议最好用新的API编写hadoop程序，以防旧的API被抛弃！！！

原来是新旧API接口间有区别，mapred代表hadoop旧API，mapreduce代表新API，新旧API间不兼容也是很正常的。之前一直没注意import的包到底是啥，导致没有区分出这两个包的区别、进而后续遇到问题。解决方法如下，引用mapreduce包中的ChainMapper即可（原理上把所有类和API都换成mapred包里的也行，但是有新的为啥还要用旧的呢，而且旧的那一套和我一直学习的还不一样）：

```
import org.apache.hadoop.mapreduce.lib.chain.ChainMapper;
```

除此之外，由于此次作业较为简单，也没有遇到什么其他大的困难了。

四、参考资料

https://blog.csdn.net/u014470581/article/details/51488008

https://www.javazxz.com/thread-8435-1-1.html