

使用伪分布式来运行作业五

一、启动伪分布式下的Hadoop，并配置好HDFS

首先，进入到 `/usr/local/hadoop` 里面，输入 `./sbin/start-dfs.sh`，然后就会启动hadoop，输入 `jps`，检测是不是每一个部分都成功启动了。

```
hadoop@yawn-virtual-machine:/usr/local/hadoop$ ./sbin/start-dfs.sh
Starting namenodes on [localhost]
Starting datanodes
Starting secondary namenodes [yawn-virtual-machine]
hadoop@yawn-virtual-machine:/usr/local/hadoop$ jps
13461 Jps
13320 SecondaryNameNode
13130 DataNode
13006 NameNode
```

然后检查hdfs是不是正常在工作，如果能输出根目录下的内容，说明正常工作。输入：

```
hdfs dfs -ls /
```

但这个时候我看到的报错是：

```
hadoop@yawn-virtual-machine:/usr/local/hadoop$ hdfs dfs -ls /
Command 'hdfs' not found, did you mean:
  command 'hdfsls' from deb hdf4-tools (4.2.15-4)
  command 'hfs' from deb hfsutils-tcltk (3.2.6-15build2)
Try: sudo apt install <deb name>
```

说明无法找到Hadoop的hdfs指令。后来排查发现是我的Hadoop环境变量没有正确配置

于是我找到Hadoop的安装路径 `/usr/local/hadoop`，准备修改 `.bashrc` 文件。输入

```
nano ~/.bashrc`
```

在文件的末尾添加以下内容（因为我的Hadoop安装路径是 `/usr/local/hadoop`）

```
#Hadoop environment variables

export HADOOP_HOME=/usr/local/hadoop
export PATH=$PATH:$HADOOP_HOME/bin
```

重新加载 `.bashrc` 文件，编辑结束之后退出bashrc文件，使用以下命令使修改生效：

```
source ~/.bashrc
```

现在再验证 `hdfs dfs -ls /`，因为现在根目录下没有东西，所以没有输出，此时正确。可以继续往后。

二、上传csv文件和txt文件

首先，需要使用 `hdfs dfs -put` 命令将本地文件上传到HDFS中，具体是 `/user/hadoop/input` 目录中。

```
hdfs dfs -mkdir -p /user/hadoop/input #该目录不存在，所以这一步是在创建
```

```
hdfs dfs -put /本地绝对路径/analyst_ratings.csv /user/hadoop/input/
```

注意：本地绝对路径是指点开文档属性展示的路径，而不是从Downloads里直接开始的路径。要写完整。

```
hadoop@yawn-virtual-machine:/usr/local/hadoop$ hdfs dfs -put /Downloads/analyst_ratings.csv /user/hadoop/input/
put: `/Downloads/analyst_ratings.csv': No such file or directory
hadoop@yawn-virtual-machine:/usr/local/hadoop$ hdfs dfs -ls /user/hadoop/input
hadoop@yawn-virtual-machine:/usr/local/hadoop$ hdfs dfs -put /home/hadoop/Downloads/analyst_ratings.csv /user/hadoop/input/
hadoop@yawn-virtual-machine:/usr/local/hadoop$ hdfs dfs -ls /user/hadoop/input
Found 1 items
-rw-r--r--  1 hadoop supergroup  52462980 2024-10-23 00:37 /user/hadoop/input/analyst_ratings.csv
hadoop@yawn-virtual-machine:/usr/local/hadoop$ hdfs dfs -put /home/hadoop/Downloads/stop-word-list.txt /user/hadoop/input/
hadoop@yawn-virtual-machine:/usr/local/hadoop$ hdfs dfs -ls /user/hadoop/input
Found 2 items
-rw-r--r--  1 hadoop supergroup  52462980 2024-10-23 00:37 /user/hadoop/input/analyst_ratings.csv
-rw-r--r--  1 hadoop supergroup    2231 2024-10-23 00:38 /user/hadoop/input/stop-word-list.txt
hadoop@yawn-virtual-machine:/usr/local/hadoop$
```

上传成功之后，可以查看HDFS现在里面存储的文件：

```
hdfs dfs -ls /user/hadoop/input
```

三、设计思路

(一) StockCount

作业要求：在HDFS上加载上市公司热点新闻标题数据集（analyst_ratings.csv），统计数据集上市公司股票代码（“stock”列）的出现次数，按出现次数从大到小输出，输出格式为“<排名>: <股票代码>, <次数>”。

Mapper代码：

定义这部分是为了实现股票代码的计数，Mapper将从csv文件中读取每一行，提取“stock”列中的股票代码并且输出股票代码和1。

```
public static class StockMapper extends Mapper<Object, Text, Text, IntWritable> {
    private final static IntWritable one = new IntWritable(1);
    private Text stockCode = new Text()

    • public void map(Object key, Text value, Context context) throws IOException,
      InterruptedException {
    • // 按逗号分隔行数据
    • String[] fields = value.toString().split(",");
    • // 确保有至少4列
    • if (fields.length == 4) {
    •     String stock = fields[3].trim(); // 第四列为股票代码
    •     if (!stock.isEmpty()) {
    •         stockCode.set(stock);
    •         context.write(stockCode, one); // 输出（股票代码，1）
    •     }
    • }
```

```

    •         }
    •     }
    • }
    • }

```

Reducer代码:

Reducer负责接收Mapper的输出，统计每个股票代码的出现次数。

此处展示部分代码

```

public static class StockReducer extends Reducer<Text, IntWritable, Text, Text> {
    private List<StockCountPair> stockCounts = new ArrayList<>();

    • // 内部类用于存储股票代码和计数
    • public static class StockCountPair {
    •     private String stock;
    •     private int count;

    •     public StockCountPair(String stock, int count) {
    •         this.stock = stock;
    •         this.count = count;
    •     }

    •     public String getStock() {
    •         return stock;
    •     }

    •     public int getCount() {
    •         return count;
    •     }
    • }

```

Driver类:

```

public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "stock count");
    job.setJarByClass(StockCount.class);
    job.setMapperClass(StockMapper.class);
    job.setReducerClass(StockReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0])); // 输入文件路径
    FileOutputFormat.setOutputPath(job, new Path(args[1])); // 输出文件路径
    System.exit(job.waitForCompletion(true) ? 0 : 1); // 提交任务
}

```

main函数部分:

完成整个任务的配置，配置好mapper和reducer以及driver，同时指定之后运行需要的输入文件路径，实现启动整个文件的运行。

(二) Word Frequency

作业要求：在HDFS上加载上市公司热点新闻标题数据集（analyst_ratings.csv）和停词表（stop-word-list.txt），统计数据集热点新闻标题（“headline”列）中出现的前100个高频单词，按出现次数从大到小输出。要求忽略大小写，忽略标点符号，忽略停词（stop-word-list.txt）。输出格式为“<排名>: <单词>, <次数>”。

Mapper代码：

加载停词表stop-word-list.txt，逐行读取csv文件的数据，用split进行切片，提取headline部分，忽略标点符号，忽略大小写，按照空格分割单词。经过这样的处理之后，查看每一个单词是否是处在stop-word-list.txt里的，如果不是，则将这个单词和1按照键值对的方式输出，<word,1>，表示这个单词出现了一次，便于之后的reduce进行合并处理。

```
// 加载停词表
protected void setup(Context context) throws IOException,
InterruptedException {
    Configuration conf = context.getConfiguration();
    Path stopwordsPath = new Path(conf.get("stopwords.path")); // 停词文件路径
    FileSystem fs = FileSystem.get(conf); // 使用Hadoop FileSystem API来读取
HDFS文件

    • FSDataInputStream in = fs.open(stopwordsPath); // 打开文件
    • BufferedReader br = new BufferedReader(new InputStreamReader(in)); // 使用
BufferedReader包裹流
    • String stopword;
    • while ((stopword = br.readLine()) != null) {
    •     stopwords.add(stopword.trim().toLowerCase());
    • }
    • br.close();
    • in.close();
}

    • public void map(Object key, Text value, Context context) throws IOException,
InterruptedException {
    •     String[] fields = value.toString().split(",");
    •     if (fields.length == 4) {
    •         String headline = fields[1]; // 第二列为新闻标题
    •         // 去除标点，转换为小写，按空格分词
    •         String[] words = headline.replaceAll("[^a-zA-Z ]",
"").toLowerCase().split("\\s+");

    •         for (String wordStr : words) {
    •             // 忽略停词和空词
    •             if (!stopwords.contains(wordStr) && !wordStr.isEmpty()) {
    •                 word.set(wordStr);
    •                 context.write(word, one); // 输出 (单词, 1)
    •             }
    •         }
    •     }
}
}
```

Reducer代码:

接收Mapper先前的输出，作为自己的输入。将所有键相同的项进行合并，并且计算出总数，存储结果。按照要求的格式输出。即“<排名>: <单词>, <次数>”。

```
public static class WordReducer extends Reducer<Text, IntWritable, Text, Text> {
    private Map<String, Integer> wordCountMap = new HashMap<>();

    • public void reduce(Text key, Iterable<IntWritable> values, Context context)
    throws IOException, InterruptedException {
    •     int sum = 0;
    •     // 累加每个单词的出现次数
    •     for (IntWritable val : values) {
    •         sum += val.get();
    •     }
    •     // 将结果存入Map中
    •     wordCountMap.put(key.toString(), sum);
    • }

    • @Override
    • protected void cleanup(Context context) throws IOException,
    InterruptedException {
    •     // 将Map转换为List并按出现次数进行降序排序
    •     List<Map.Entry<String, Integer>> sortedList = new ArrayList<>
    (wordCountMap.entrySet());
    •     Collections.sort(sortedList, new Comparator<Map.Entry<String, Integer>>
    () {
    •         public int compare(Map.Entry<String, Integer> o1, Map.Entry<String,
    Integer> o2) {
    •             return o2.getValue().compareTo(o1.getValue()); // 降序排序
    •         }
    •     });

    •     // 只输出前100个高频单词
    •     int rank = 1;
    •     for (Map.Entry<String, Integer> entry : sortedList) {
    •         if (rank > 100) break;
    •         String outputValue = String.format("%d: %s, %d", rank,
    entry.getKey(), entry.getValue());
    •         context.write(new Text(), new Text(outputValue)); // 输出格式为 "<排名
    >: <单词>, <次数>"
    •         rank++;
    •     }
    • }
}
```

Driver类:

启动程序的，main函数的部分:

```
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    conf.set("stopwords.path", args[2]); // 停用词表文件路径

    • Job job = Job.getInstance(conf, "word frequency count");
```

```

• job.setJarByClass(WordFrequency.class);
• job.setMapperClass(WordMapper.class);
• job.setReducerClass(WordReducer.class);
• job.setOutputKeyClass(Text.class);
• job.setOutputValueClass(IntWritable.class);
• FileInputFormat.addInputPath(job, new Path(args[0])); // 输入文件路径
• FileOutputFormat.setOutputPath(job, new Path(args[1])); // 输出文件路径
• System.exit(job.waitForCompletion(true) ? 0 : 1); // 提交任务
}

```

四、运行过程以及运行结果

在 /home 里创建 workspace 文件夹，进入 workspace 再新建一个 hw5 的文件夹，在这个里面建立三个文件夹，分别是 build，lib 和 src。build 用来存放从 java 文件生成成为 jar 文件时产生的 .class 文件，lib 是存储可能的依赖文件，src 是存储 java 源文件。

1、编译Java文件（此时不必运行Hadoop），生成jar文件

首先进入存放Java文件的src文件夹里，打开终端。

首先对src文件夹里的所有Java文件进行编译：

```

hadoop@yawn-virtual-machine:~/workspace/hw5/src$ javac -classpath $(hadoop
classpath) -d ../build *.java

```

然后进入build文件夹

```

hadoop@yawn-virtual-machine:~/workspace/hw5/src$ cd ../build

```

将Java文件转换成jar文件，让其可以在Hadoop上运行

```

hadoop@yawn-virtual-machine:~/workspace/hw5/build$ jar -cvf ../stock-count.jar
*.class

```

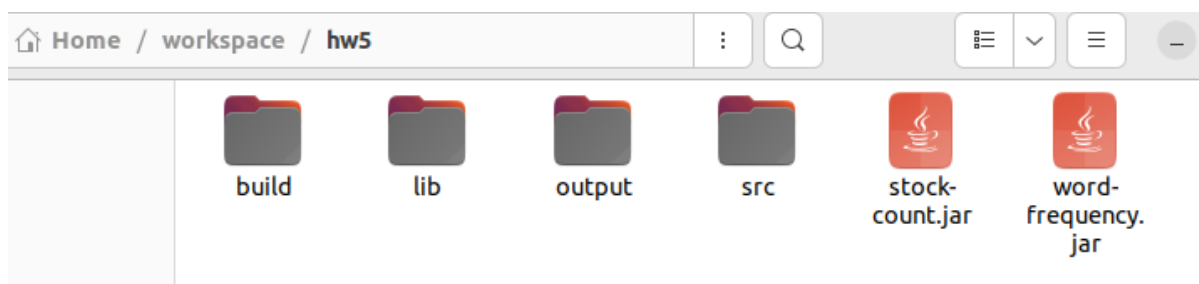
然后会出现一些配置过程的信息：

```

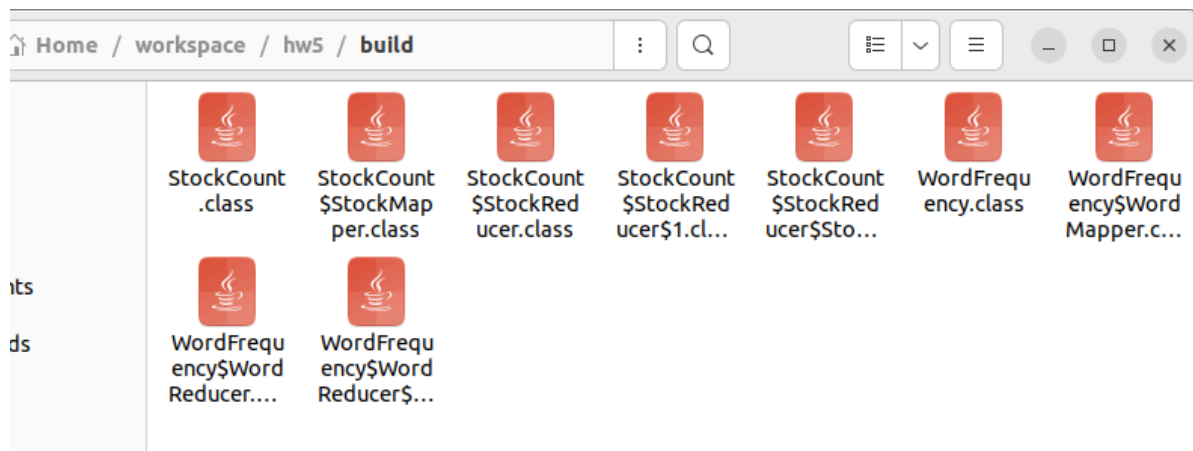
added manifest
adding: StockCount$StockMapper.class(in = 1810) (out= 769)(deflated 57%)
adding: StockCount$StockReducer$1.class(in = 1077) (out= 508)(deflated 52%)
adding: StockCount$StockReducer$StockCountPair.class(in = 599) (out= 353)
(deflated 41%)
adding: StockCount$StockReducer.class(in = 2898) (out= 1188)(deflated 59%)
adding: StockCount.class(in = 1447) (out= 785)(deflated 45%)

```

配置完成之后，会看到hw5这个文件夹下出现了两个打包好的jar文件：



进入build文件夹会发现同时生成的class文件若干：



2、打开并运行伪分布式下的Hadoop

```
hadoop@yawn-virtual-machine:~$ cd /usr/local/hadoop
hadoop@yawn-virtual-machine:/usr/local/hadoop$ ./sbin/start-dfs.sh
Starting namenodes on [localhost]
Starting datanodes
Starting secondary namenodes [yawn-virtual-machine]
```

用 jps 指令查看是否正常运行：

```
hadoop@yawn-virtual-machine:/usr/local/hadoop$ jps
22918 DataNode
23257 Jps
22793 NameNode
23130 SecondaryNameNode
```

出现这几项说明Hadoop正常启动了。可以继续后面的步骤。

3、在Hadoop上运行任务

(1) 运行任务一：stock-count.jar

a) 运行

`/home/hadoop/workspace/hw5/stock-count.jar`：将要运行的jar文件所在的绝对路径；

`StockCount`：stock-count.jar文件里的main的类名；

`/user/hadoop/input`：需要的输入的路径；

`/user/hadoop/output`：输出存储的位置。

```
hadoop@yawn-virtual-machine:/usr/local/hadoop$ hadoop jar
/home/hadoop/workspace/hw5/stock-count.jar StockCount /user/hadoop/input
/user/hadoop/output
```

```
2024-10-23 18:12:40,841 INFO impl.MetricsConfig: Loaded properties from hadoop-  
metrics2.properties  
2024-10-23 18:12:40,982 INFO impl.MetricsSystemImpl: Scheduled Metric snapshot  
period at 10 second(s).  
2024-10-23 18:12:40,982 INFO impl.MetricsSystemImpl: JobTracker metrics system  
started  
2024-10-23 18:12:41,145 WARN mapreduce.JobResourceUploader: Hadoop command-line  
option parsing not performed. Implement the Tool interface and execute your  
application with ToolRunner to remedy this.  
.....【只截取了部分运行过程】
```

b) 查看运行结果

```
hadoop@yawn-virtual-machine:/usr/local/hadoop$ hdfs dfs -ls /user/hadoop/output
```

```
Found 2 items  
-rw-r--r--  1 hadoop supergroup          0 2024-10-23 18:12  
/user/hadoop/output/_SUCCESS  
-rw-r--r--  1 hadoop supergroup    86831 2024-10-23 18:12  
/user/hadoop/output/part-r-00000
```

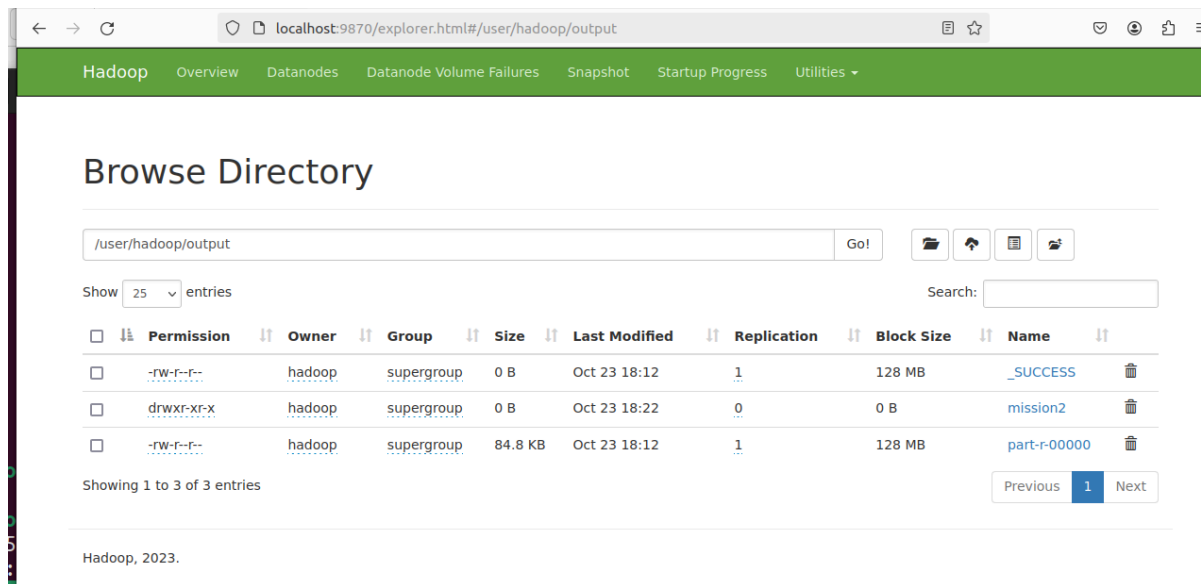
可以看到现在生成了一个part-r-00000的文件，这个就是存在output的输出。

也可以在命令行里查看内容，使用 `cat` 指令：

```
hadoop@yawn-virtual-machine:/usr/local/hadoop$ hdfs dfs -cat  
/user/hadoop/output/part-r-00000  
1: MS, 726  
2: MRK, 704  
3: QQQ, 693  
4: BABA, 689  
5: EWU, 681  
6: GILD, 663  
7: JNJ, 663  
8: MU, 659  
9: NVDA, 655  
10: VZ, 648  
11: KO, 643  
12: QCOM, 636  
13: M, 635  
14: NFLX, 635  
15: EBAY, 621  
.....  
5719: WHLRP, 1  
5720: WIP, 1  
5721: WPS, 1  
5722: XGTIW, 1  
5723: XOVR, 1  
5724: YAO, 1  
5725: YMLI, 1  
5726: YYY, 1  
5727: ZFC, 1  
5728: stock, 1
```


c) 在网页可视化地查看文件储存结果

在浏览器输入<http://localhost:9870>，点击页面上方的导航栏的“utilities”，点击“browse the file system”来查看hdfs文件系统中的文件和目录结构。



d) 将运行结果文件拉取到本地文件夹里

在hw5里新建一个output，将运行结果拉取至此处：

```
hadoop@yawn-virtual-machine:/usr/local/hadoop$ hdfs dfs -get  
/user/hadoop/output/part-r-00000 /home/hadoop/workspace/hw5/output
```

(2) 运行任务二：word-frequency.jar

a) 运行

```
hadoop@yawn-virtual-machine:/usr/local/hadoop$ hadoop jar  
/home/hadoop/workspace/hw5/word-frequency.jar wordFrequency /user/hadoop/input  
/user/hadoop/output
```

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 2  
    at wordFrequency.main(wordFrequency.java:96)  
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)  
    at  
sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)  
    at  
sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)  
    at java.lang.reflect.Method.invoke(Method.java:498)  
    at org.apache.hadoop.util.RunJar.run(RunJar.java:328)  
    at org.apache.hadoop.util.RunJar.main(RunJar.java:241)
```

Note

此时发现有报错，因为我还是像第一个任务那样直接选的input，这时因为需要用到input里的两个文件，所以要显性地指出。

经修改之后就可以正常运行了：

```
hadoop@yawn-virtual-machine:/usr/local/hadoop$ hadoop jar
/home/hadoop/workspace/hw5/word-frequency.jar wordFrequency
/user/hadoop/input/analyst_ratings.csv /user/hadoop/output/mission2
/user/hadoop/input/stop-word-list.txt
```

b) 查看运行结果

```
hadoop@yawn-virtual-machine:/usr/local/hadoop$ hdfs dfs -ls
/user/hadoop/output/mission2
Found 2 items
-rw-r--r--  1 hadoop supergroup          0 2024-10-23 18:22
/user/hadoop/output/mission2/_SUCCESS
-rw-r--r--  1 hadoop supergroup    1799 2024-10-23 18:22
/user/hadoop/output/mission2/part-r-00000
```

然后使用 `cat` 指令查看：

```
hadoop@yawn-virtual-machine:/usr/local/hadoop$ hdfs dfs -cat
/user/hadoop/output/mission2/part-r-00000
1: stocks, 37669
2: shares, 26843
3: q, 25950
4: vs, 24905
5: m, 24538
6: update, 23804
7: market, 22458
8: est, 20181
9: reports, 19300
10: eps, 18050
.....
96: wednesdays, 3163
97: thursdays, 3096
98: coverage, 3077
99: amid, 3049
100: ceo, 3007
```

c) 在网页可视化地查看文件储存结果





localhost:9870/explorer.html#/user/hadoop/output/mission2

Hadoop Overview Datanodes Datanode Volume Failures Snapshot Startup Progress Utilities

Browse Directory



/user/hadoop/output/mission2

Go!



Show 25 entries

Search:

<input type="checkbox"/>	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name	<input type="checkbox"/>
<input type="checkbox"/>	-rw-r--r--	hadoop	supergroup	0 B	Oct 23 18:22	1	128 MB	_SUCCESS	
<input type="checkbox"/>	-rw-r--r--	hadoop	supergroup	1.76 KB	Oct 23 18:22	1	128 MB	part-r-00000	

Showing 1 to 2 of 2 entries

Previous 1 Next

Hadoop, 2023.

d) 将运行结果文件拉取到本地文件夹里

将运行结果拉取至hw5里的output:

```
hadoop@yawn-virtual-machine:/usr/local/hadoop$ hdfs dfs -get  
/user/hadoop/output/mission2/part-r-00000  
/home/hadoop/workspace/hw5/output/mission2
```

五、程序分析

(一) stockcount

不足:

1、大规模数据处理时的内存使用问题:

当前的代码在 Mapper 中处理每行 CSV 数据时, 直接使用 `String.split(",")` 方法来分割字段。如果文件中的某些行非常长或存在非常多的字段, 这种方式会对内存产生较大的压力, 特别是在处理超大数据集时, 容易出现性能瓶颈。

2、不必要的数据传输:

在 MapReduce 中, Mapper 产生的中间结果需要通过网络传输到 Reducer。在当前实现中, Mapper 会生成许多小的中间结果, 每次只发送股票代码和 1。对于非常大的数据集, 这会产生大量的小数据包, 增加网络开销。

优化:

1、优化 CSV 解析:

使用更高效的 CSV 解析库 (如 OpenCSV 或 Apache Commons CSV) 来替代 `String.split(",")`, 这将显著提高处理大型 CSV 文件时的性能和内存使用效率。

2、Combiner 使用:

使用 Combiner 来减少 Mapper 产生的中间结果的数量。在目前的实现中, 所有 stock 计数都会发送 Reducer。如果在 Mapper 中使用 Combiner, 可以在每个 Mapper 本地先对同一股票代码的计数进行合并, 减少需要传输的数据量, 从而提升性能。

(二) Word Fequency

不足:

1、停词处理的效率:

停词表是通过每次 Mapper 初始化时加载的, 但并没有缓存或进行优化处理。如果停词表非常大, 可能会导致较高的初始化时间。此外, 每次对每个单词进行停词检查时, 直接在 Set 中查找, 但可以使用更高效的数据结构来提升查询速度。

优化：

1、停词表处理：

停词表可以在分布式缓存中进行管理，而不是每个 Mapper 独立加载。可以使用 Hadoop 的 DistributedCache 将停词表加载到内存中，让每个节点在本地访问停词表，从而提高处理效率。

```
DistributedCache.addCacheFile(new URI("/path/to/stopword-list.txt"), conf);
```