大数据技术基础实验二
实验报告

毛子恒
2019211397
北京邮电大学 计算机学院

日期：2022 年 3 月 19 日

1 概述

1.1 实验目的

1. 了解 IDEA 构建大数据工程的过程
2. 熟悉使用 Java 语言编写大数据程序；
3. 了解 MapReduce 的工作原理；
4. 掌握在集群上运行程序的方法。

1.2 实验步骤

1. 使用 IDEA 构建大数据工程；
2. 编写 WordCount 程序；
3. 程序打包和运行。

2 实验结果及分析

WordCount 程序 依照指导书编写 WordCount 程序。

`WordCount` 是实现通过MapReduce统计单词数量的类，其中包含两个子类 `TokenizerMapper` 和 `IntSumReducer`，分别实现了 Map 和 Reduce 过程。

`TokenizerMapper` 类如图 1所示，该类的 `map` 方法用于将输入转化为形如 `{word: 1}` 的键值对。

`IntSumReducer` 类如图 2所示，该类用于将具有相同键的各个值求和，统计出各个单词出现的次数，转化为形如 `{word: 2}` 的键值对。

`WordCount` 类的 `main` 方法如图 3所示，该方法是程序的入口，用于获取系统配置、输入输出路径、装载类以及驱动 job 进行。

程序打包 依照指导书的说明打包生成 jar 包，如图 4，去除其中的 MANIFEST.MF 文件，并且上传到服务器。

```java
public static class TokenizerMapper extends Mapper<Object, Text, Text, IntWritable> {
    private final static IntWritable one = new IntWritable( value: 1); // 定义常量1
    private Text word = new Text();

    public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
        StringTokenizer itr = new StringTokenizer(value.toString()); // 迭代每一个Token
        while (itr.hasMoreTokens()) {
            word.set(itr.nextToken()); // 设置word为下一个Token
            context.write(word, one); // 写入键值对{word: 1}
        }
    }
}
```

图 **1:** TokenizerMapper 类

```java
public static class IntSumReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
    private IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {
        int sum = 0;
        for (IntWritable val : values) { // 迭代键值对{key: [values]}
            sum += val.get(); // 求[values]的和
        }
        result.set(sum);
        context.write(key, result); // 写入键值对{key: sum}
    }
}
```

图 **2:** IntSumReducer 类

```java
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration(); // 系统配置信息
    String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs(); // 运行参数
    if (otherArgs.length != 2) { // 运行参数必须恰好为2个
        System.err.println("Usage: WordCount <in> <out>");
        System.exit( status: 2);
    }
    Job job = new Job(conf, jobName: "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class); // Mapper类
    job.setCombinerClass(IntSumReducer.class); // Combiner类
    job.setReducerClass(IntSumReducer.class); // Reducer类
    job.setOutputKeyClass(Text.class); // 输出的key类型
    job.setOutputValueClass(IntWritable.class); // 输出的value类型
    FileInputFormat.addInputPath(job, new Path(otherArgs[0])); // 从参数中构建输入文件
    FileOutputFormat.setOutputPath(job, new Path(otherArgs[1])); // 从参数中构建输出文件
    System.exit(job.waitForCompletion( verbose: true) ? 0 : 1);
}
```

图 **3:** main 方法

程序运行 首先创建输入文件 `2019211397-mzh-input.txt`，内容如图 **5**。

启动集群，通过如下命令创建文件夹并且将输入文件传入 HDFS 当中：

```
hadoop fs -mkdir -p /test
hadoop fs -put ./2019211397-mzh-input.txt /test/2019211397-mzh-input.txt
```

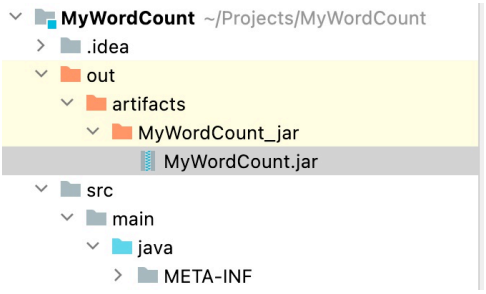

图 4: 程序打包

```
[root@mzh-2019211397-0001 ~]# hadoop fs -text /test/2019211397-mzh-input.txt
22/03/19 18:00:55 WARN util.NativeCodeLoader: Unable to load native-hadoop l
hello world
hello world
hello world
hadoop spark
hadoop spark
hadoop spark
hadoop spark
dog fish
dog fish
dog fish
dog fish
dog fish
```

图 5: 输入文件

在此处如果出现“There are 0 datanode(s) running and no node(s) are excluded in this operation.”的错误，那么应该检查 DataNode 是否已经启动，首先停止集群，检查四个主机的 core-site.xml 中的 `hadoop.tmp.dir` 项应该为`/home/modules/hadoop-2.7.7/tmp`，同时删除各个主机下的这个文件夹，并且注释 hosts 文件中 127.0.0.1 的有关项，最后重新启动集群。

运行命令和结果如图 6和图 7所示，可见利用打包好的 jar 包中的 WordCount 类，我们成功发起并完成了一个 mapreduce 的 job。

```
[root@mzh-2019211397-0001 ~]# hadoop jar ./MyWordCount.jar WordCount /test/2019211397-mzh-input.txt /test/2019211397-mzh-output
22/03/19 17:58:56 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
22/03/19 17:58:57 INFO client.RMProxy: Connecting to ResourceManager at mzh-2019211397-0001/192.168.0.168:8032
22/03/19 17:58:59 INFO input.FileInputFormat: Total input paths to process : 1
22/03/19 17:58:59 INFO mapreduce.JobSubmitter: number of splits:1
22/03/19 17:58:59 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1647683796427_0001
22/03/19 17:59:00 INFO impl.YarnClientImpl: Submitted application application_1647683796427_0001
22/03/19 17:59:00 INFO mapreduce.Job: The url to track the job: http://mzh-2019211397-0001:8088/proxy/application_1647683796427_0001/
22/03/19 17:59:00 INFO mapreduce.Job: Running job: job_1647683796427_0001
22/03/19 17:59:09 INFO mapreduce.Job: Job job_1647683796427_0001 running in uber mode : false
22/03/19 17:59:09 INFO mapreduce.Job:  map 0% reduce 0%
22/03/19 17:59:13 INFO mapreduce.Job:  map 100% reduce 0%
22/03/19 17:59:21 INFO mapreduce.Job:  map 100% reduce 100%
```

图 6: 运行命令

最后得到的统计结果如图 8所示。

3 实验总结

本次实验中我编写代码进行了简单的 MapReduce 操作，使我的 Java 代码能力得到了增强，对 HDFS 和 MapReduce 的原理理解更加深刻。

```
22/03/19 17:59:22 INFO mapreduce.Job: Job job_1647683796427_0001 completed successfully
22/03/19 17:59:22 INFO mapreduce.Job: Counters: 49
        File System Counters
                FILE: Number of bytes read=76
                FILE: Number of bytes written=251083
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=259
                HDFS: Number of bytes written=46
                HDFS: Number of read operations=6
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
        Job Counters
                Launched map tasks=1
                Launched reduce tasks=1
                Data-local map tasks=1
                Total time spent by all maps in occupied slots (ms)=2328
                Total time spent by all reduces in occupied slots (ms)=5317
                Total time spent by all map tasks (ms)=2328
                Total time spent by all reduce tasks (ms)=5317
                Total vcore-milliseconds taken by all map tasks=2328
                Total vcore-milliseconds taken by all reduce tasks=5317
                Total megabyte-milliseconds taken by all map tasks=2383872
                Total megabyte-milliseconds taken by all reduce tasks=5444608
        Map-Reduce Framework
                Map input records=12
                Map output records=24
                Map output bytes=229
                Map output materialized bytes=76
                Input split bytes=126
                Combine input records=24
                Combine output records=6
                Reduce input groups=6
                Reduce shuffle bytes=76
                Reduce input records=6
                Reduce output records=6
                Spilled Records=12
                Shuffled Maps =1
                Failed Shuffles=0
                Merged Map outputs=1
                GC time elapsed (ms)=117
                CPU time spent (ms)=890
                Physical memory (bytes) snapshot=350224384
                Virtual memory (bytes) snapshot=2585395200
                Total committed heap usage (bytes)=173015040
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=133
        File Output Format Counters
                Bytes Written=46
```

图 **7:** 运行结果

```
[root@mzh-2019211397-0001 ~]# hadoop fs -text /test/2019211397-mzh-output/part-r-00000
22/03/19 18:00:13 WARN util.NativeCodeLoader: Unable to load native-hadoop library for
dog     5
fish    5
hadoop  4
hello   3
spark   4
world   3
```

图 **8:** 输出文件