实 验 报 告 内 容

一・实验内容

实验题目：MapReduce 编程实践

实验内容与要求：

1. 使用 MapReduce 实现多个文本文件中 `WordCount` 词频统计功能，实验编写 Map 处理逻辑、编写 Reduce 处理逻辑、编写 `main` 方法。

二・实验目的

1. 通过实验掌握基本的 MapReduce 编程方法。
2. 实现统计 HDFS 系统中多个文本文件中的单词出现频率。

三・实验过程截图及说明

1. 实验环境与前提

本实验在 Hadoop 伪分布式环境下进行，使用 `lab-client` 容器完成代码编写、编译打包与任务提交。输入数据存放于 HDFS 目录 `/input`，输出数据写入 HDFS 目录 `/output`。

进入容器命令如下：

```
docker exec -it lab-client bash
```

2. 生成数据并上传到 HDFS

在 `/workspace` 目录下准备 3 个文本文件，并上传到 HDFS 输入目录 `/input`：

```
cd /workspace

echo "Hello Hadoop Hello MapReduce" > file1.txt
echo "MapReduce is cool and Hadoop is powerful" > file2.txt
echo "CachyOS runs Docker very fast" > file3.txt

hdfs dfs -mkdir -p /input
hdfs dfs -put file1.txt file2.txt file3.txt /input/
hdfs dfs -ls /input/
```

3. 编写 `WordCount` 程序

程序由三部分组成：

1. `TokenizerMapper`：对输入行进行分词，并输出 `<word, 1>`。
2. `IntSumReducer`：对相同 `word` 的计数进行累加求和。
3. `main`：配置 `Job`，指定输入输出路径，提交任务并等待完成。

`WordCount.java` 如下：

```
import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCount {

    public static class TokenizerMapper extends Mapper<Object, Text, Text,
IntWritable>{

        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();

        public void map(Object key, Text value, Context context) throws IOException,
InterruptedException {
            StringTokenizer itr = new StringTokenizer(value.toString());
            while (itr.hasMoreTokens()) {
                word.set(itr.nextToken());
                context.write(word, one);
            }
        }
    }

    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) {
                sum += val.get();
            }
            result.set(sum);
            context.write(key, result);
        }
    }

    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "word count");
```

```
        job.setJarByClass(WordCount.class);
        job.setMapperClass(TokenizerMapper.class);
        job.setCombinerClass(IntSumReducer.class);
        job.setReducerClass(IntSumReducer.class);

        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

4. 编译、打包与运行

在容器终端中执行下列命令编译并打包为 `wc.jar`：

```
rm -f *.class
javac -cp `hadoop classpath` WordCount.java
jar cf wc.jar WordCount*.class
```

运行任务（注意：输出目录必须不存在，否则任务会失败）：

```
hadoop jar wc.jar WordCount /input /output
```

5. 查看输出结果

运行完成后，查看 HDFS 输出目录并读取结果文件：

```
hdfs dfs -ls /output
hdfs dfs -cat /output/part-r-00000
```

本次实验的词频统计输出如下：

```
CachyOS 1
Docker 1
Hadoop 2
Hello 2
MapReduce 2
and 1
cool 1
fast 1
is 2
powerful 1
runs 1
very 1
```

图 1 展示了 `WordCount` 代码、任务运行后的计数器摘要以及输出结果文件内容，证明多文本输入的词频统计功能可用。

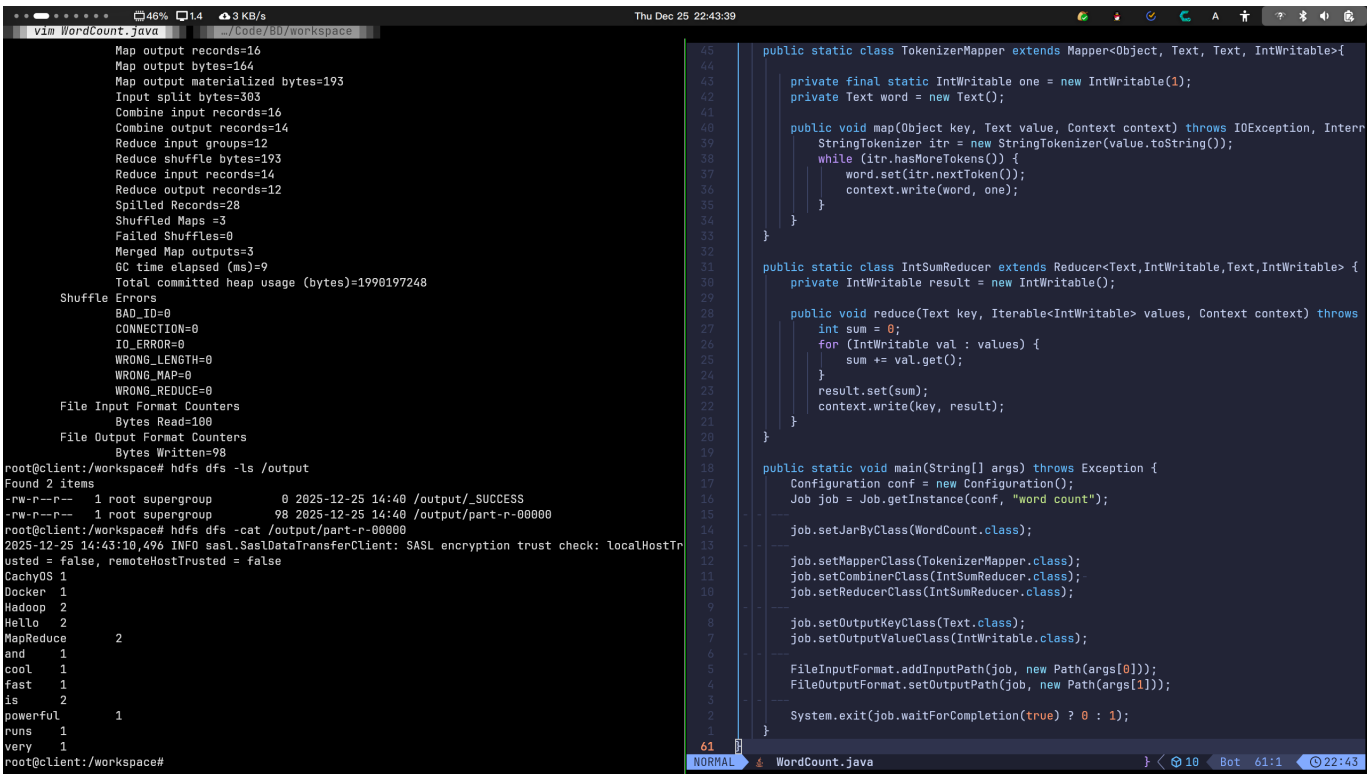

图 1 MapReduce WordCount：运行与输出结果

四、实验总结及心得体会

实验总结：本实验完成 `WordCount` 的 Map、Reduce 与 `main` 方法编写，并在 HDFS 上对多个输入文本文件进行词频统计，最终将结果写入 `/output/part-r-00000` 并验证输出正确，满足实验内容与要求。

心得体会：MapReduce 的核心在于将业务拆解为可并行的映射与可聚合的归约两步，`Combiner` 能在 Mapper 端进行局部聚合以减少网络与落盘开销。实际操作中需要注意输出目录必须不存在，建议在实验脚本中显式管理输出路径或在提交前检查目录状态，以避免重复运行时的失败。