实验四　MapReduce 2.x 编程

4.1 实验目的

- 学习编写简单的基于 Java API 的 MapReduce 程序
- 学习在单机集中式、单机伪分布、分布式部署方式下运行 MapReduce 程序

4.2 实验任务

- 完成基于 Java API 编写的“WordCount”的 MapReduce 程序
- 在单机集中式、单机伪分布式、分布式部署方式下调试、提交并运行该应用程序

4.3 实验环境

- 操作系统：Ubuntu 18.04
- JDK 版本：1.8
- Hadoop 版本：2.10.1
- IDEA 版本：2020.2.3 （Ultimate 版）

4.4 实验步骤

4.4.1 编写 MapReduce 应用程序

(1) 新建 Maven 项目并添加依赖

- 新建名为“MapReduceDemo”的 Maven 项目
- 编辑项目根目录下的 pom.xml 文件，在 <dependencies> 标签中添加 Hadoop 相关的依赖包 hadoop-common，hadoop-client，hadoop-hdfs

```
<dependencies>
    <!-- https://mvnrepository.com/artifact/org.apache.hadoop/hadoop-common
        -->
    <dependency>
        <groupId>org.apache.hadoop</groupId>
        <artifactId>hadoop-common</artifactId>
        <version>2.10.1</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/org.apache.hadoop/hadoop-client
```

```
        -->
    <dependency>
        <groupId>org.apache.hadoop</groupId>
        <artifactId>hadoop-client</artifactId>
        <version>2.10.1</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/org.apache.hadoop/hadoop-hdfs -->
    <dependency>
        <groupId>org.apache.hadoop</groupId>
        <artifactId>hadoop-hdfs</artifactId>
        <version>2.10.1</version>
    </dependency>
</dependencies>
```

修改完成后，在菜单界面选择 View->Tool Windows->Maven，在弹出的界面中点击 Reload All Maven Projects 加载依赖文件，第一次加载此过程可能耗时较长。

(2) 编写 Java 程序

编写一个 Java 应用程序，实现对给定文本中的单词进行计数。

- 在项目的 *src/main/java* 目录下，选择 New->Package，输入名称 *cn.edu.ecnu.mapreduce.example.java.wordcount*
- 右键单击建好的包，选择 New->Java Class，输入名称 *WordCountMapper*

```
package cn.edu.ecnu.mapreduce.example.java.wordcount;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

import java.io.IOException;

/* 步骤1：确定输入键值对[K1,V1]的数据类型为[LongWritable,Text]，输出键值对
    [K2,V2]的数据类型为[Text,IntWritable] */
public class WordCountMapper extends Mapper<LongWritable, Text, Text,
    IntWritable> {

    @Override
    protected void map(LongWritable key, Text value, Context context)
            throws IOException, InterruptedException {
        /* 步骤2：编写处理逻辑将[K1,V1]转换为[K2,V2]并输出 */
        // 以空格作为分隔符拆分成单词
```

```
        String[] datas = value.toString().split(" ");
        for (String data : datas) {
            // 输出分词结果
            context.write(new Text(data), new IntWritable(1));
        }
    }
}
```

- 右键单击建好的包，选择 New->Java Class，输入名称 *WordCountReducer*

```
package cn.edu.ecnu.mapreduce.example.java.wordcount;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

import java.io.IOException;

/* 步骤1: 确定输入键值对[K2,List(V2)]的数据类型为[Text,
    IntWritable], 输出键值对[K3,V3]的数据类型为[Text,IntWritable] */
public class WordCountReducer extends Reducer<Text, IntWritable, Text,
    IntWritable> {
    @Override
    protected void reduce(Text key, Iterable<IntWritable> values, Context
        context)
            throws IOException, InterruptedException {
        /* 步骤2: 编写处理逻辑将[K2,List(V2)]转换为[K3,V3]并输出 */
        int sum = 0;
        // 遍历累加求和
        for (IntWritable value : values) {
            sum += value.get();
        }
        // 输出计数结果
        context.write(key, new IntWritable(sum));
    }
}
```

- 右键单击建好的包，选择 New->Java Class，输入名称 *WordCount*

```
package cn.edu.ecnu.mapreduce.example.java.wordcount;

import org.apache.hadoop.conf.Configured;
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.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

public class WordCount extends Configured implements Tool {

  @Override
  public int run(String[] args) throws Exception {
    /* 步骤1: 设置作业的信息 */
    Job job = Job.getInstance(getConf(), getClass().getSimpleName());
    // 设置程序的类名
    job.setJarByClass(getClass());

    // 设置数据的输入输出路径
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    // 设置map和reduce方法
    job.setMapperClass(WordCountMapper.class);
    job.setReducerClass(WordCountReducer.class);

    // 设置map方法的输出键值对数据类型
    job.setMapOutputKeyClass(Text.class);
    job.setMapOutputValueClass(IntWritable.class);
    // 设置reduce方法的输出键值对数据类型
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);

    return job.waitForCompletion(true) ? 0 : 1;
  }

  public static void main(String[] args) throws Exception {
    /* 步骤2: 运行作业 */
    int exitCode = ToolRunner.run(new WordCount(), args);
    System.exit(exitCode);
  }
}
```

4.4.2 调试 MapReduce 应用程序

使用 IDEA 调试 WordCount 应用程序[1]。

(1) 准备数据

使用之前准备的数据集 pd.train，数据集的位置为*/home/dase-local/input/*。

(2) 配置运行环境

点击菜单栏 Run -> Edit Configuration, 在弹出的界面中点击 + 号选择 Application，新建 Application 配置，Name 为 WordCount，配置界面如图4.1 所示。

a) 配置 Main Class 为 *cn.edu.ecnu.mapreduce.example.java.wordcount.WordCount*

b) 配置 Program arguments 为 */home/dase-local/input/ /home/dase-local/IdeaProjects/MapReduceDemo/output*

该程序需要传入两个参数，第一个参数为输入文件路径，即要统计单词数的文本文件或文件夹的路径，若输入的是文件夹的路径，输出的是文件夹中所有文本文件的单词数统计结果；第二个参数为输出文件路径，即存放统计结果的文件路径。输出文件路径无需在程序运行前创建好，程序运行过程中会生成该输出文件路径。

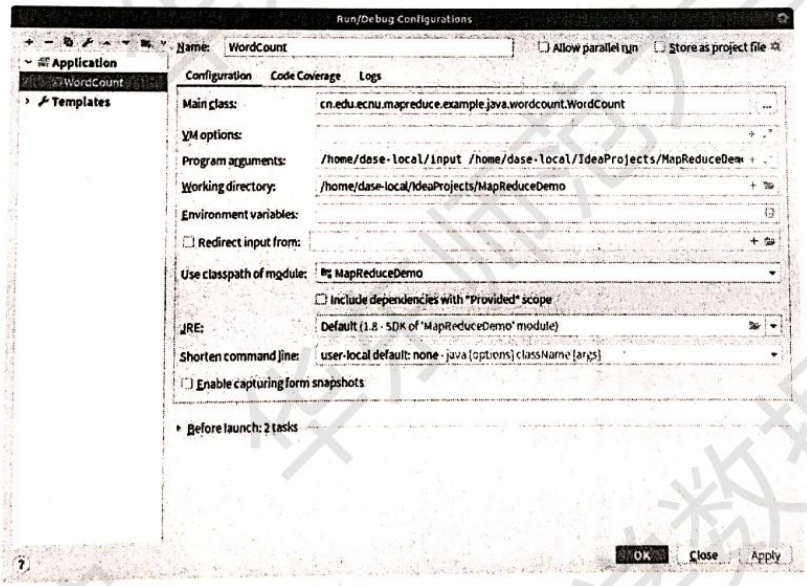

图 4.1 运行参数配置界面

(3) 运行应用程序

在菜单栏点击 Run->Run 'WordCount'，在 IDEA 中直接运行程序，待运行完毕后，项目根目录下会出现 "output" 文件夹，其中 "part-r-00000" 文件就是运行结果，如图4.2所示。

[1]若在 Windows 下调试 MapReduce 应用程序，则需按照附录 A先进行相关配置

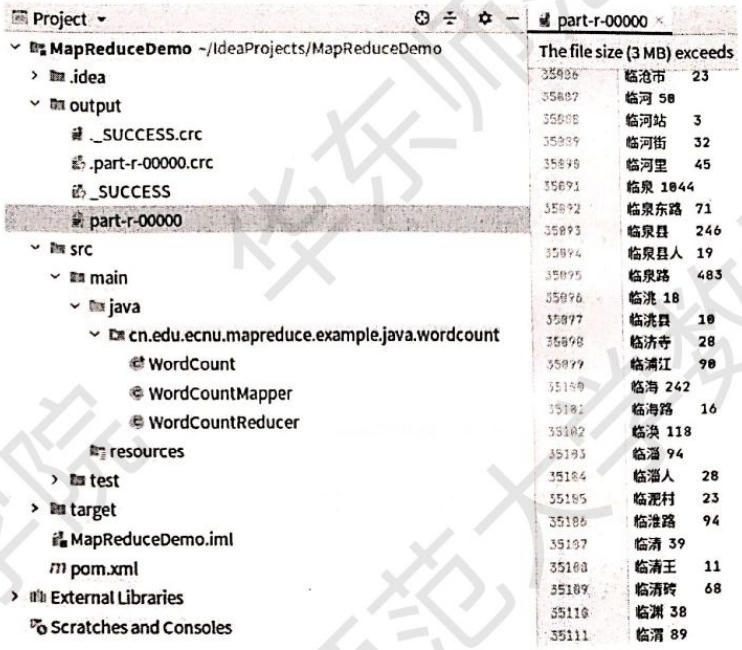

图 4.2 IDEA 中应用程序调试结果

4.4.3 运行 MapReduce 应用程序

在部署的 Hadoop 上运行 WordCount 应用程序。

(1) 准备工作

以下操作在各节点均以 dase-local 用户身份进行：

- 使用 IDEA 将项目打包成可执行 jar 包

  jar 包名称为 WordCount.jar，打包路径为*/home/dase-local/IdeaProjects/HDFSFileOP/out/artifacts/WordCount/*，配置界面如图4.3所示。

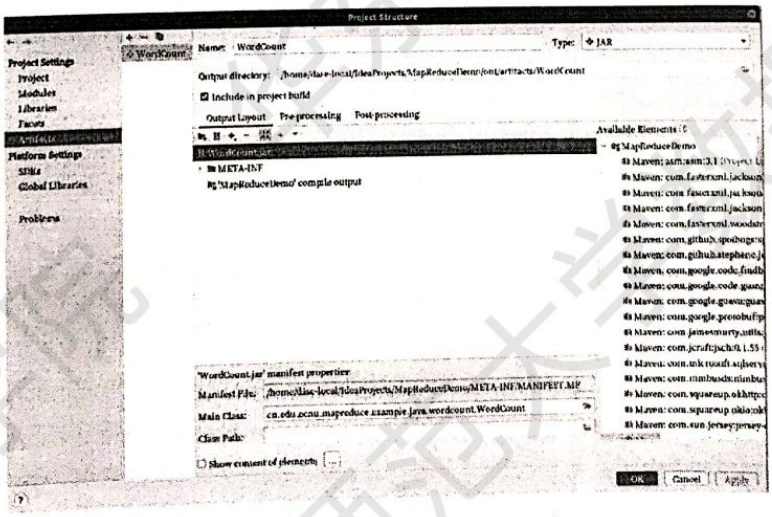

图 4.3 将项目打包为可执行 jar 包

• 复制 jar 包到指定目录

在客户端节点执行以下命令，将之前打好的包拷贝至*/home/dase-local/hadoop-2.10.1/myApp* 路径下。

```
mkdir ~/hadoop-2.10.1/myApp/ #在hadoop目录下新建myApp/目录
cp  ~/IdeaProjects/MapReduceDemo/out/artifacts/WordCount/WordCount.jar
    ~/hadoop-2.10.1/myApp/
    #将打包好的可执行jar包拷贝到hadoop安装路径下的myApp/目录下
```

(2) 单机伪分布式部署方式下运行应用程序

以下操作在各节点均以 dase-local 用户身份进行：

• 启动 Yarn 和 HDFS 服务

```
su dase-local
~/hadoop-2.10.1/sbin/start-yarn.sh
~/hadoop-2.10.1/sbin/mr-jobhistory-daemon.sh start historyserver
~/hadoop-2.10.1/sbin/start-dfs.sh
```

• 上传输入文件

输入文件 pd.train 在上一章中已上传至 *hdfs://localhost:9000/user/dase-local/input/*。

• 通过提交 jar 包运行应用程序

```
cd hadoop-2.10.1/
./bin/hadoop jar ./myApp/WordCount.jar input/pd.train output
    #使用hadoop命令在Yarn模式下运行jar包，并在运行时输入参数
```

该程序实现了统计 HDFS "*/user/dase-local/input/*" 目录下的 "README.txt" 文件单词数量的操作。运行结果存储在 HDFS 的 "*/user/dase-local/output/*" 目录下的 "part-r-00000" 文件中，如图4.4所示。

• 查看运行结果

```
cd hadoop-2.10.1/
./bin/hdfs dfs -tail output/part-r-00000 #查看输出结果
```

通过命令行查看的输出结果如图 4.5所示。

• 停止相关服务

```
~/hadoop-2.10.1/sbin/stop-yarn.sh
~/hadoop-2.10.1/sbin/mr-jobhistory-daemon.sh stop historyserver
~/hadoop-2.10.1/sbin/stop-dfs.sh
```

(3) 分布式部署方式下运行应用程序

图 4.4 单机伪分布式部署下应用程序运行结果存储位置

```
MBA   2
MH    26
MK2   68
MP3   39
MPV   12
N     390
NBA   30
P订制  21
PDA   38
PM    640
PMF   10
PP    20
PSA   23
Power      10
```

图 4.5 单机伪分布式部署下应用程序运行结果

以下操作在各节点均以 dase-dis 用户身份进行：

- 复制 jar 包到指定目录

  在客户端节点执行以下命令，将之前打好的包拷贝至 dase-dis 用户*/home/dase-dis/hadoop-2.10.1/myApp* 目录下。

```
su dase-dis
mkdir ~/hadoop-2.10.1/myApp/ #在hadoop目录下新建myApp/目录
scp dase-local@localhost:/home/dase-local/hadoop-2.10.1/myApp/ WordCount.jar
    /home/dase-dis/hadoop-2.10.1/myApp/
```

- 启动 Yarn 和 HDFS 服务

  在主节点执行以下命令：

```
su dase-dis
~/hadoop-2.10.1/sbin/start-yarn.sh
~/hadoop-2.10.1/sbin/mr-jobhistory-daemon.sh start historyserver
```

```
~/hadoop-2.10.1/sbin/start-dfs.sh
```

- 上传输入文件

  输入文件 pd.train 在上一章中已上传至 *hdfs://ecnu01:9000/user/dase-dis/input/*。

- 通过提交 jar 包运行应用程序

  在客户端节点执行以下命令：

```
cd ~/hadoop-2.10.1/
./bin/hadoop jar ./myApp/WordCount.jar input/pd.train output
./bin/hdfs dfs -tail output/part-r-00000 #查看输出结果
```

该程序实现了统计 HDFS "*/user/dase-dis/input/*" 目录下的 "README.txt" 文件单词数量的操作。运行结果存储在 HDFS 的 "*/user/dase-dis/output/*" 目录下的 "part-r-00000" 文件中。

- 停止服务

  在主节点执行以下命令：

```
~/hadoop-2.10.1/sbin/stop-yarn.sh
~/hadoop-2.10.1/sbin/mr-jobhistory-daemon.sh stop historyserver
~/hadoop-2.10.1/sbin/stop-dfs.sh
```

4.5 思考题

1 如何从 Hadoop JobHistory 的 Web UI 中查看某个 MapReduce 应用程序分别启动了多少个 Map 任务和 Reduce 任务？

2 如何从 Hadoop JobHistory 的 Web UI 中查看某个 MapReduce 应用程序启动的 Map 任务和 Reduce 任务分别是在哪些 NodeManager 上执行的？

3 对于一个 MapReduce 应用程序，是否存在某一时刻 Map 任务和 Reduce 任务同时运行？请结合 Hadoop JobHistory 的 Web UI 给出例证。

4 编写一个利用 Combine 机制实现的 WordCount 程序，使用实验二中的 pd.train 数据集，在分布式部署方式下分别运行本实验中的 WordCount 程序和使用 Combine 机制的 WordCount 程序，观察二者的时间差并分析原因。

附录

A Windows 下调试 MapReduce 应用程序

(1) 下载 winutils

在 Windows 下，Hadoop 需要一些 native 库来访问本地文件系统，否则无法正确运行 MapReduce 程序。因此，需要下载所需的 native 库，即 winutils[2]。将下载完成的 winutils 进行解压，并将解压文件夹下的 hadoop.dll、winutils.exe 等文件放置在 C:\Hadoop\bin 目录下。

(2) 配置环境变量

- 在“此电脑”图标上右键选择属性，然后选择“高级系统设置”，如图 4.6所示。

图 4.6 选择高级系统设置

- 在弹出的界面中选择“高级”并点击该界面的“环境变量”，如图 4.7所示。
- 点击系统变量下的“新建”按钮，并添加名为“HADOOP_HOME”的环境变量，如图 4.8所示。
- 点击系统变量中名为“PATH”的环境变量，并在其中添加 winutils 相关的路径，如图 4.9所示。

(3) 编程注意事项

- 环境变量配置完成之后需要重启 IDE。
- 请确保 winutils 版本与程序引入的 hadoop 依赖版本保持一致。

[2]https://github.com/steveloughran/winutils/releases/download/tag_2017-08-29-hadoop-2.8.1-native/hadoop-2.8.1.zip

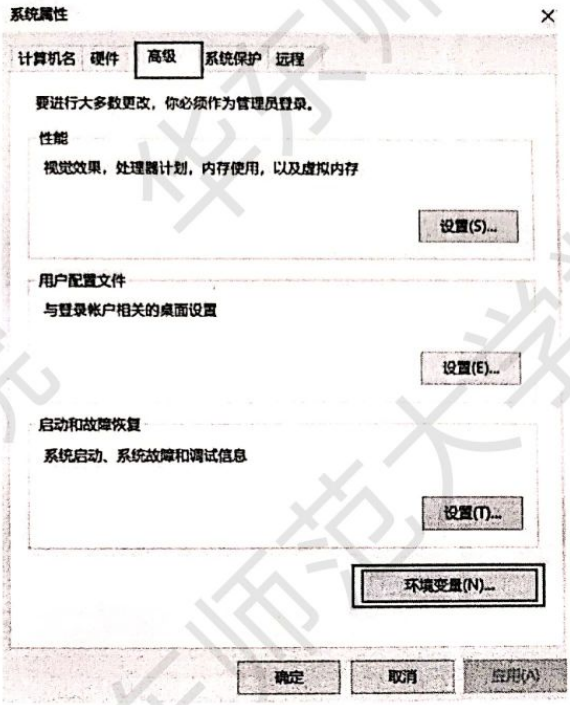

图 4.7 更改环境变量配置

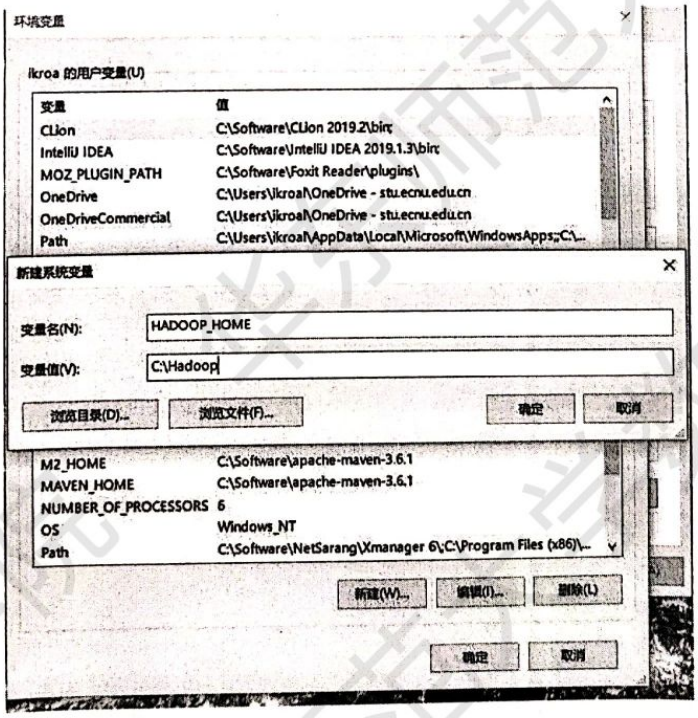

图 4.8 添加名为 HADOOP_HOME 的环境变量

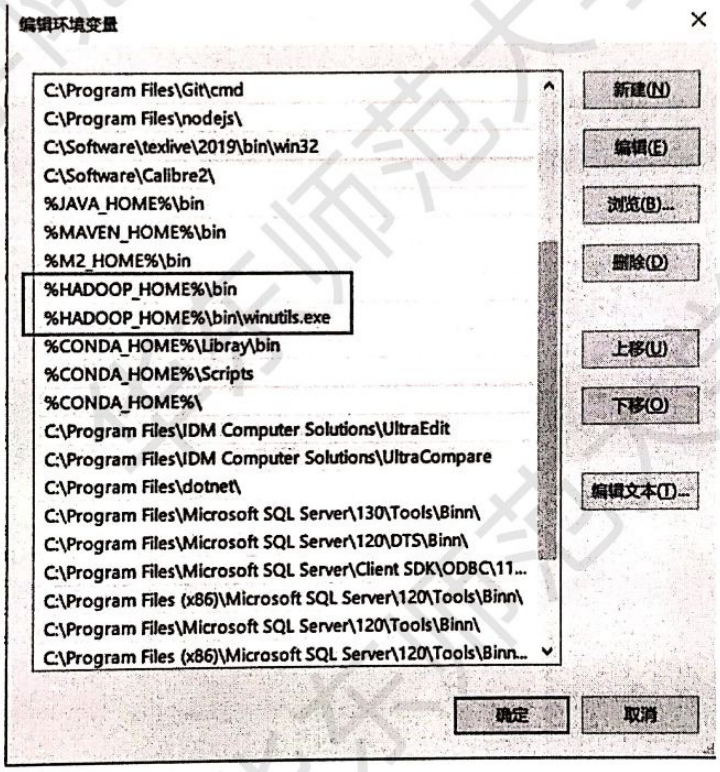

图 4.9 在 PATH 中添加 winutils 相关路径