深 圳 大 学 实 验 报 告

课程名称：智能网络与计算

实验项目名称：实验四 云-边协同计算实验

学院：计算机与软件学院

专业：计算机科学与技术

指导教师：车越岭

报告人：吴嘉楷 学号：2022150168 班级：国际班

实验时间：2024 年 11 月 20 日、12 月 4 日

实验报告提交时间：2024 年 12 月 3 日

教务处制

实验 4　云-边协同计算实验

实验目的与要求：

1. 了解 MapReduce 计算模型的原理；
2. 学会使用编程语言和工具实现 MapReduce 的 Map 和 Reduce 功能，并行实现特定任务的高效计算；

方法、步骤：

1. 阅读经典论文“MapReduce: Simplified data processing on large clusters，OSDI 2004”，学习掌握 MapReduce 计算框架的基本原理；
2. 以熟悉的编程语言（如 Java、Python 等）编程实现 MapReduce 功能，进而完成对给定文本文件的词频统计。针对给定文件（如 A.txt），统计里面文章中单词的出现频次并降序输出统计结果到文件 Sta_A.txt。Sta_A.txt 文件包含 A.txt 文件里面每个单词及其出现的频次。
   a) 数据集包含两个文件，1 个小文件约 20 MB 左右，1 个大文件约 1.5 GB 左右。
3. **探索：**除了编程实现 MapReduce 之外，也可以在电脑按照本地的 Apache Spark 计算框架（下载链接：https://spark.apache.org/downloads.html，Spark 安装参考链接：https://dblab.xmu.edu.cn/blog/4322/ ），并利用 Spark 来实现词频统计任务。

实验过程及内容：

（此处写详细的实验步骤、代码，需要有代码注释、实验截图、照片等证明材料）

1. 从 blackboard 上下载计算部分的课件，得到经典论文“MapReduce: Simplified data processing on large clusters，OSDI 2004”，并进行阅读和学习。

MapReduce 是一个为大规模数据集设计的编程模型，它允许并行处理和生成大型数据，它的核心思想是通过两个简单的函数 Map 和 Reduce 来处理数据。

MapReduce 的系统架构：

MapReduce 框架通常由一个主节点（Master）和多个工作节点（Workers）组成。主节点负责分配任务给工作节点，并跟踪任务的进度。工作节点执行分配给它们的 Map 或 Reduce 任务。此外，框架还提供了容错、负载平衡和并行化处理等特性。这些特性使得 MapReduce 非常适合在大型集群上处理大规模数据集。

MapReduce 的三个关键阶段：

（1）Map 阶段：

在这个阶段，输入数据被分割成多个块（chunks），每个块被分配给一个 Map 任务。Map 任务对每个输入的键值对（k1, v1）执行用户定义的 map 函数，产生中间结果的键值对（k2, v2）。这些中间结果通常会被写入临时存储。

（2）Shuffle 阶段：

在 Map 阶段完成后，系统会根据中间键（k2）对数据进行重新分布，确保所有具有相同键的数据都被发送到同一个 Reduce 任务。这个过程也称为分区（Partitioning）和排序（Sorting）。

（3）Reduce 阶段：

Reduce 任务接收来自 Map 任务的中间结果，对每个键（k2）及其关联的值列表（list(v2)）执行用户定义的 reduce 函数。Reduce 函数的输出通常是最终结果，这些结果会被写入文件系统。

MapReduce 使用键值对作为其核心数据结构。Map 函数的输出和 Reduce 函数的输入都是键值对，因为键（Key）允许 MapReduce 在多台机器之间分配和并行化负载。

2. 使用 java 语言编程实现 MapReduce 功能，进而完成对给定文本文件的词频统计。

（1）编写 java 代码

首先，在 IDEA 上创建 maven 项目，并命名为 WordCounter，包名定为 szu.mr：

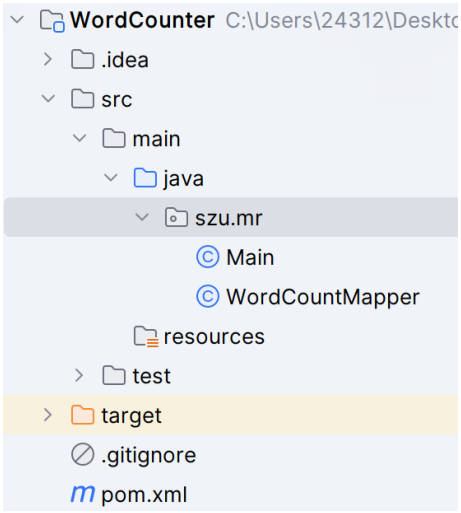

图 1 项目结构

然后，在 pom.xml 文件中添加 hadoop 依赖。在 maven 项目中添加 Hadoop 依赖是因为我们在使用 MapReduce 程序时，需要用到 hadoop 分布式框架，其中集成了 MapReduce 的相关类，便于我们继承后加以使用。

在添加 hadoop 依赖后，maven 会帮助我们自动下载添加的依赖 hadoop。

```
<dependency>
    <groupId>org.apache.hadoop</groupId>
    <artifactId>hadoop-client</artifactId>
    <version>3.3.5</version>
</dependency>
```

图 2 添加 hadoop 依赖

创建 Mapper 子类 WordCountMapper，用于将字符串分块，并生成键值对（key, value），为 Reducer 提供中间数据，将每个单词及其出现的次数（初始为 1）传递给下游处理，最终实现单词计数功能：（并且对特殊字符和数字进行了过滤）

```java
public class WordCountMapper extends Mapper<LongWritable,Text,Text, IntWritable>{

    @Override
    protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
        // 获取行内容
        String line = value.toString();
        // 按空格拆分得到单词数组
        String[] words = line.split( regex: " "); //regex正则表达式
        // 遍历单词数组，生成输出键值对
        for (String word : words) {
            // 过滤特殊字符、数字，只保留字母
            if (word.matches( regex: "[a-zA-Z]+"))
                context.write(new Text(word), new IntWritable( value: 1));
        }

    }
}
```

图 3 Map 阶段的代码实现

接着，创建 Reducer 的子类 WordCountReducer，用来接收来自 Map 任务的中间结果，对每个键（k）及其关联的值列表（list(v)）执行用户定义的 reduce 函数，将相同单词的词频合并累加起来，从而统计出每个单词的词频。

```java
public class WordCountReducer extends Reducer<Text,IntWritable,Text,IntWritable> {

    @Override
    protected void reduce(Text key, Iterable<IntWritable> values, Context context)
            throws IOException, InterruptedException {
        // 定义输出键出现次数
        int count = 0;
        // 遍历输出值迭代对象，统计其出现次数
        for (IntWritable value : values) {
            count = count + value.get();
        }
        // 生成键值对输出
        context.write(key, new IntWritable(count));
    }
}
```

图 4 Reduce 阶段的代码实现

最后，编写程序的主驱动类 WordCountDriver，它负责配置并运行整个 MapReduce 作业，同时对结果进行处理和排序，并输出到文件。

A. 配置和初始化 MapReduce 作业

```java
// 创建配置对象
Configuration conf = new Configuration();
// 设置数据节点主机名属性
conf.set("dfs.client.use.datanode.hostname", "true");

// 获取作业实例
Job job = Job.getInstance(conf);
// 设置作业启动类
job.setJarByClass(WordCountDriver.class);

// 设置Mapper类
job.setMapperClass(WordCountMapper.class);
```

图 5 初始化过程的部分代码

B. 根据输入选择运行模式

因为数据源包含两个文件：小文件和大文件，对小文件而言，我们可以不采用分区的处理方式；对于大文件而言，进行分区的效率会更好。同时，两个文件的存放路径以及输出路径也不相同，于是，我们需要根据用户输入来选择不同的逻辑进行处理，提高程序的健壮性、完整性。

```
// 选择小文件还是大文件：1.小文件 2.大文件
Scanner sc = new Scanner(System.in);
System.out.println("请输入模式：1.小文件 2.大文件");
int mode = sc.nextInt();
String fileName = "test1.txt";
if (mode == 1){
    // 设置分区数量（reduce任务的数量、结果文件的数量,默认为 1）
    job.setNumReduceTasks(1);
} else if (mode == 2){
    // 设置分区数量（reduce任务的数量、结果文件的数量,默认为 1）
    job.setNumReduceTasks(6);
    fileName = "test2.txt";
}
```

图 6 选择运行模式（小文件或大文件）

C. 设置输入输出路径

```
// 定义文件夹路径
String dicRoot = "C:\\Users\\24312\\Desktop\\智能网络\\实验四\\WordCounter";
// 创建输入目录
Path inputPath = new Path( pathString: dicRoot + "/data/" + fileName);
// 创建输出目录
Path outputPath = new Path( pathString: dicRoot + "/result" + mode);

// 获取文件系统
FileSystem fs = FileSystem.getLocal(conf);
// 删除输出目录（第二个参数设置是否递归）
fs.delete(outputPath,  b: true);

// 给作业添加输入目录（允许多个）
FileInputFormat.addInputPath(job, inputPath);
// 给作业设置输出目录（只能一个）
FileOutputFormat.setOutputPath(job, outputPath);
```

图 7 设置输入输出路径

D. 读取词频统计结果并保存在 Map 容器中

```
Map<String, Integer> wordCountMap = new HashMap<>();// 用于存储统计结果
// 读取结果文件
for (int i = 0; i < fileStatuses.length; i++) {
    if (fileStatuses[i].isFile()) { // 确保是文件
        FSDataInputStream in = fs.open(fileStatuses[i].getPath());
        Scanner scanner = new Scanner(in);
        while (scanner.hasNextLine()) {
            String line = scanner.nextLine();
            String[] parts = line.split( regex: "\t");
            if (parts.length == 2) {
                String word = parts[0];
                int count = Integer.parseInt(parts[1]);
                wordCountMap.put(word, count);
            }
        }
        scanner.close();
        in.close();
    }
}
```

图 8 将结果暂存在 Map 中

这里的目的是为了方便接下来在程序中借助 list 容器对词频统计结果进行降序排序。

E. 将统计结果进行降序排序，并输出到文件中

```
// 对结果排序
List<Map.Entry<String, Integer>> sortedEntries = new ArrayList<>(wordCountMap.entrySet());
sortedEntries.sort((o1, o2) -> o2.getValue().compareTo(o1.getValue())); // 降序排序

// 将排序结果写回文件
Path sortedResultPath = new Path( pathString: outputPath + "/Sta_" + fileName);
FSDataOutputStream out = fs.create(sortedResultPath);

for (Map.Entry<String, Integer> entry : sortedEntries) {
    out.writeBytes( s: entry.getKey() + "\t" + entry.getValue() + "\n");
}
out.close();
```

图 9 排序并输出

（2）检验程序的正确性

运行程序，选择（1.小文件）模式，但是，程序出现报错提示：

```
Exception in thread "main" java.lang.RuntimeException Create breakpoint Lingma → : java.io.FileNotFoundException: java.io.FileNotFoundException: HADOOP_HOME and hadoop.home.dir
are unset. -see https://wiki.apache.org/hadoop/WindowsProblems
    at org.apache.hadoop.util.Shell.getWinUtilsPath(Shell.java:788)
    at org.apache.hadoop.util.Shell.getSetPermissionCommand(Shell.java:297)
    at org.apache.hadoop.util.Shell.getSetPermissionCommand(Shell.java:313)
    at org.apache.hadoop.fs.RawLocalFileSystem.setPermission(RawLocalFileSystem.java:1114)
    at org.apache.hadoop.fs.RawLocalFileSystem.mkOneDirWithMode(RawLocalFileSystem.java:796)
    at org.apache.hadoop.fs.RawLocalFileSystem.mkdirsWithOptionalPermission(RawLocalFileSystem.java:836)
    at org.apache.hadoop.fs.RawLocalFileSystem.mkdirs(RawLocalFileSystem.java:808)
    at org.apache.hadoop.fs.RawLocalFileSystem.mkdirsWithOptionalPermission(RawLocalFileSystem.java:835)
    at org.apache.hadoop.fs.RawLocalFileSystem.mkdirs(RawLocalFileSystem.java:808)
    at org.apache.hadoop.fs.ChecksumFileSystem.mkdirs(ChecksumFileSystem.java:997)
```

图 10 程序报错

（3）解决报错问题

由报错提示信息，我们大概就可以猜测到，我们缺少了一个 HADOOP_HOME 的系统变量以及 util.Shell 什么的插件。

点进报错提示信息中的蓝色链接，我们可以看到解决方法：

1.下载 HADOOP.DLL 和 WINUTILS.EXE； 2.配置 HADOOP_HOME 环境变量。

WindowsProblems

由 ASF Infrabot创建于七月 09, 2019

Problems running Hadoop on Windows

Hadoop requires native libraries on Windows to work properly -that includes to access the `file://` filesystem, where Hadoop uses some Windows APIs to implement posix-like file access permissions.

This is implemented in `HADOOP.DLL` and `WINUTILS.EXE`

In particular, `%HADOOP_HOME%\BIN\WINUTILS.EXE` must be locatable.

If it is not, Hadoop or an application built on top of Hadoop will fail.

How to fix a missing WINUTILS.EXE

You can fix this problem in two ways

1. Install a full native windows Hadoop version. The ASF does not currently (September 2015) release such a version; releases are available externally.
2. Or: get the `WINUTILS.EXE` binary from a Hadoop redistribution. There is a repository of this for some Hadoop versions on github.

Then

1. Set the environment variable `%HADOOP_HOME%` to point to the directory above the `BIN` dir containing `WINUTILS.EXE`.
2. Or: run the Java process with the system property `hadoop.home.dir` set to the home directory.

图 11 解决报错的方案

（4）通过查询网络资料，我找到了 winutils.exe 的下载资源，并下载到本地

下载网址：[cdarlint/winutils: winutils.exe hadoop.dll and hdfs.dll binaries for hadoop windows]

下载结果：

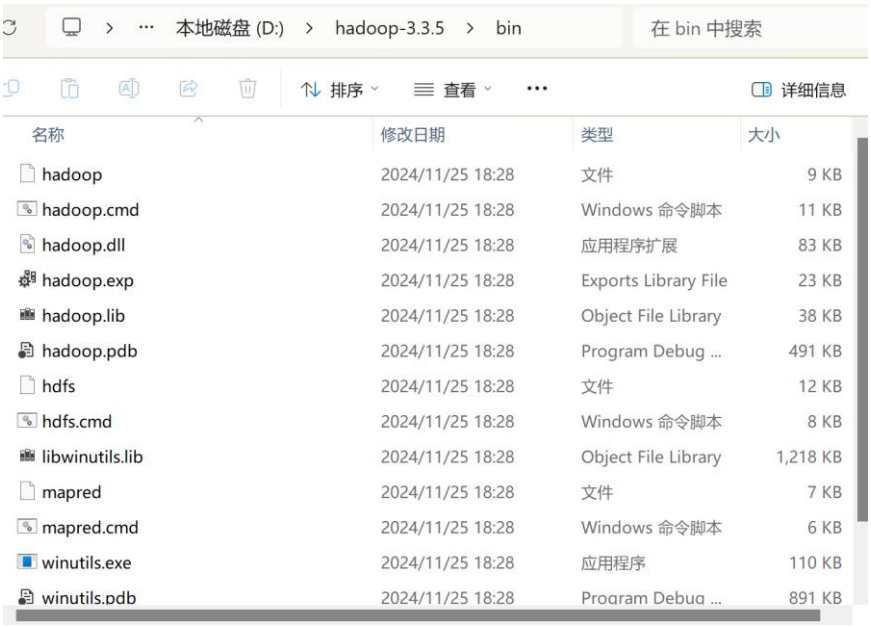

图 12 winutils 的目录

（5）配置环境变量

如果没有配置环境变量，就会由于找不到 winutils.exe 而报错。在此，我们需要根据官方文档的指示，将 HADOOP_HOME 配置为 winutils.exe 的父级目录路径，然后在将该路径下的 bin 目录添加到 path 环境变量中。

A. 将 HADOOP_HOME 配置为 winutils.exe 的父级目录路径：

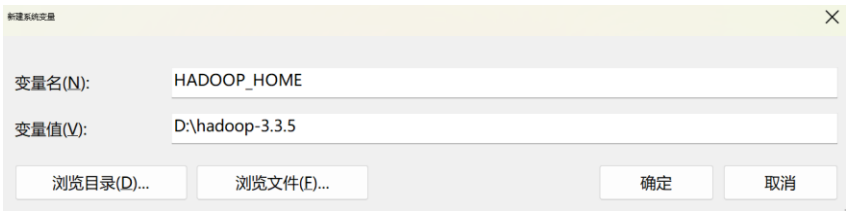

图 13 配置 HADOOP_HOME 变量

B. 将该路径下的 bin 目录添加到 path 环境变量中：

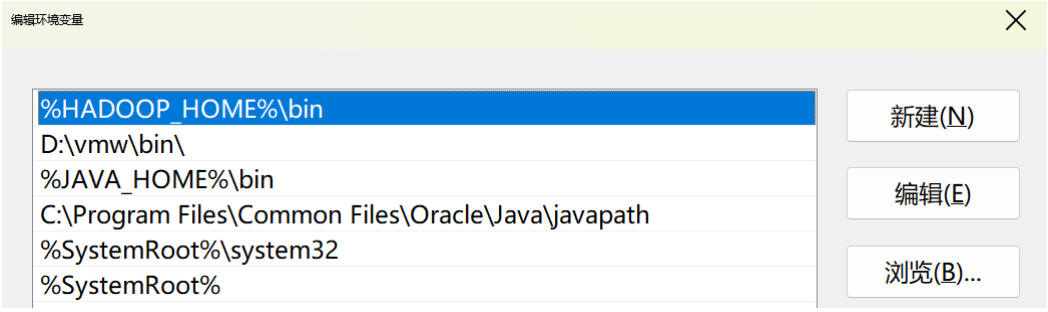

图 14 修改 path 环境变量

（6）运行程序统计词频

A. 选择小文件模式，统计小文件的词频并输出到 Sta_前缀的结果文件中：

```
请输入模式：1.小文件 2.大文件
1
======统计结果======
排序结果写入：C:/Users/24312/Desktop/智能网络/实验四/WordCounter/result1/Sta_test1.txt
```

图 15 控制台输出情况

图 16 生成的结果文件（一）

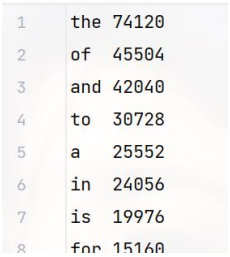

图 17 部分词频统计结果

B. 选择大文件模式，统计大文件的词频并输出到 Sta_前缀的结果文件中：

```
请输入模式：1.小文件 2.大文件
2
======统计结果======
排序结果写入：C:/Users/24312/Desktop/智能网络/实验四/WordCounter/result2/Sta_test2.txt
```

图 18 控制台输出结果

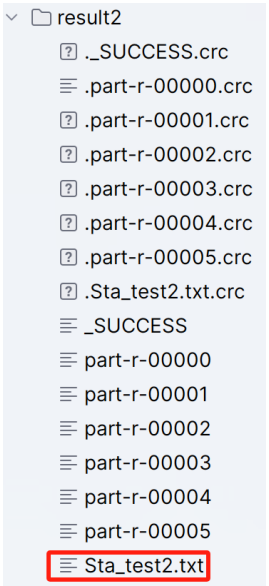

图 19 输出的结果文件（二）

```
and 2261061
the 2171790
of  1819220
a   1128206
to  1099983
in  1064735
on  768568
for 691004
is  641661
```

图 20 部分词频统计结果

3. 探索：使用 spark 框架实现词频统计任务

（1）安装 spark 框架

安装链接：https://spark.apache.org/downloads.html

（2）使用 pyspark（一个基于 Apache Spark 的 Python API）进行词频统计：

基本代码如下，对于两个测试示例只需要更改输入和输出的文件即可：

```
import time
lines = sc.textFile("./test2.txt")
start_time = time.time()
wordCount_1 = lines.flatMap(lambda line:line.split(" "))
wordCount_2 = wordCount_1.map(lambda x:(x,1))
wordCount_3 = wordCount_2.reduceByKey(lambda a,b:a+b)
#print(wordCount_3.collect())
sorted_result = wordCount_3.sortBy(lambda x: x[1], ascending=False)
sorted_result.saveAsTextFile("./ansB.txt")
end_time = time.time()
execution_time = end_time - start_time
print("execution time: ", execution_time)
```

（3）在命令行中启动 pyspark：

```
└─$ ./pyspark
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
Python 3.11.4 (main, Jun  7 2023, 10:13:09) [GCC 12.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
24/01/06 17:48:01 WARN Utils: Your hostname, kali resolves to a loopback address: 127.0.1.1; us
ing 192.168.206.128 instead (on interface eth0)
24/01/06 17:48:01 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
24/01/06 17:48:04 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform
 ... using builtin-java classes where applicable
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 3.4.0
      /_/

Using Python version 3.11.4 (main, Jun  7 2023 10:13:09)
Spark context Web UI available at http://192.168.206.128:4040
```

图 21 启动 pyspark

（4）对样例 A 进行词频统计并输出：

在命令行中输入设计好的 python 代码，统计小文件的词频：

```
>>> import time
>>> lines = sc.textFile("./A.txt")
>>> start_time = time.time()
>>> wordCount_1 = lines.flatMap(lambda line:line.split(" "))
>>> wordCount_2 = wordCount_1.map(lambda x:(x,1))
>>> wordCount_3 = wordCount_2.reduceByKey(lambda a,b:a+b)
>>> #print(wordCount_3.collect())
>>> sorted_result = wordCount_3.sortBy(lambda x: x[1], ascending=False)
>>> sorted_result.saveAsTextFile("./ansA.txt")
>>> end_time = time.time()
>>> execution_time = end_time - start_time
>>> print("execution time: ", execution_time)
execution time:  4.102804899215698
```

图 22 命令行输入 python 代码

```
the 74120
of 45504
and 42040
to 30728
a 25552
in 24056
is 19976
for 15160
on 11888
```

图 23 小文件的词频统计结果

（5）对样例 B 进行词频统计并输出：

在命令行中输入设计好的 python 代码，统计小文件的词频：

```
>>> import time
>>> lines = sc.textFile("./test2.txt")
>>> start_time = time.time()
>>> wordCount_1 = lines.flatMap(lambda line:line.split(" "))
>>> wordCount_2 = wordCount_1.map(lambda x:(x,1))
>>> wordCount_3 = wordCount_2.reduceByKey(lambda a,b:a+b)
>>> #print(wordCount_3.collect())
>>> sorted_result = wordCount_3.sortBy(lambda x: x[1], ascending=False)
>>> sorted_result.saveAsTextFile("./ansB.txt")
>>> end_time = time.time()
>>> execution_time = end_time - start_time
>>> print("execution time: ", execution_time)
execution time:  61.26446747779846
```

图 24 命令行输入 python 代码

```
and 2261061
the 2171790
of 1819220
a 1128206
to 1099983
in 1064735
on 768568
for 691004
```

图 25 大文件的词频统计结果

对比上面的执行时间，可以发现使用了 spark 计算框架的运行时间总体来说要小于直接使用 java 编程的执行时间。

对于像测试文件 A 这样相对的小数据来说，使用框架与直接编程的 MapReduce 实现消耗时间差不多，而对于测试机 B 这样的大数据来说，使用 spark 计算框架消耗的时间要远小于直接编程实现的 MapReduce，其原因主要如下：

A. In-Memory 计算

Spark 使用 RDD（弹性分布式数据集）作为基本数据抽象，可以将数据缓存在内存中，允许在迭代计算中重复使用数据。相比之下，传统的 MapReduce 在每个阶段都需要将数据写入磁盘，这会导致高额的 IO 开销。Spark 利用内存的特性，避免了这种磁盘读写，因而执行速度更快。

B. DAG 调度执行

Spark 使用基于 DAG（有向无环图）的调度器，在执行作业之前会构建一个作业的执行计划。这使得 Spark 可以优化作业的执行顺序、合并多个操作，减少不必要的中间结果存储，从而提高了执行效率。

C. **延迟执行**

Spark 中的转换操作（如 map、filter）是惰性的，只有遇到行动操作（如 collect、saveAsTextFile）时才会触发实际的计算。这种延迟执行的机制允许 Spark 在执行之前进行优化和调整计算流程。

D. **内建的高级 API 和工具**

Spark 提供了丰富的高级 API（如 DataFrame、Dataset 等）和丰富的工具（如 MLlib、Spark SQL 等），使得开发者可以更快速地进行大规模数据处理和机器学习任务，而无需手动实现复杂的 MapReduce 过程。

实验结论：

（此处写实验结果以及对结果的分析、讨论，另外写实验过程中遇到的问题以及解决办法）

在本次实验中，我深入理解并实践了 MapReduce 计算模型的原理和应用。通过阅读经典论文“MapReduce: Simplified data processing on large clusters，OSDI 2004”，我掌握了 MapReduce 的基本框架和工作机制。

在实验过程中，我使用 Java 编程语言实现了 MapReduce 的 Map 和 Reduce 功能，完成了对给定文本文件的词频统计任务。

除了传统的编程实现，利用本地的 Apache Spark 计算框架也是一个很有意义的探索。Spark 提供了高效的分布式计算能力，对于大规模数据的处理有着良好的支持。通过对比小文件和大文件的处理，我发现对于小文件，直接编程实现的 MapReduce 与使用 Spark 框架的执行时间相差不大，而对于大文件，Spark 框架的执行时间更短，这主要得益于 Spark 的 In-Memory 计算、DAG 调度执行、延迟执行以及内建的高级 API 和工具。

实验难题：

1. 环境配置问题

首次运行程序时，报错提示缺少 HADOOP_HOME 系统变量以及 util.Shell 插件。通过查阅官方文档和网络资料，我下载了必要的 Hadoop DLL 和 WINUTILS.EXE 文件，并正确配置了 HADOOP_HOME 环境变量，解决了环境配置问题。

2. 特殊字符过滤问题

在实现 MapReduce 的过程中，对于特殊字符和数字的过滤处理不够完善，导致词频统计结果不准确。我通过修改 Map 函数，增加了对特殊字符和数字的过滤逻辑，提高了词频统计的准确性。

心得体会：

（此处随便写写你的所想、所得，或者建议）

通过这次实验，我深刻体会到了 MapReduce 框架在处理大规模数据集时的优势。它通过将复杂的数据处理任务分解为简单的 Map 和 Reduce 操作，极大地简化了并行计算的实现。

同时，我也认识到了 Spark 框架在现代大数据处理中的重要性。Spark 不仅提高了数据处理的效率，还提供了更为丰富的数据处理工具和 API，使得大规模数据处理变得更加灵活和高效。此外，我也体会到了在实际开发中遇到问题时，如何通过查阅资料和社区支持来解决问题的重要性。

总的来说，我有三个方面的心得：

1. 数据规模带来的挑战

处理大文件的词频统计可能面临内存和计算资源的挑战。在本次实验中，可以已经可以看到大文件本身首先是占用内存较大的，计算量也显然十分大，从运行对大文件的词频分析速度之慢可以感受到这一点。针对大文件，需要考虑分块处理、优化内存占用等问题，保证程序的稳定性和效率。

2. 深入理解分布式计算

实践中理解 MapReduce 并使用 Spark 框架让我更深入地理解了分布式计算的优势和挑战。能够针对大规模数据实现词频统计的经历加深了对分布式计算模型的认识。

3. 技术实践与工具应用

在实现词频统计任务的过程中，熟悉了编程语言的 MapReduce 实现方式，并学习了如何配置和使用 Spark 框架进行分布式计算。这些技术实践对于今后处理大数据场景下的任务会有很大帮助。

指导教师批阅意见：

成绩评定：　　分

指导教师签字：车越岭

年　　月　　日

备注：