Spark 实验部分

实验一 Spark 安装与配置实验

【实验目的】

通过本实验，学习如何在 Linux 环境下配置大数据分析引擎 Spark 的安装与使用，为后续的 Spark 大数据处理和分析工作做好准备。

【实验内容】

1. 安装 Linux 系统（建议通过虚拟机）。
2. 在 Linux 系统配置 Hadoop 基础环境。
3. 安装 Java 环境
4. 安装 Spark (选择和 Hadoop 集成的安装包)

【实验环境】

1. 操作系统：Linux（Ubuntu）
2. 软件环境：Java JDK 1.8、Spark3.4
3. 硬件要求：至少 1 台计算机或虚拟机，建议配置至少 4GB 内存和 100GB 的硬盘空间用于安装 Spark。
4. 网络连接：互联网连接，用于下载所需的软件和文档。

【实验步骤】

1. 用浏览器访问 spark.apache.org，打开 documentation 下面 latest release 页面，阅读 Spark 概述。下载 spark-3.4.2-bin-hadoop3.tgz
2. tar 解压到 home 目录某文件下: tar -xzvf spark-3.4.2-bin-hadoop3.tgz
3. 参考文档说明，选择单机模式(standalone deplogy mode)，spark 也可以单独运行。
4. bin 目录下./spark-shell 运行。

若分开安装 Hadoop 和 Spark，请参考 https://dblab.xmu.edu.cn/blog/804/

Spark 实验部分

实验二 WordCount 实验

【实验目的】

本实验旨在通过编写和执行基于 Spark 编程模型的 WordCount 程序，帮助学生深入理解 Spark 的工作原理，并学会使用 Spark 框架进行大规模数据处理。通过此实验，学生将能够掌握 Spark 编程的基本概念、编写简单的 Spark 程序以及运行它们在分布式环境中。

【实验内容】

1. Scala 程序实现 WordCount，掌握 RDD 和函数式编程思想。
2. 使用命令行执行 WordCount 程序。
3. 使用 Eclipse 编译、打包 WordCount 程序。
4. 查看程序执行结果。

【实验环境】

1. 操作系统：Linux（Ubuntu）
2. 软件环境：Java JDK 1.8、Spark3.4
3. 硬件要求：至少 1 台计算机或虚拟机，建议配置至少 4GB 内存和 100GB 的硬盘空间用于安装 Hadoop。
4. 网络连接：互联网连接，用于下载所需的软件和文档。

【实验步骤】

1. 使用命令行执行 scale 代码，注意双引号格式，在 spark-shell 中读取 Linux 系统本地文件 **file:///home/stu/software/hadoop/README.txt，统计**“Hadoop”单词的个数。

scala> val sc = new SparkContext( )

scala> val textFile = sc.textFile(“file:///home/stu/software/hadoop/README.txt”)

scala> textFile.count()

```
scala> val textFile = sc.textFile("file:///home/stu/software/hadoop/README.txt")
textFile: org.apache.spark.rdd.RDD[String] = file:///home/stu/software/hadoop/README.txt MapPart
sole>:24

scala> textFile.count()
res6: Long = 31
```

scala> val linesWithHadoop = textFile.filter(line => line.contains(“Hadoop”))

scala> linesWithHadoop.count()

```
scala> val linesWithHadoop = textFile.filter(line => line.contains("Hadoop"))
linesWithHadoop: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[8] at filter at <console>:25

scala> linesWithHadoop.count()
res7: Long = 2
```

上面两条语句可以通过链式操作合并为：

scala> val linesCountWithHadoop = textFile.filter(line => line.contains(“Hadoop”)).count()

```
scala> val linesCountWithHadoop = textFile.filter(line => line.contains("hadoop")).count()
linesCountWithHadoop: Long = 2
```

2. 在 spark-shell 中读取 HDFS 系统文件“/user/hadoop/test.txt”（如果该文件不存在，请先创建），然后，统计出文件的行数；

scala> val lines = sc.textFile("hdfs://localhost:9000/user/hadoop/test.txt")

```
(base) hadoop@ubuntu:/usr/local/hadoop$ bin/hdfs dfs -put ~/test.txt /user/hadoop
2023-05-21 15:30:53,718 INFO sasl.SaslDataTransferClient: SASL encryption trust check: loca
lHostTrusted = false, remoteHostTrusted = false
(base) hadoop@ubuntu:/usr/local/hadoop$ bin/hdfs dfs -ls /user/hadoop
Found 3 items
drwxr-xr-x   - hadoop supergroup          0 2023-05-06 19:26 /user/hadoop/input
drwxr-xr-x   - hadoop supergroup          0 2023-05-06 19:27 /user/hadoop/output
-rw-r--r--   1 hadoop supergroup         25 2023-05-21 15:30 /user/hadoop test.txt
(base) hadoop@ubuntu:/usr/local/hadoop$
```

```
scala> val textFile=sc.textFile("hdfs://localhost:9000/user/hadoop/test.txt")
textFile: org.apache.spark.rdd.RDD[String] = hdfs://localhost:9000/user/hadoop/test.txt Map
PartitionsRDD[3] at textFile at <console>:24

scala> textFile.count()
res1: Long = 5

scala>
```

3. 使用命令行来实现 wordcount，即统计文件中各个单词的计数：

scala> val sc = new SparkContext( )

scala> val textFile = sc.textFile(“file:///home/stu/software/hadoop/README.txt”)

scala> val wordCounts = textFile.flatmap(line => line.split(“ ”)).map(word => (word,1)).reduceByKey((a,b) => a+b)

scala> wordCount.collect()

scala> wordCount.foreach(println)

4. 用 scale 语言实现 wordcount

WordCount 程序参考代码 1：

```
import java.io.File
import scala.io.Source

object WordCountApp {

  def main(args: Array[String]): Unit = {
    //文件路径
    val filePath = "file:///home/stu/software/hadoop/README.txt "
    val codec = "utf-8"
    //打开文件
    val file = Source.fromFile(filePath, codec)
    val wc = file.getLines().flatMap(_.split("\t")).toList.map((_, 1)).groupBy((_._1)).mapValues(_.size)
    println(wc)
    // 关闭文件
    file.close()
  }
}
```

WordCount 程序参考代码 2：

```
import org.apache.spark.rdd.RDD
import org.apache.spark.{SparkConf, SparkContext}

/** * Scala 原生实现 wordcount */
object WordCount {
  def main(args: Array[String]): Unit = {

    val list = List("cw is cool", "wc is beautiful", "andy is beautiful", "mike is cool")
```

```
        /** * 第一步，将 list 中的元素按照分隔符这里是空格拆分，然后展开 * 先 map(_.split(" "))将每一个元素按照空格拆分 * 然后 flatten 展开 * flatmap 即为上面两个步骤的整合 */
        val res0 = list.map(_.split(" ")).flatten
        val res1 = list.flatMap(_.split(" "))
        println("第一步结果")
        println(res0)
        println(res1)

        /** * 第二步是将拆分后得到的每个单词生成一个元组 *k 是单词名称，v 任意字符即可这里是 1 */
        val res3 = res1.map((_, 1))
        println("第二步结果")
        println(res3)
        /** * 第三步是根据相同的 key 合并 */
        val res4 = res3.groupBy(_._1)
        println("第三步结果")
        println(res4)
        /** * 最后一步是求出 groupBy 后的每个 key 对应的 value 的 size 大小，即单词出现的个数 */
        val res5 = res4.mapValues(_.size)
        println("最后一步结果")
        println(res5.toBuffer)
    }
  }
```

5. 试一试，是否还有其他的实现方式？

Spark 实验部分

实验三 Spark Streaming 实验

【实验目的】

本实验旨在通过编写和执行基于 Spark Streaming 编程模型的 wordcount 程序，帮助学生深入理解 Spark Streaming 的工作原理，并学会使用 Spark 框架进行大规模数据处理。通过此实验，学生将能够掌握 Spark 编程的基本概念、编写简单的 Spark 程序以及运行它们在分布式环境中。

【实验内容】

基于 Spark Streaming 编程模型实现 wordcount 程序。编写 Spark Streaming 程序的基本步骤是：

1.创建 SparkSession 实例；

2.创建 DataFrame 表示从数据源输入的每一行数据；

3.DataFrame 转换，类似于 RDD 转换操作；

4.创建 StreamingQuery 开启流查询；

5.调用 StreamingQuery.awaitTermination()方法，等待流查询结束。

【实验环境】

1. 操作系统：Linux（Ubuntu）
2. 软件环境：Java JDK 1.8、Spark3.4
3. 硬件要求：至少 1 台计算机或虚拟机，建议配置至少 4GB 内存和 100GB 的硬盘空间用于安装 Hadoop。
4. 网络连接：互联网连接，用于下载所需的软件和文档。

【实验步骤】

把 wordcount 程序写成单独一个 scala 文件，然后提交给 spark 执行；

1. 首先打开一个终端，输入以下命令:

```
cd /usr/local/spark/mycode
mkdir streaming
```

```
cd streaming
mkdir -p src/main/scala
cd src/main/scala
vim TestStructuredStreaming.scala
```

2. 用 vim 打开一个 scala 文件，在该文件输入以下内容:

```
import org.apache.spark.sql.functions._
import org.apache.spark.sql.SparkSession

object WordCountStructuredStreaming{
  def main(args: Array[String]){
  val spark = SparkSession.builder.appName("StructuredNetworkWordCount").
getOrCreate()
  import spark.implicits._
  val lines = spark.readStream.format("socket").option("host","localhost").
option("port",9999).load()
  val words = lines.as[String].flatMap(_.split(" "))
  val wordCounts = words.groupBy("value").count()
  val query = wordCounts.writeStream.outputMode("complete").
format("console").start()
 query.awaitTermination()
   }
 }
```

3. 代码写好之后，退出终端，然后在/usr/lcoal/spark/mycode/streaming 目录下创建 simple.sbt 文件:

```
cd /usr/local/spark/mycode/streaming
vim simple.sbt
```

打开 vim 编辑器以后，输入以下内容:

```
name := "Simple Project"
version := "1.0"
scalaVersion := "2.11.8"
libraryDependencies += "org.apache.spark" %% "spark-sql" % "2.0.0"
```

4. 通过执行 sbt 打包编译:

```
cd /usr/local/spark/mycode/streaming
/usr/local/sbt/sbt package
```

打包成功以后，就可以输入以下命令启动这个程序:

```
cd /usr/local/spark/mycode/streaming
/usr/local/spark/bin/spark-submit --class "WordCountStructuredStreaming" ./target/scala-2.11/simple-project_2.11-1.0.jar
```

执行上输出程序之后，就开启了监听状态，当我们在终端输入“hello world，hello Beijing, hello world”之后，spark 应用终端即可输出”hello”和”world”单词出现的次数。