实验三：Spark

1. 实验目的

* 学习Spark分布式处理框架，理解RDD概念
* 学习使用spark-shell基本命令，使用spark编写简单的程序

1. 任务与要求

* 学习Spark分布式处理框架，理解RDD的概念
* 学习spark-shell的常用指令，掌握如何进入、退出spark-shell以及如何提交任务
* 编写WordCount程序，对样例数据进行词频统计
* 编写均值方差计算程序，计算数值数据的均值与方差

1. 实验原理

3.1． Spark

Spark是分布式批处理框架，提供分析挖掘与迭代式内存计算能力，支持多种语言（Scala/Java/Python）的应用开发。 适用以下场景：

* 数据处理（Data Processing）：可以用来快速处理数据，兼具容错性和可扩展性。
* 迭代计算（Iterative Computation）：支持迭代计算，有效应对多步的数据处理逻辑。
* 数据挖掘（Data Mining）：在海量数据基础上进行复杂的挖掘分析，可支持各种数据挖掘和机器学习算法。
* 流式处理（Streaming Processing）：支持秒级延迟的流式处理，可支持多种外部数据源。
* 查询分析（Query Analysis）：支持标准SQL查询分析，同时提供DSL（DataFrame）， 并支持多种外部输入。

3.2．RDD

RDD全名为弹性分布数据集（Resilient Distributed Dataset），它是Spark的核心概念，指的是一个只读，可变分区的分布式数据集，该数据集的内容可以缓存在内存中，并在多次计算中间重用。

RDD的生成途径：

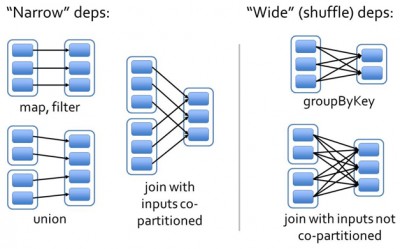
* 从Hadoop文件系统（或与Hadoop兼容的其它存储系统）输入（例如HDFS）创建 。
* 从父RDD转换得到新RDD。
* 从集合转换而来。

RDD的存储：

* 用户可以选择不同的存储级别（例如DISK\_ONLY，MEMORY\_AND\_DISK）存储RDD以便重用。
* 当前RDD默认只存储于内存，当内存不足时，RDD也不会溢出到磁盘中。

3.2. Dependency（RDD的依赖）

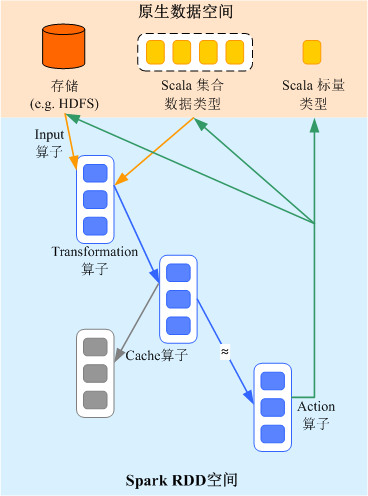
父RDD与子RDD之间的逻辑关系，可分为窄依赖和宽依赖两种。



* 窄依赖：指父RDD的每一个分区最多被一个子RDD的分区所用，表现为一个父RDD的分区对应于一个子RDD的分区，或者两个父RDD的分区对应于一个子RDD的分区。上图中，map/filter和union属于第一类前者，对输入进行协同划分（co-partitioned）的join属于第二类后者。
* 宽依赖：指子RDD的分区依赖于父RDD的所有分区，子RDD必须要通过shuffle类操作实现父RDD的数据全部重新分配，如上图中的groupByKey和未经协同划分的join。

3.3．Transformation和Action（RDD的操作）

对RDD的操作包含Transformation（返回值还是一个RDD）和Action（返回值不是一个RDD）两种。RDD的操作流程如下图所示。其中Transformation操作是Lazy的，也就是说从一个RDD转换生成另一个RDD的操作不是马上执行，Spark在遇到Transformation操作时只会记录需要这样的操作，并不会去执行，需要等到有Action操作的时候才会真正启动计算过程进行计算。Action操作会返回结果或把RDD数据写到存储系统中。Action是触发Spark启动计算的动因。

3.4．Shuffle

Shuffle是Spark框架中一个特定的phase，当Map的输出结果要被Reduce使用时，输出结果需要按key哈希，并且分发到每一个Reducer上去，这个过程就是shuffle。由于shuffle涉及到了磁盘的读写和网络的传输，因此shuffle性能的高低直接影响到了整个程序的运行效率。

右图描述了Shuffle算法的整个流程。

1. 实验步骤

**4.1 任务1. 掌握spark-shell常用指令的使用**

spark-shell是一个强大的交互式分析数据工具，同时也是一种学习Spark的有效工具，它可以使用scala或python编写，本课程主要介绍scala。

接下来学习spark-shell的使用。首先，登录服务器（原先的节点出了点问题，更换了新的登录节点，IP地址会发布在群里），开启spark-shell：

|  |
| --- |
| **2018211009@thumm01:~$ spark-shell**  Setting default log level to "WARN".  To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).  Spark Context Web UI is available at Spark Master Public URL  Spark context available as 'sc' (master = spark://thumm01:7077, app id = app-20191110185959-0001).  Spark session available as 'spark'.  Welcome to  \_\_\_\_ \_\_  / \_\_/\_\_ \_\_\_ \_\_\_\_\_/ /\_\_  \_\ \/ \_ \/ \_ `/ \_\_/ '\_/  /\_\_\_/ .\_\_/\\_,\_/\_/ /\_/\\_\ version 2.4.4  /\_/  Using Scala version 2.12.8 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0\_221)  Type in expressions to have them evaluated.  Type :help for more information.  scala> |

Spark-shell除了可以使用scala语言操作外，还有一些基本指令，这些指令都以“:”开头，指令的用法可以使用 **:help**查看：

|  |
| --- |
| **scala> :help**  All commands can be abbreviated, e.g., :he instead of :help.  :completions <string> output completions for the given string  :edit <id>|<line> edit history  :help [command] print this summary or command-specific help  :imports [name name ...] show import history, identifying sources of names  **:load <path> interpret lines in a file**  **:quit exit the interpreter**  :reset [options] reset the repl to its initial state, forgetting all session entries  **:save <path> save replayable session to a file**  :sh <command line> run a shell command (result is implicitly => List[String])  :settings <options> update compiler options, if possible; see reset  ...... |

其中几个常用的指令有:

* :quit 退出spark-shell控制台
* :load <path> 加载使用scala编写的spark-shell脚本
* :save <path> 将当前上下文的历史指令保存为文件

当我们在spark-shell中调试程序完成后，我们可以使用:save指令将历史保存到一个文件，我们可以使用这个文件恢复之前的调试的上下文，也能将其做成一个具有一定功能的脚本。

**4.2 使用Spark进行词频统计**

接下来我们使用spark来做词频统计。在上一次实验中，我们使用Java编写MapReduce程序进行词频统计, 那个程序如果让我们自己编写我想大部分人会不知所措。这次实验课我们使用spark实现词频统计的功能，只需短短几行就能实现词频统计的功能。

首先，进入在服务器命令行输入，进入spark-shell。

|  |
| --- |
| **2018211009@thumm01:~$ spark-shell**  ......  scala> |

接下来，我们需要加载待统计词频的数据集。

|  |
| --- |
| **scala> val textFile = sc.textFile("/wc\_dataset.txt")**  textFile: org.apache.spark.rdd.RDD[String] = /wc\_dataset.txt MapPartitionsRDD[1] at textFile at <console>:24 |

这句命令中, sc（Spark-Context）是spark-shell的上下文，这个变量是进入spark-shell就有的，可以用来设置一些运行参数；val textFile是定义一个变量名为textFile的变量，它的值是使用sc.textFile函数加载HDFS中/wc\_dataset.txt文件的内容。

接下来可以查看这个textFile的内容：

|  |
| --- |
| **scala> textFile.first() #查看第一行**  res0: String = chapter  **scala> textFile.count() #查看行数**  res1: Long = 2683500 |

使用一行代码统计词频：

|  |
| --- |
| **scala> val result = textFile.flatMap(l => l.split(" ")).map(w => (w, 1)).reduceByKey(\_ + \_)**  result: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[4] at reduceByKey at <console>:25  **scala> result.first() #查看结果的第一行内容**  res3: (String, Int) = (someone,100)  **scala> result.saveAsTextFile("wc\_output")** |

接下来我们来解释这行代码，代码的流程如下图所示：

****

* 从HDFS中加载文件wc\_dataset.txt， 保存内容到变量textFile
* 对textFile逐行处理，对每一行按空格进行分割，得到一个字符串列表
* 使用map将字符串列表转成一个键值对列表[<key1, value1>, <key2, value2>, ……],其中键为单词，值为词频（没有合并之前为1）。
* 将不同的键值对根据相同的键不断地合并，直至无法合并，得到词频统计结果。
* 将结果保存到HDFS中(保存到了/user/学号/wc\_output)。

使用:quit指令退出spark-shell，然后将结果复制到本地：

|  |
| --- |
| **scala> :quit**  **2018211009@thumm01:~$ hadoop fs -get wc\_output ./**  **2018211009@thumm01:~$ cd wc\_output/**  **2018211009@thumm01:~/wc\_output$ ls**  \_SUCCESS part-00000 part-00001  **2018211009@thumm01:~/wc\_output$ head -n 5 part-00000**  (someone,100)  (bone,100)  (doubtfully,200)  (order,300)  (spirited,100) |

**4.3 使用Spark计算均值与方差**

第二次MapReduce实验要求大家实现自己的MapReduce框架，然后使用自己的框架来计算均值与方差，接下来我们使用Spark来实现这个功能。

首先，创建一个数据集(1 到1000,000)：

|  |
| --- |
| **2018211009@thumm01:~$ for ((i=1; i<=1000000; i=i+1)); do echo $i >> numbers.txt; done**  **2018211009@thumm01:~$ tail -n 3 numbers.txt**  **999998**  **999999**  **1000000** |

这个数据集的均值为500000.5, 方差为288675.1345946685

接着将这个数据集上传到HDFS:

|  |
| --- |
| **2018211009@thumm01:~$ hadoop fs -put numbers.txt**  **2018211009@thumm01:~$ hadoop fs -tail numbers.txt**  **……**  **999999**  **1000000** |

运行spark-shell, 从HDFS加载number.txt：

|  |
| --- |
| **scala> val numbers = sc.textFile("numbers.txt")**  numbers: org.apache.spark.rdd.RDD[String] = numbers.txt MapPartitionsRDD[1] at textFile at <console>:24 |

加载的数据为字符串形式，需要转成数值型，这里转double型

|  |
| --- |
| **scala> val numbers\_double = numbers.map(num => num.toDouble)**  numbers\_double: org.apache.spark.rdd.RDD[Double] = MapPartitionsRDD[2] at map at <console>:25 |

统计数字个数

|  |
| --- |
| **scala> val n\_num = numbers\_double.count()**  n\_num: Long = 1000000 |

计算均值

|  |
| --- |
| **scala> val mean = numbers\_double.reduce(\_ + \_) / n\_num**  mean: Double = 500000.5 |

计算方差

|  |
| --- |
| **scala> val variance = numbers\_double.map(num => num - mean).map(num => num\*num).reduce(\_ + \_) / n\_num**  variance: Double = 8.333333333359143E10 |

计算标准差

|  |
| --- |
| **scala> import scala.math.\_ #导入scala数学库**  import scala.math.\_  **scala> val std = sqrt(variance)**  std: Double = 288675.1345952599 |

1. 作业提交要求

* 在词频统计任务中，要求大家：
  + 使用自己的数据集, 并将结果下载到本地合并，贴出词频最高的10个单词的统计结果。
  + 换用另一种RDD函数组合实现WordCount功能（越多越好, bonus!），测试不同实现方式的所用的时间。
* 将以上作业所用到的数据、命令、步骤截图等写入实验报告，然后连同所有代码文件一同打包成压缩文件，上交至网络学堂。
* 迟交作业一周以内，以50%比例计分；一周以上不再计分。另一经发现抄袭情况，零分处理。
* 助教联系方式：严浩鹏（[yhp18@mails.tsinghua.edu.cn](mailto:yhp18@mails.tsinghua.edu.cn)）