大数据词频统计实验报告

目录

一、实验目标

1.使用 Hadoop 或者 Spark 对多文件数据实现词频统计

2.Hadoop 分布式文件系统与本地文件系统的运算速度对比

3.spark 在本地模式下不同 CPU 配置的运算速度对比

二、实验设计

1.数据源

为了满足大数据的要求，笔者从 WOS 平台上以“big data”为主题关键词搜索，在结果中下载了 2000 条 txt 格式的文献信息，作为本实验的数据。

2.实验内容

（1） Hadoop 结合 Spark 尝试大数据统计词频

基于应用趋势，该种架构比较流行，笔者特此将两者结合起来进行实验，另外基于 Spark 的运算方式更为友好和高效，有 Python、Java、Scala 三种语言的支持，同时为了学习新的语言使用 Scala 进行编程实验。

（2） 分布式文件系统与本地文件系统的对比

使用 Spark 作为计算工具，将数据分别存储在本地和 HDFS 中，通过完成同样的任务进行比较。其中 Spark 的配置为“pyspark–master local[*]”，即采用本地模式运行，并且使用本地所有的 CPU 核心。

（3） spark 不同配置下的运算速度对比

在本地模式下，设置 CPU 个数分别为 2，4，8 个进行运算速度比较。

3.代码模块设计

（1）多文件路径的遍历和存储模块

取得数据目录下的文件，并将其存储在可迭代的对象中，例如数组等。

（2）数据读取和转换模块

按照路径读取数据，并将所有的数据存储在一个对象中，可通过累加的方式实现。

（3）数据内容统计及存储模块

首先需要通过空格分词，然后使用特殊的类 map 实现对词频的统计。

（4）计时模块

使用 time 类实现，通过开始于结束时间的差值得到运算花费时间。

三、实验流程

1.本机配置信息

关于配置过程，厦门大学林子雨教授的流程已经非常详细，其中有问题或者没有说明的部分将在下文指出。笔者主要参考以下两篇博文：

http://dblab.xmu.edu.cn/blog/install-hadoop/

http://dblab.xmu.edu.cn/blog/1307-2/

（1）虚拟机：VMware-player-15.5.2-15785246.exe

地址：http://download3.vmware.com/software/player/file/VMware-player-15.5.2-15785246.exe

（2）Ubuntu: ubuntu-18.04.4-desktop-amd64.iso

地址：https://releases.ubuntu.com/18.04.4/ubuntu-18.04.4-desktop-amd64.iso

（3）Hadoop: hadoop-2.10.0.tar.gz

地址：http://mirror.bit.edu.cn/apache/hadoop/common/hadoop-2.10.0/hadoop-2.10.0.tar.gz

（4）**Spark:** spark-3.0.0-preview2-bin-without-hadoop.tgz

地址：https://mirrors.tuna.tsinghua.edu.cn/apache/spark/spark-3.0.0-preview2/spark-3.0.0-preview2-bin-without-hadoop.tgz

（5）**Java:** 11.0.7

Linxu 下直接下载，此处可参考：http://dblab.xmu.edu.cn/blog/install-hadoop/

2.配置过程中的问题

（1）linux 使用 vim 或者 vi 时对文件的编辑

a) 打开文件时，初始模式为“replace”,按键盘上的“Insert”即可类似于 word 文件编辑模式。

b) 需要保存和返回时，再按一遍“Insert”返回初始模式，输入“：wq”,表示保存并推出，即可完成。

（2）“Ctrl+shift+c”是 linux 下的复制，粘贴同理

3.数据下载及上传

使用 ubtun 系统中自带的火狐浏览器，前往 WOS 下载数据，将其存储在本地：默认为“Home//Downloads//”,修改为“Home//Downloads//data”文件夹下。同时，启动 Hadoop 服务，将文件夹上传至 HDFS 中备用。

（1）连接服务器

```
ssh localhost
```

（2）启动 Hadoop 服务

```
cd /usr/local/hadoop
./sbin/start-dfs.sh
```

（3）上传数据

```
./bin/hdfs dfs -put /usr/local/spark/mycode/wordcount/data
```

（4）查看数据

```
./bin/hdfs dfs -ls /user/hadoop/data
```

4.spark 配置及 spark-shell 启动

（1）配置 spark

其中 local[4]指的是，使用本地模式并启动 4 个 CPU 进行计算，默认情况使用所有 CPU。

```
cd /usr/local/spark
./bin/pyspark --master local[4]
```

（2）启动 spark-shell

```
./bin/spark-shell
```

5.Scala 编程实现

（1）文件目录的读取

a) 本地文件读取较为简单，直接从绝对路径读取即可，但要注意路径的写法。

```
/*获取文件夹下的所有数据文件路径，返回迭代器*/
    def get_subdirs(dir: File): Iterator[File] = {
          val dir = new File("//user//hadoop//data")
        val d = dir.listFiles.filter(_.isDirectory)
        val f = dir.listFiles.toIterator
        f ++ d.toIterator.flatMap(get_subdirs _)
    }
```

b) HDFS 文件的交互需要设计配置文件系统，才能获得 HDFS 下的路径。

```
/*1.配置 HDFS 文件系统*/
  def getHdfs(path: String) = {
    val conf = new Configuration()
    FileSystem.get(URI.create(path), conf)
  }
  /*2.获取路径*/
  def getFilesAndDirs(path: String): Array[Path] = {
```

```
    val fs = getHdfs(path).listStatus(new Path(path))
    FileUtil.stat2Paths(fs)
  }
  /*3.返回目录下的路径*/
  def listFiles(path: String): Array[Path] = {
    getFilesAndDirs(path).filter(getHdfs(path).getFileStatus(_).isFile())
   }
```

（2）文件数据的读取

为了将其转化为 RDD，首先要有“import scala.io.Source”指令。其次要对返回的结果进行遍历读取。

```
/*读取所有文件的数据*/
    val dir = new Path("t")
    val data = sc.textFile(dir)
    for (e <- file_array){
        val t= e
        data ++ sc.textFile(t.toString)
}
```

（3）数据的统计

该部分，首先将 RDD 数据转变为 map 数据，同时利用空格分隔词汇，将 map 中的频词值相加，最终得到频词统计结果，但是为了检验结果的正确性，又将其及进行排序，输出前五的词汇，最后将统计结果存储在 HDFS 中。

```
scala>     println(topWordCount.take(5).foreach(x=>println(x)))
(7726,0)
(6637,the)
(5760,and)
(5559,of)
(3244,to)
```

图 1 词频前五的词汇

```
/*统计词频*/
    val WordCount = data.
    flatMap(str=>str.split(" ")).
    filter(!_.isEmpty).
    map(word=>(word,1)).
    reduceByKey(_+_)
    /*给数据频词排序并输出*/
    val topWordCount = WordCount.map{case (word, count) => (count,
```

```
word)}.sortByKey(false)
      println(topWordCount.take(5).foreach(x=>println(x)))
      /*存储数据*/

WordCount.saveAsTextFile("hdfs://localhost:9000/user/hadoop/wordcount5")
```

（4）计时模块

```
/*计时器*/
val startTime: Long = System.currentTimeMillis
【中间程序】
      /*运行时间测试*/
      val endTime: Long = System.currentTimeMillis
      System.out.println("程序运行时间：  " + (endTime - startTime) + "ms")
```

6.运算

（1）scala 文件的运行

这里既可以使用 Spark-shell 直接输入，也可以使用 sbt 进行运行，但需要另外的配置，笔者对 sbt 进行了配置,使用独立编程的方式。

（2）本地系统与 HDFS 系统的对比

该部分选取其中一个文件，本地系统和 HDFS 系统分别在 Spark 下进行计算，都采用默认模式。

表 1 不同存储系统处理对比对比

系统	时间(ms)
HDFS	3757
	2806
	2592
Local	2988
	3242
	2487

（3）不同 CPU 配置下的 Spark 运算对比

表 2 不同 CPU 运算速度比较

CPU 数量	时间(ms)
8	1878
	3700
	1846
4	10585
	5014
	2886
2	3929
	2049
	1870

7.运算结果的本地存储

其中“wordcount”为输出路径，“./output”为本地存储路径，注意此时的相对路径。

```
cd /usr/local/hadoop
./bin/hdfs dfs -get wordcount Home./Downloads/wordscout
```

四、实验结果分析

1.HDFS 并未表现出明显的速度

（1）数据量仍然过少

囿于人工下载数据，目前的数据量并没有达到使用该架构能够带来速度提升

的数量级，同时还可能受到随机因素的影响（电脑内部计算进程），出现比本地文件系统速度还要慢的情况，如果要看到实际的差异，仍然需要下载大量的数据。

（2）受伪分布式限制

当前伪分布式是将分布式配置运用到机器上，但在实际计算中仍然使用一台机器计算，并没有进行并行计算，因此分布式的存储系统没有表现出高速的特性，反而在存取数据时可能影响速度。

2.CPU 个数越多未表现出明显的速度提升

（1）虚拟机环境影响

虚拟机是 windows 系统中的一个软件，还有大量的其他软件占用 CPU，其运行速度一般相比真正的服务器系统慢很多，因此 Spark 的配置对于结果的影响并不大。

（2）数据量过小

数据量过小的情况下，虚拟机内部并没有占用大量的计算资源，导致效果不明显。

五、实验心得

1.实践与理论之间差距较大

虽然在课堂上已经听老师讲授了各种架构背后的原理，而且网络上有大量的实践资源，但配置系统和进行实验的过程充满了挑战，如何在大量的资料中快速定位资源和学习是一个非常重要的能力，同时通过实践之后加深了对 Hadoop 和 Spark 的理解，为以后快速上手这两个大数据工具打下了基础。

2.实际问题与工具的匹配度

本实验体现出数据量与大数据架构工具的匹配性，其实这个现象的背后是实际问题与工具的匹配性，所以解决问题需要从问题实际出发，使用恰当的工具解决问题，尤其是面对技术问题时要既符合实际情况，也要符合资源分配合理的要求。