Spark快速入门

Dec 16th, 2014 3:59 pm

Apache Spark是新兴的一种快速通用的大规模数据处理引擎。它的优势有三个方面：

- 通用计算引擎 能够运行MapReduce、数据挖掘、图运算、流式计算、SQL等多种框架；
- 基于内存 数据可缓存在内存中，特别适用于需要迭代多次运算的场景；
- 与Hadoop集成 能够直接读写HDFS中的数据，并能运行在YARN之上。

Spark是用Scala语言编写的，所提供的API也很好地利用了这门语言的特性。它也可以使用Java和Python编写应用。本文将用Scala进行讲解。

安装Spark和SBT

- 从官网上下载编译好的压缩包，解压到一个文件夹中。下载时需注意对应的Hadoop版本，如要读写CDH4 HDFS中的数据，则应下载Pre-built for CDH4这个版本。
- 为了方便起见，可以将spark/bin添加到$PATH环境变量中：

```
1 export SPARK_HOME=/path/to/spark
2 export PATH=$PATH:$SPARK_HOME/bin
```

- 在练习例子时，我们还会用到SBT这个工具，它是用来编译打包Scala项目的。Linux下的安装过程比较简单：
  - 下载sbt-launch.jar到$HOME/bin目录；
  - 新建$HOME/bin/sbt文件，权限设置为755，内容如下：

```
1 SBT_OPTS="-Xms512M -Xmx1536M -Xss1M -XX:+CMSClassUnloadingEnabled -XX:MaxPermSize=256M"
2 java $SBT_OPTS -jar `dirname $0`/sbt-launch.jar "$@"
```

日志分析示例

假设我们有如下格式的日志文件，保存在/tmp/logs.txt文件中：

```
1 2014-12-11 18:33:52  INFO    Java    some message
2 2014-12-11 18:34:33   INFO    MySQL   some message
3 2014-12-11 18:34:54   WARN    Java    some message
4 2014-12-11 18:35:25   WARN    Nginx   some message
5 2014-12-11 18:36:09   INFO    Java    some message
```

每条记录有四个字段，即时间、级别、应用、信息，使用制表符分隔。

Spark提供了一个交互式的命令行工具，可以直接执行Spark查询：

```
1 $ spark-shell
2 Welcome to
3       ____              __
4      / __/__  ___ _____/ /__
5     _\ \/ _ \/ _ `/ __/  '_/
6    /___/ .__/\_,_/_/ /_/\_\   version 1.1.0
7       /_/
8 Spark context available as sc.
9 scala>
```

加载并预览数据

```
1 scala> val lines = sc.textFile("/tmp/logs.txt")
2 lines: org.apache.spark.rdd.RDD[String] = /tmp/logs.txt MappedRDD[1] at textFile at <console>:12
3
4 scala> lines.first()
5 res0: String = 2014-12-11 18:33:52    INFO    Java    some message
```

- sc是一个SparkContext类型的变量，可以认为是Spark的入口，这个对象在spark-shell中已经自动创建了。
- sc.textFile()用于生成一个RDD，并声明该RDD指向的是/tmp/logs.txt文件。RDD可以暂时认为是一个列表，列表中的元素是一行行日志（因此是String类型）。这里的路径也可以是HDFS上的文件，如hdfs://127.0.0.1:8020/user/hadoop/logs.txt。
- lines.first()表示调用RDD提供的一个方法：first()，返回第一行数据。

解析日志

为了能对日志进行筛选，如只处理级别为ERROR的日志，我们需要将每行日志按制表符进行分割：

```
1 scala> val logs = lines.map(line => line.split("\t"))
```

```
2 logs: org.apache.spark.rdd.RDD[Array[String]] = MappedRDD[2] at map at <console>:14
3
4 scala> logs.first()
5 res1: Array[String] = Array(2014-12-11 18:33:52, INFO, Java, some message)
```

- lines.map(f)表示对RDD中的每一个元素使用f函数来处理，并返回一个新的RDD。
- line => line.split(“\t”)是一个匿名函数，又称为Lambda表达式、闭包等。它的作用和普通的函数是一样的，如这个匿名函数的参数是line（String类型），返回值是Array数组类型，因为String.split()函数返回的是数组。
- 同样使用first()方法来看这个RDD的首条记录，可以发现日志已经被拆分成四个元素了。

过滤并计数

我们想要统计错误日志的数量：

```
1 scala> val errors = logs.filter(log => log(1) == "ERROR")
2 errors: org.apache.spark.rdd.RDD[Array[String]] = FilteredRDD[3] at filter at <console>:16
3
4 scala> errors.first()
5 res2: Array[String] = Array(2014-12-11 18:39:42, ERROR, Java, some message)
6
7 scala> errors.count()
8 res3: Long = 158
```

- logs.filter(f)表示筛选出满足函数f的记录，其中函数f需要返回一个布尔值。
- log(1) == “ERROR"表示获取每行日志的第二个元素（即日志级别），并判断是否等于ERROR。
- errors.count()用于返回该RDD中的记录。

缓存

由于我们还会对错误日志做一些处理，为了加快速度，可以将错误日志缓存到内存中，从而省去解析和过滤的过程：

```
1 scala> errors.cache()
```

errors.cache()函数会告知Spark计算完成后将结果保存在内存中。所以说Spark是否缓存结果是需要用户手动触发的。在实际应用中，我们需要迭代处理的往往只是一部分数据，因此很适合放到内存里。

需要注意的是，cache函数并不会立刻执行缓存操作，事实上map、filter等函数都不会立刻执行，而是在用户执行了一些特定操作后才会触发，比如first、count、reduce等。这两类操作分别称为Transformations和Actions。

显示前10条记录

```
1 scala> val firstTenErrors = errors.take(10)
2 firstTenErors: Array[Array[String]] = Array(Array(2014-12-11 18:39:42, ERROR, Java, some message), Array(2014-12-11 18:40:23, ERROR, Nginx, some message), ..
3
4 scala> firstTenErrors.map(log => log.mkString("\t")).foreach(line => println(line))
5 2014-12-11 18:39:42    ERROR   Java    some message
6 2014-12-11 18:40:23    ERROR   Nginx   some message
7 ...
```

errors.take(n)方法可用于返回RDD前N条记录，它的返回值是一个数组。之后对firstTenErrors的处理使用的是Scala集合类库中的方法，如map、foreach，和RDD提供的接口基本一致。所以说用Scala编写Spark程序是最自然的。

按应用进行统计

我们想要知道错误日志中有几条Java、几条Nginx，这和常见的Wordcount思路是一样的。

```
1  scala> val apps = errors.map(log => (log(2), 1))
2  apps: org.apache.spark.rdd.RDD[(String, Int)] = MappedRDD[15] at map at <console>:18
3
4  scala> apps.first()
5  res20: (String, Int) = (Java,1)
6
7  scala> val counts = apps.reduceByKey((a, b) => a + b)
8  counts: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[17] at reduceByKey at <console>:20
9
10 scala> counts.foreach(t => println(t))
11 (Java,58)
12 (Nginx,53)
13 (MySQL,47)
```

errors.map(log => (log(2), 1))用于将每条日志转换为键值对，键是应用（Java、Nginx等），值是1，如("Java", 1)，这种数据结构在Scala中称为元组（Tuple），这里它有两个元素，因此称为二元组。

对于数据类型是二元组的RDD，Spark提供了额外的方法，reduceByKey(f)就是其中之一。它的作用是按键进行分组，然后对同一个键下的所有值使用f函数进行归约（reduce）。归约的过程是：使用列表中第一、第二个元素进行计算，然后用结果和第三元素进行计算，直至列表耗尽。如：

```
1 scala> Array(1, 2, 3, 4).reduce((a, b) => a + b)
2 res23: Int = 10
```

上述代码的计算过程即((1 + 2) + 3) + 4。

counts.foreach(f)表示遍历RDD中的每条记录，并应用f函数。这里的f函数是一条打印语句（println）。

打包应用程序

为了让我们的日志分析程序能够在集群上运行，我们需要创建一个Scala项目。项目的大致结构是：

```
spark-sandbox
├── build.sbt
├── project
│   ├── build.properties
│   └── plugins.sbt
└── src
    └── main
        └── scala
            └── LogMining.scala
```

你可以直接使用这个项目作为模板。下面说明一些关键部分：

配置依赖

build.sbt

```
libraryDependencies += "org.apache.spark" %% "spark-core" % "1.1.1"
```

程序内容

src/main/scala/LogMining.scala

```
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark.SparkConf

object LogMining extends App {
  val conf = new SparkConf().setAppName("LogMining")
  val sc = new SparkContext(conf)
  val inputFile = args(0)
  val lines = sc.textFile(inputFile)
  // 解析日志
  val logs = lines.map(_.split("\t"))
  val errors = logs.filter(_(1) == "ERROR")
  // 缓存错误日志
  errors.cache()
  // 统计错误日志记录数
  println(errors.count())
  // 获取前10条MySQL的错误日志
  val mysqlErrors = errors.filter(_(2) == "MySQL")
  mysqlErrors.take(10).map(_ mkString "\t").foreach(println)
  // 统计每个应用的错误日志数
  val errorApps = errors.map(_(2) -> 1)
  errorApps.countByKey().foreach(println)
}
```

打包运行

```
$ cd spark-sandbox
$ sbt package
$ spark-submit --class LogMining --master local target/scala-2.10/spark-sandbox_2.10-0.1.0.jar data/logs.txt
```

参考资料

- Spark Programming Guide
- Introduction to Spark Developer Training
- Spark Runtime Internals

Posted by Ji ZHANG Dec 16th, 2014 3:59 pm big data, tutorial