**Spark 编程示例**

目录

[1.一些概念 1](#_Toc432172100)

[2. 如何编程 1](#_Toc432172101)

[3．弹性分布式数据集 2](#_Toc432172102)

[4 分析nginx日志中状态码出现次数 3](#_Toc432172103)

[5 参考文章 4](#_Toc432172104)

# 1.一些概念

每一个spark 的应用，都是由一个驱动程序构成，它运行用户的main 函数，在一个集群上执行各种各样的并行操作。

Spark 提出的最主要抽象概念是RDD（弹性分布式数据集），它是一个有容错机制（划分到集群的各个节点上）并可以被并行操作的元素集合。

两种类型的RDD:

并行集合：接受一个已存在的scala集合，然后进行各种并行计算。

外部数据集：外部存储系统，例如文件系统、HDFS、HBase以及任何支持Hadoop InputFormat 的数据源。

这两种类型的RDD都可以通过相同的方式进行操作，用户可以让spark 保留一个RDD在内存中，使其能在并行操作中被有效的重复使用，并且，RDD能自动从节点故障中恢复。

Spark 的第二个抽象概念是共享变量，可以在并行操作中使用，在默认情况下，spark通过不同节点上的一系列任务来运行一个函数，它将每一个函数中用到的变量拷贝传递到每一个任务中，有时候，一个变量需要在任务之间，或任务与驱动程序之间被共享。

Spark支持两种类型的共享变量：广播变量，可以在内存的所有的节点上缓存变量；累加器：只能用于做加法的变量，例如计数或求和。

# 2. 如何编程

初始化spark

在一个spark程序中要做的第一件事就是创建一个SparkContext对象来告诉spark如何连接一个集群，为了创建SparkContext，你首先需要创建一个SparkConf对象，这个对象会包含你的应用的一些相关信息。这个通常是通过下面的构造器来实现的：

new SparkContext(master,appName,[sparkHome],[jars])

参数说明：

master:用于指定所连接的Spark或Mesos集群的URL。

appName:应用的名称，将会在集群的web监控UI中显示。

sparkHome:可选，spark的安装路径

jars：可选，在本地机器上的JAR文件列表，其中包括你应用的代码以及任何的依赖，Spark 将会把他们部署到所有的集群结点上。

在 python 中初始化，示例代码如下：

conf = SparkConf().setAppName("Hello Spark").setMaster("local")

sc = SparkContext(conf=conf)

如果你在一个集群上运行 spark-shell，则 master 参数默认为 local。在实际使用中，当你在集群中运行你的程序，你一般不会把 master 参数写死在代码中，而是通过用 spark-submit 运行程序来获得这个参数。但是，在本地测试以及单元测试时，你仍需要自行传入 local 来运行Spark程序。

运行代码

运行代码有几种方式，一是通过 spark-shell 来运行 scala 代码，一是编写 java 代码并打成包以 spark on yarn 方式运行，还有一种是通过 PySpark 来运行 python 代码。

在 spark-shell 和 PySpark 命令行中，一个特殊的集成在解释器里的 SparkContext 变量已经建立好了，变量名叫做 sc，创建你自己的 SparkContext 不会起作用。

# 3．弹性分布式数据集

3.1 并行集合

并行集合是通过调用SparkContext的parallelize 方法，在一个已经存在的scala集合上创建一个seq对象。

Parallelize方法还可以接受一个参数slices，表示数据集切分的份数，spark将会在集群上为每一份数据起一个任务，比如，你可以在集群的每个cpu上分布2-4个slices，一般来说，spark 会尝试根据集群的状况，来自动设定slices的数目，当然，你也可以手动设置。

Python 示例程序：

data = [1, 2, 3, 4, 5]

distData = sc.parallelize(data)

distData.reduce(lambda a, b: a + b)

3.2 外部数据源

Spark可以从存储在 HDFS，或者 Hadoop 支持的其它文件系统（包括本地文件， HBase 等等）上的文件创建分布式数据集。Spark 支持TextFile、SequenceFiles 以及其他任何 Hadoop InputFormat 格式的输入。

TextFile 的 RDD 可以通过下面方式创建，该方法接受一个文件的 URI 地址，该地址可以是本地路径，或者 hdfs://、s3n:// 等 URL 地址

distFile = sc.textFile("data.txt")

除了文本文件之外，Spark 还支持其他格式的输入：

* SparkContext.wholeTextFiles 方法可以读取一个包含多个小文件的目录，并以 filename，content 键值对的方式返回结果。
* 对于 SequenceFiles，可以使用 SparkContext 的 sequenceFile[K, V]` 方法创建。像 IntWritable 和 Text 一样，它们必须是 Hadoop 的 Writable 接口的子类。另外，对于几种通用 Writable 类型，Spark 允许你指定原生类型来替代。例如：sequencFile[Int, String] 将会自动读取 IntWritable 和 Texts。
* 对于其他类型的 Hadoop 输入格式，你可以使用 SparkContext.hadoopRDD 方法，它可以接收任意类型的 JobConf 和输入格式类，键类型和值类型。按照像 Hadoop 作业一样的方法设置输入源就可以了。
* RDD.saveAsObjectFile 和 SparkContext.objectFile 提供了以 Java 序列化的简单方式来保存 RDD。虽然这种方式没有 Avro 高效，但也是一种简单的方式来保存任意的 RDD。

3.3 RDD 操作

RDD支持两类操作

 转化操作，用于从已有的数据集转化产生新的数据集

 启动操作，用于在计算结束后向驱动程序返回结果。

Python 示例：

lines = sc.textFile("data.txt")

lineLengths = lines.map(lambda s: len(s))

totalLength = lineLengths.reduce(lambda a, b: a + b)

第一行定义了一个由外部文件产生的基本RDD。这个数据集不是从内存中载入的也不是由其他操作产生的；lines仅仅是一个指向文件的指针。第二行将lineLengths定义为map操作的结果。再强调一次，由于惰性求值的缘故，lineLengths并不会被立即计算得到。最后，我们运行了reduce操作，这是一个启动操作。从这个操作开始，Spark将计算过程划分成许多任务并在多机上运行，每台机器运行自己部分的map操作和reduce操作，最终将自己部分的运算结果返回给驱动程序。

如果我们希望以后重复使用lineLengths，只需在reduce前加入下面这行代码：

lineLengths.persist()

这条代码将使得 lineLengths 在第一次计算生成之后保存在内存中。

# 4 分析nginx日志中状态码出现次数

先将测试数据上传到 hdfs:

hadoop fs -put access.log

然后，编写一个 python 文件，保存为 SimpleApp.py：

from pyspark import SparkContext

logFile = "access.log"

sc = SparkContext("local", "Simple App")

rdd = sc.textFile(logFile).cache()

counts = rdd.map(lambda line: line.split()[8]).map(lambda word: (word, 1)).reduceByKey(lambda a, b: a + b).sortByKey(lambda x: x)

# This is just a demo on how to bring all the sorted data back to a single node.

# In reality, we wouldn't want to collect all the data to the driver node.

output = counts.collect()

for (word, count) in output:

 print "%s: %i" % (word, count)

counts.saveAsTextFile("/data/result")

sc.stop()

接下来，运行下面代码：

spark-submit --master local[4] SimpleApp.py

运行成功之后，你会在终端看到以下输出：

200: 682

301: 7

304: 10

404: 125

# 5 参考文章

Spark编程指南