第七讲 Spark编程

徐 辰
cxu@dase.ecnu.edu.cn
華東師範大學

大纲

2

- 编程方式
  - Scala Shell
  - Java/Scala IDE
- 编程接口
- 编程实例

Spark-shell

3

执行如下命令启动Spark Shell（默认是local模式）：

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

启动Spark Shell成功后在输出信息的末尾可以看到 “Scala >”的命令提示符

```
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.1.0
      /_/

Using Scala version 2.11.8 (OpenJDK 64-Bit Server VM, Java 1.8.0_121)
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```

一个Driver就包括main方法和分布式数据集
Spark Shell本身就是一个Driver，里面已经包含了main方法

在spark-shell中运行代码

4

spark-shell命令及其常用的参数如下：

```
./bin/spark-shell --master <master-url>
```

Spark的运行模式取决于传递给SparkContext的Master URL的值。Master URL可以是以下任一种形式：

* local 使用一个Worker线程本地化运行SPARK(完全不并行)
* local[*] 使用逻辑CPU个数数量的线程来本地化运行Spark
* local[K] 使用K个Worker线程本地化运行Spark（理想情况下，K应该根据运行机器的CPU核数设定）
* spark://HOST:PORT 连接到指定的Spark standalone master。默认端口是7077
* yarn-client 以客户端模式连接YARN集群。集群的位置可以在HADOOP_CONF_DIR 环境变量中找到
* yarn-cluster 以集群模式连接YARN集群。集群的位置可以在HADOOP_CONF_DIR 环境变量中找到
* mesos://HOST:PORT 连接到指定的Mesos集群。默认接口是5050

在spark-shell中运行代码

5

在Spark中采用本地模式启动Spark Shell的命令主要包含以下参数：
--master：这个参数表示当前的Spark Shell要连接到哪个master，如果是local[*]，就是使用本地模式启动spark-shell，其中，中括号内的星号表示需要使用几个CPU核心(core)，也就是启动几个线程模拟Spark集群
--jars： 这个参数用于把相关的JAR包添加到CLASSPATH中；如果有多个jar包，可以使用逗号分隔符连接它们

在spark-shell中运行代码

6

可以在里面输入scala代码进行调试：

```
scala> 8*2+5
res0: Int = 21
```

可以使用命令 “:quit”退出Spark Shell：

```
scala>:quit
```

或者，也可以直接使用 “Ctrl+D”组合键，退出Spark Shell

大纲

7

- 编程方式
  - Scala Shell
  - Java/Scala IDE
- 编程接口
- 编程实例

应用程序编写流程

8

- 在IDE中编写Java/Scala程序
- 利用IDE打包jar
- 命令行提交jar包

```
bin/spark-submit --class "SimpleApp" ~/sparkapp/target/scala-2.10/simple-project_2.10-1.0.jar
```

大纲

9

- 编程方式
- 编程接口
  - RDD
  - Spark SQL
- 编程实例

Spark程序结构（Java）

10

```
package X;
import org.apache.spark.SparkContext
import org.apache.spark.api.java.JavaSparkConf
public class Y {
  public static void main(String[] args) {
    SparkConf conf = new SparkConf()
    JavaSparkContext sc = new JavaSparkContext(conf)
    sc.Z …
    sc.close()
  }
}
```

导入Spark所需的包

函数入口

初始化SparkContext

RDD操作

关闭SparkContext

Spark程序结构（Scala）

11

```
package X;
import org.apache.spark.SparkContext
import org.apache.spark.SparkConf
object Y {
  def main(args: Array[String]) {
    val conf = new SparkConf()
    val sc = new SparkContext(conf)
    sc.Z …
    sc.stop()
  }
}
```

导入Spark所需的包

函数入口

初始化SparkContext

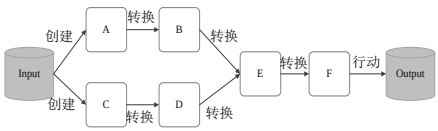
RDD操作

关闭SparkContext

RDD操作

12

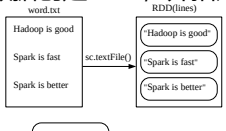
- RDD创建
- RDD转换
- RDD行动
- RDD保存



RDD创建

13

- 从文件系统中加载数据创建RDD，并指定分区的个数
  - 本地文件系统
  - HDFS
  - 其它




```
scala> val lines = sc.textFile("file:///usr/local/spark/mycode/rdd/word.txt")
lines: org.apache.spark.rdd.RDD[String] =
file:///usr/local/spark/mycode/rdd/word.txt MapPartitionsRDD[12] at textFile
at <console>:27
```

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

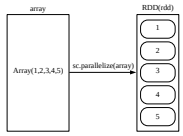
RDD创建

14

- 通过并行集合（数组）创建RDD

可以调用SparkContext的parallelize方法，在Driver中一个已经存在的集合（数组）上创建。

```
scala>val array = Array(1,2,3,4,5)
array: Array[Int] = Array(1, 2, 3, 4, 5)
scala>val rdd = sc.parallelize(array)
rdd: org.apache.spark.rdd.RDD[Int] =
ParallelCollectionRDD[13] at
parallelize at <console>:29
```



或者，也可以从列表中创建：

```
scala>val list = List(1,2,3,4,5)
list: List[Int] = List(1, 2, 3, 4, 5)
scala>val rdd = sc.parallelize(list)
rdd: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[14] at
parallelize at <console>:29
```

设置RDD分区

15

- 创建RDD时手动指定分区个数

在调用textFile()和parallelize()方法的时候手动指定分区个数即可，语法格式如下：

sc.textFile(path, partitionNum)

其中，path参数用于指定要加载的文件的地址，partitionNum参数用于指定分区个数。

```
scala> val  array = Array(1,2,3,4,5)
scala> val  rdd = sc.parallelize(array,2)  //设置两个分区
```

常用的RDD转换

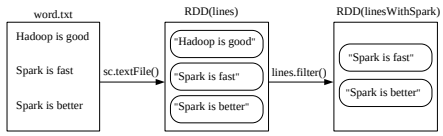
16

操作	含义
filter(func)	筛选出满足函数func的元素，并返回一个新的数据集
map(func)	将每个元素传递到函数func中，并将结果返回为一个新的数据集
flatMap(func)	与map()相似，但每个输入元素都可以映射到0或多个输出结果
groupByKey()	应用于(K,V)键值对的数据集时，返回一个新的(K, Iterable)形式的数据集
reduceByKey(func)	应用于(K,V)键值对的数据集时，返回一个新的(K, V)形式的数据集，其中每个值是将每个key传递到函数func中进行聚合后的结果
sortByKey()	返回一个根据键排序的RDD
join()	对于内连接，对于给定的两个输入数据集(K,V1)和(K,V2)，只有在两个数据集中都存在的key才会被输出，最终得到一个(K,(V1,V2))类型的数据集

filter(func)

17

```
scala>  val  lines =sc.textFile(file:///usr/local/spark/mycode/rdd/word.txt)
scala>  val  linesWithSpark=lines.filter(line => line.contains("Spark"))
```

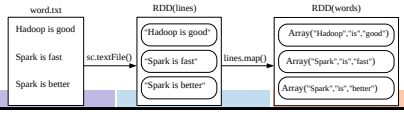


map(func)

18

```
scala> data=Array(1,2,3,4,5)
scala> val  rdd1= sc.parallelize(data)
scala> val  rdd2=rdd1.map(x=>x+10)
```

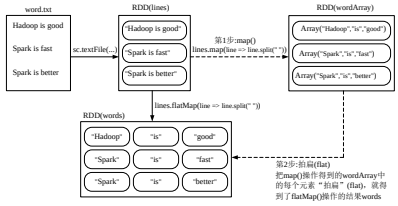

```
scala> val  lines = sc.textFile("file:///usr/local/spark/mycode/rdd/word.txt")
scala> val  words=lines.map(line => line.split(" "))
```



flatMap(func)

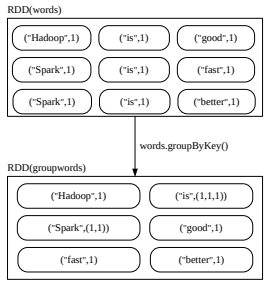
19

```
scala> val  lines = sc.textFile("file:///usr/local/spark/mycode/rdd/word.txt")
scala> val  words=lines.flatMap(line => line.split(" "))
```



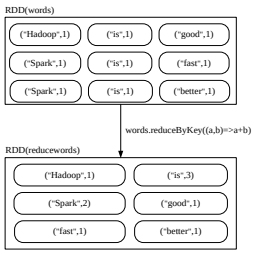
groupByKey()

20



reduceByKey(func)

21



sortByKey()

22

sortByKey()的功能是返回一个根据键排序的RDD

(Hadoop,1)
(Spark,1)
(Hive,1)
(Spark,1)

```
scala> pairRDD.sortByKey()
res0: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[2] at
sortByKey at <console>:34
scala> pairRDD.sortByKey().foreach(println)
(Hadoop,1)
(Hive,1)
(Spark,1)
(Spark,1)
```

join()

23

join就表示内连接。对于内连接，对于给定的两个输入数据集(K,V1)和(K,V2)，只有在两个数据集中都存在的key才会被输出，最终得到一个(K,(V1,V2))类型的数据集。

```
scala> val pairRDD1 = sc.parallelize(Array(("spark",1),("spark",2),("hadoop",3),("hadoop",5)))
pairRDD1: org.apache.spark.rdd.RDD[(String, Int)] = ParallelCollectionRDD[24] at parallelize at
<console>:27

scala> val pairRDD2 = sc.parallelize(Array(("spark","fast")))
pairRDD2: org.apache.spark.rdd.RDD[(String, String)] = ParallelCollectionRDD[25] at parallelize at
<console>:27

scala> pairRDD1.join(pairRDD2)
res9: org.apache.spark.rdd.RDD[(String, (Int, String))] = MapPartitionsRDD[28] at join at <console>:32

scala> pairRDD1.join(pairRDD2).foreach(println)
(spark,(1,fast))
(spark,(2,fast))
```

RDD Repartition

24

- 通过转换操作得到新 RDD 时，直接调用 repartition 方法或自定义分区方法

```
scala> val  data = sc.textFile("file:///usr/local/spark/mycode/rdd/word.txt",2)
data: org.apache.spark.rdd.RDD[String] =
file:///usr/local/spark/mycode/rdd/word.txt MapPartitionsRDD[12] at textFile at
<console>:24
scala> data.partitions.size  //显示data这个RDD的分区数量
res2: Int=2
scala> val  rdd = data.repartition(1)  //对data这个RDD进行重新分区
rdd: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[11] at repartition
at :26
scala> rdd.partitions.size
res4: Int = 1
```

- 问题：什么时候会用该方法?

RDD行动操作

25

- 惰性机制：整个转换过程只是记录了转换的轨迹，并不会发生真正的计算，只有遇到行动操作时，才会发生真正的计算，开始从血缘关系源头开始，进行物理的转换操作

操作	含义
count()	返回数据集中的元素个数
collect()	以数组的形式返回数据集中的所有元素
first()	返回数据集中的第一个元素
take(n)	以数组的形式返回数据集中的前n个元素
reduce(func)	通过函数func（输入两个参数并返回一个值）聚合数据集中的元素
foreach(func)	将数据集中的每个元素传递到函数func中运行

RDD行动操作

26

```
scala> val rdd=sc.parallelize(Array(1,2,3,4,5))
rdd: org.apache.spark.rdd.RDD[Int]=ParallelCollectionRDD[1] at parallelize
at <console>:24
scala> rdd.count()
res0: Long = 5
scala> rdd.first()
res1: Int = 1
scala> rdd.take(3)
res2: Array[Int] = Array(1,2,3)
scala> rdd.reduce((a,b)=>a+b)
res3: Int = 15
scala> rdd.collect()
res4: Array[Int] = Array(1,2,3,4,5)
scala> rdd.foreach(elem=>println(elem))
1
2
3
4
5
```

RDD保存

27

- RDD写入到本地文本文件

```
scala> val textFile = sc.
| textFile("file:///usr/local/spark/mycode/wordcount/word.txt")
scala> textFile.
| saveAsTextFile("file:///usr/local/spark/mycode/wordcount/writeback")
```

- RDD中的数据保存到HDFS文件中

```
scala> textFile.saveAsTextFile("writeback")
```

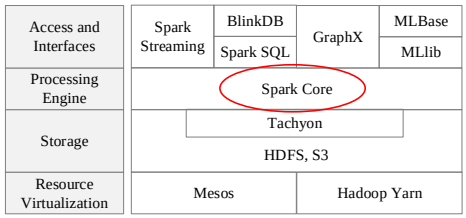
大纲

28

- 编程方式
- 编程接口
  - RDD
  - Spark SQL
- 编程实例

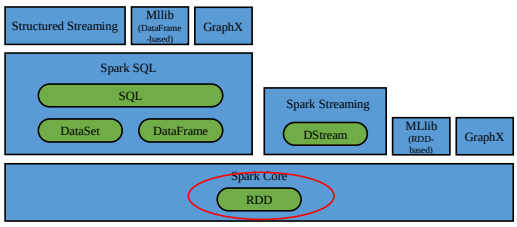
Spark软件栈

29



Spark API

30



大纲

31

- 编程方式
- 编程接口
- 编程实例
  - WordCount
  - PageRank
  - K-Means
  - Join及其优化
  - Checkpoint

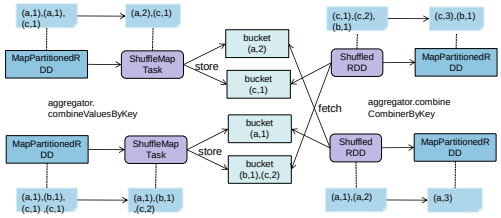
WordCount in Java

32

```
JavaRDD<String> textFile = sc.textFile("hdfs://...");
JavaPairRDD<String, Integer> counts = textFile
    .flatMap(s -> Arrays.asList(s.split(" ")).iterator())
    .mapToPair(word -> new Tuple2<>(word, 1))
    .reduceByKey((a, b) -> a + b);
counts.saveAsTextFile("hdfs://...");
```

reduceByKey执行示意

33



大纲

34

- 编程方式
- 编程接口
- 编程实例
  - WordCount
  - PageRank
  - K-Means
  - Join及其优化
  - Checkpoint

Basic Idea

35

Give pages ranks (scores) based on links to them

- Links from many pages ➔ high rank
- Link from a high-rank page ➔ high rank

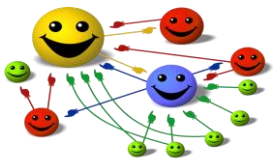
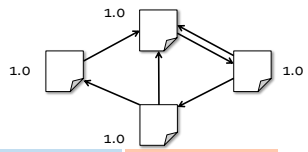


Image: en.wikipedia.org/wiki/File:PageRank-hi-res-2.png

Algorithm

36

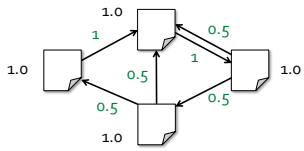
1. Start each page at a rank of 1
2. On each iteration, have page p contribute $rank_p$ / $|neighbors_p|$ to its neighbors
3. Set each page's rank to 0.15 + 0.85 × contribs



Algorithm

37

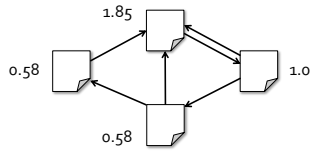
1. Start each page at a rank of 1
2. On each iteration, have page p contribute $rank_p$ / $|neighbors_p|$ to its neighbors
3. Set each page's rank to 0.15 + 0.85 × contribs



Algorithm

38

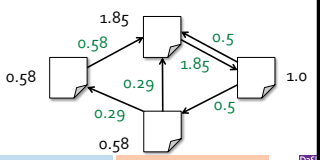
1. Start each page at a rank of 1
2. On each iteration, have page p contribute $rank_p$ / $|neighbors_p|$ to its neighbors
3. Set each page's rank to 0.15 + 0.85 × contribs



Algorithm

39

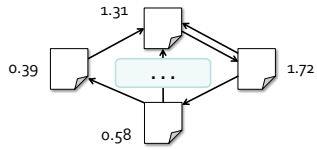
1. Start each page at a rank of 1
2. On each iteration, have page p contribute $rank_p$ / $|neighbors_p|$ to its neighbors
3. Set each page's rank to 0.15 + 0.85 × contribs



Algorithm

40

1. Start each page at a rank of 1
2. On each iteration, have page p contribute $rank_p$ / $|neighbors_p|$ to its neighbors
3. Set each page's rank to 0.15 + 0.85 × contribs

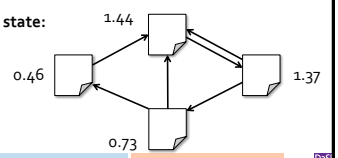


Algorithm

41

1. Start each page at a rank of 1
2. On each iteration, have page p contribute $rank_p$ / $|neighbors_p|$ to its neighbors
3. Set each page's rank to 0.15 + 0.85 × contribs

Final state:



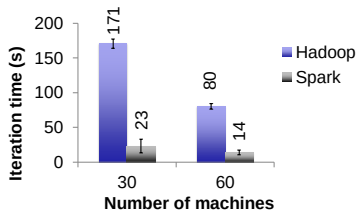
Scala Implementation

42

```
val links = // load RDD of (url, neighbors) pairs
var ranks = // load RDD of (url, rank) pairs

for (i <- 1 to ITERATIONS) {
  val contribs = links.join(ranks).flatMap {
    case (url, (links, rank)) =>
      links.map(dest => (dest, rank/links.size))
  }
  ranks = contribs.reduceByKey(_ + _)
                  .mapValues(0.15 + 0.85 * _)
}
ranks.saveAsTextFile(...)
```

PageRank Performance

43



大纲

44

- 编程方式
- 编程接口
- 编程实例
  - WordCount
  - PageRank
  - K-Means
  - Join及其优化
  - Checkpoint

K-Means代码框架

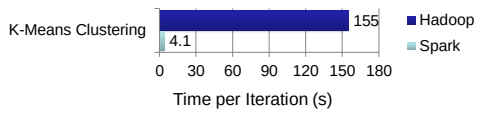
45

```
point //数据点
kpoint //初始中心点

while(tempDist > convergeDist) {
    …
   newPoints
    …
   tempDist = 0.0
   for (mapping <- newPoints) {
     tempDist += squaredDistance(kPoints(mapping._1), mapping._2)
   }
   …
}
```

https://github.com/apache/spark/blob/master/examples/src/main/scala/org/apache/spark/examples/LocalKMeans.scala

K-Means性能

46



大纲

47

- 编程方式
- 编程接口
- 编程实例
  - WordCount
  - PageRank
  - K-Means
  - Join及其优化
  - Checkpoint

Scala Implementation

48

```
val links = // load RDD of (url, neighbors) pairs
var ranks = // load RDD of (url, rank) pairs

Val result = links.join(ranks)
```

Join优化

49

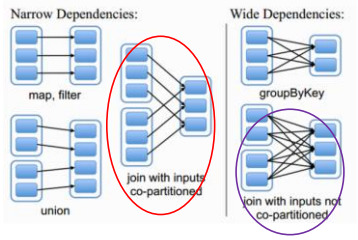
- 假如

```
val links = // load RDD of (url, neighbors) pairs
var ranks = // load RDD of (url, rank) pairs

val rankBroadCast = sc.broadcast(rank)

links.map(r => {
 val rankBroadCastValue = rankBroadCast.value
 if (rankBroadCastValue.contains(r._3)) {
 // 获取rankBroadCastValue中对应key的value
 val left = rankBroadCastValue.get(r._3).
            get(left._1, left._2, r._2)
 }
 else
   null
})
```

Join优化

50

- Join输入的partition不同将决定是否需要shuffle



大纲

51

- 编程方式
- 编程接口
- 编程实例
  - WordCount
  - PageRank
  - K-Means
  - Join及其优化
  - Checkpoint

Scala Implementation

52

```
val links = // load RDD of (url, neighbors) pairs
var ranks = // load RDD of (url, rank) pairs

Val result = links.join(ranks)

result.checkpoint("hdfs://xxx")

result.map(xxx).reduce(xxx)
```

课后阅读

53

- 论文
  - 吴信东, 嵇圣硒. MapReduce与Spark用于大数据分析之比较. 软件学报 2018, 29(6): 1770−1791
  - Armbrust, M., Ghodsi, A., Zaharia, M., Xin, R. S., Lian, C., Huai, Y., … Franklin, M. J. (2015). Spark SQL: Relational Data Processing in Spark. In *SIGMOD Conference* (pp. 1383–1394).（研究生）

本讲小节

54

- 编程方式
- 编程接口
- 编程实例

谢谢！Q&A