基于上网日志分析的Spark 练习手册

九鼎定制版

1. Spark Shell/Spark submit操作练习 3

1.1 练习目的： 3

1.2 练习案例说明： 3

1.3 练习步骤: 3

2. Spark 编程操作练习 5

2.1 练习目的： 5

2.2 练习案例说明： 5

2.3 练习步骤: 5

6. Spark 在不同集群的练习 8

3.1 练习目的： 8

3.2 练习案例说明： 9

3.3 练习步骤: 9

7. Spark SQL练习 9

5.1 练习目的： 9

5.2 练习案例说明： 9

5.3 练习步骤: 9

8. Spark SQL Thrift练习 12

6.1 练习目的： 12

6.2 练习案例说明： 12

6.3 练习步骤: 12

9. Spark Streaming 基础开发练习 13

7.1 练习目的： 13

7.2 练习案例说明： 13

7.3 练习步骤: 13

# Spark Shell/Spark submit操作练习

## 1.1 练习目的：

通过本练习，学员可以掌握Spark-shell,Spark-submit 基本使用。

## 1.2 练习案例说明：

## 1.3 练习步骤:

1. WordCount练习
   1. 进入Spark-Shell
      1. spark-shell --master spark://localhost:7077 --executor-memory 128m --driver-memory 128m
      2. spark-shell --master spark://localhost:7077 --executor-memory 1g --driver-memory 128m

val rdd=sc.textFile("hdfs://localhost:9000/user/hadoop/testdata/core-site.xml")

val rdd = sc.textFile("hdfs://localhost:9000/testspark.txt")

rdd.cache()

val wordcount=rdd.flatMap(\_.split(" ")).map(x=>(x,1)).reduceByKey(\_+\_)

<hadoop,1> -> <hadoop,<1,1,1>

<hadoop,1>

<hadoop,1>

<hbase,1> <hbase,1>

<ozzie,1> <ozzie,1>

<hadoop,hadoop,hadoop,hadoop

hadoop,hadoop,hadoop>

🡺 <hadoop,1>

<hadoop,1>

<hadoop,1>

<hadoop,1>

wordcount.take(10)

val wordsort=wordcount.map(x=>(x.\_2,x.\_1)).sortByKey(false).map(x=>(x.\_2,x.\_1))

wordsort.take(10)

通过 http://localhost:8080 查看 Spark 运行情况

1. 使用 Spark自带的例子Spark-submit 测试
   1. ./spark-submit --master spark://localhost:7077 --class org.apache.spark.examples.SparkPi --executor-memory 256m ../lib/spark-examples-1.1.0-hadoop2.2.0.jar 2

* 参数说明(详细可以参考上面的参数说明):
* --master Master 所在地址,可以有 Mesos、Spark、YARN 和 Local 四种,在这里为 Spark Standalone 集群,地址为 spark://hadoop1:7077
* --class 应用程序调用的类名,这里为 org.apache.spark.examples.SparkPi
* --executor-memory 每个 executor 所分配的内存大小,这里为 512M
* 执行 jar 包 这里是../lib/spark-examples-1.1.0-hadoop2.2.0.jar
* 分片数目 这里数目为 200

1. 使用 Spark自带的例子Spark-submit 测试二

* ./spark-submit --master spark://localhost:7077 --class org.apache.spark.examples.SparkPi --executor-memory 256m --total-executor-cores 2 ../lib/spark-examples-1.1.0-hadoop2.2.0.jar 200

参数说明(详细可以参考上面的参数说明):

* --master Master 所在地址,可以有 Mesos、Spark、YARN 和 Local 四种,在这里为 Spark Standalone 集群,地址为 spark://hadoop1:7077
* --class 应用程序调用的类名,这里为 org.apache.spark.examples.SparkPi
* --executor-memory 每个 executor 所分配的内存大小,这里为 512M
* --total-executor-cores 2 每个 executor 分配的内核数
* 执行 jar 包 这里是../lib/spark-examples-1.1.0-hadoop2.2.0.jar
* 分片数目 这里数目为 200

# Spark 编程操作练习

## 2.1 练习目的：

通过本练习，学员可以掌握在Spark中编写程序的方法

## 2.2 练习案例说明：

参考Action等函数说明

## 2.3 练习步骤:

1. 并行化集合例子演示

在该例子中通过 parallelize 方法定义了一个从 1~10 的数据集,然后通过 map(\_\*2)对数据 集每个数乘以 2,接着通过 filter(\_%3==0)过滤被 3 整除的数字,最后使用 toDebugString 显 示 RDD 的 LineAge,并通过 collect 计算出最终的结果

val num=sc.parallelize(1 to 10)

val doublenum = num.map(\_\*2)

val threenum = doublenum.filter(\_ % 3 == 0) threenum.toDebugString

threenum.collect

num.reduce (\_ + \_)

num.take(5)

num.first

num.count

num.take(5).foreach(println)

1. Shuffle 操作例子演示

val kv1=sc.parallelize(List(("A",1),("B",2),("C",3),("A",4),("B",5)))

kv1.sortByKey().collect

kv1.groupByKey().collect

kv1.reduceByKey(\_+\_).collect

val kv2=sc.parallelize(List(("A",4),("A",4),("C",3),("A",4),("B",5)))

kv2.distinct.collect

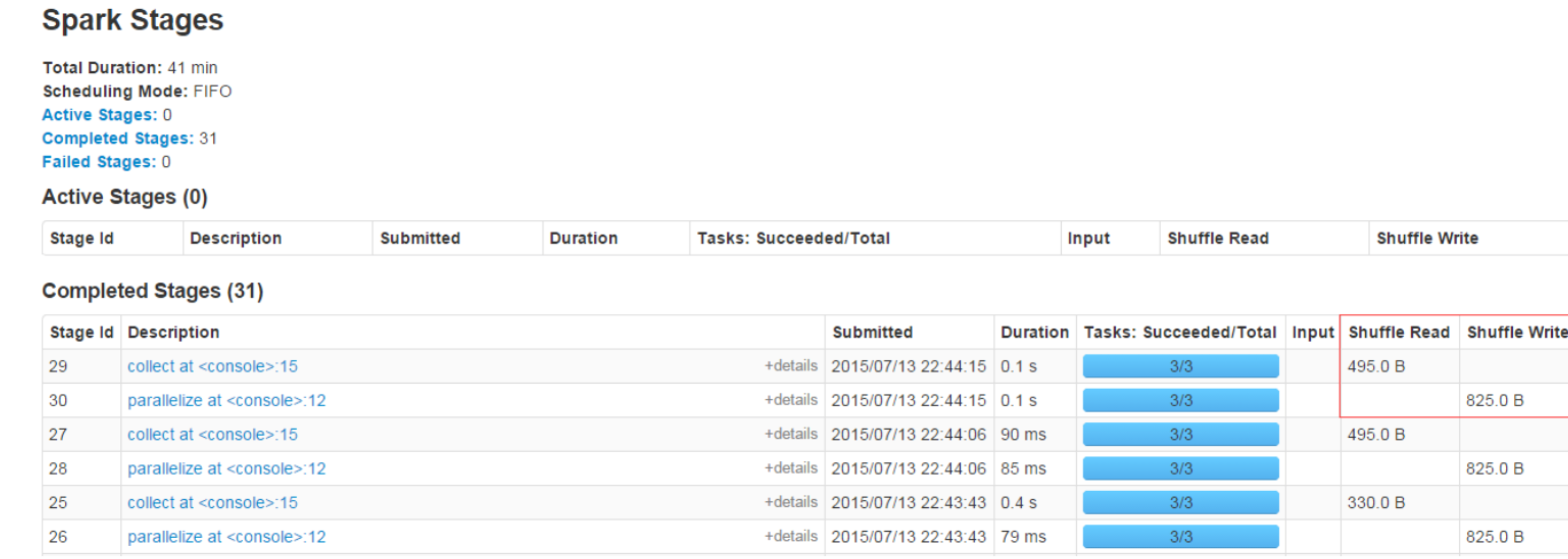
kv1.union(kv2).collect

val kv3=sc.parallelize(List(("A",10),("B",20),("D",30)))

kv1.join(kv3).collect

kv1.cogroup(kv3).collect

­­



1. 文件例子读取

在 该 步 骤 中 RDD 的 变 换 过 程 为 : HadoopRDD->MappedRDD-> FlatMappedRDD->MappedRDD->PairRDDFunctions->ShuffleRDD->MapPartitionsRD D

val text = sc.textFile("hdfs://hadoop1:9000/class3/directory/")

text.toDebugString

val words=text.flatMap(\_.split(" "))

val wordscount=words.map(x=>(x,1)).reduceByKey(\_+\_)

wordscount.toDebugString

wordscount.collect

RDD 类型的变化过程如下:

* 首先使用 textFile()读取 HDFS 数据形成 MappedRDD,这里有可能有疑问,从 HDFS 读取的数据不是 HadoopRDD,怎么变成了 MappedRDD。回答这个问题需要从 Spark 源码进 行分析,在 sparkContext 类中的 textFile()方法读取 HDFS 文件后,使用了 map()生成了 MappedRDD
* 然后使用 flatMap()方法对文件内容按照空格拆分单词,拆分形成 FlatMappedRDD
* 其次使用 map(x=>(x(1),1))对上步骤拆分的单词形成(单词,1)数据对,此时生成的

MappedRDD,最后 使用 reduceByKey()方法对单词的频度统计,由此生成 ShuffledRDD, 并由 collect 运行作业得出结果

1. 搜狗日志查询例子演示

查询搜索结果排名第 1 点击次序排在第 2 的数据

val rdd1 = sc.textFile("hdfs://hadoop1:9000/sogou/SogouQ1.txt") val rdd2=rdd1.map(\_.split("\t")).filter(\_.length==6)

rdd2.count()

val rdd3=rdd2.filter(\_(3).toInt==1).filter(\_(4).toInt==2) rdd3.count()

rdd3.toDebugString

Session 查询次数排行榜并把结果保存在 HDFS 中

val rdd4 = rdd2.map(x=>(x(1),1)).reduceByKey(\_+\_).map(x=>(x.\_2,x.\_1)). sortByKey(false).map(x=>(x.\_2,x.\_1))

rdd4.toDebugString

rdd4.saveAsTextFile("hdfs://hadoop1:9000/test/output1")

该命令运行的过程如下:

rdd4 的生成比较复杂,我们分步骤进行解析,轴线 map(x=>(x(1),1))是获取每行的第二个 字段(用户 Session)计数为 1,然后 reduceByKey(\_+\_)是安排 Key 进行累和,即按照用 户 Session 号进行计数求查询次数,其次 map(x=>(x.\_2,x.\_1))是把 Key 和 Value 位置互换, 为后面排序提供条件,使用 sortByKey(false)对数据进行按 Key 值进行倒排,此时需要注意 的是 Key 为查询次数,最后通过 map(x=>(x.\_2,x.\_1)再次交换 Key 和 Value 位置,得到了 (用户 Session 号,查询次数)结果

1. JSON
2. JSON

# Spark 在不同集群的练习

## 3.1 练习目的：

通过本练习，学员可以掌握在不同集群环境下，Spark的运行方法

## 3.2 练习案例说明：

## 3.3 练习步骤:

YARN运行过程演示

./spark-shell --master yarn --num-executors 3 --executor-memory 1g

local运行过程演示

./spark-shell --master local --num-executors 3 --executor-memory 1g

spark cluster运行过程演示

./spark-shell --master spark --num-executors 3 --executor-memory 1g

mesos运行过程演示

./spark-shell --master mesos --num-executors 3 --executor-memory 1g

# Spark SQL练习

## 5.1 练习目的：

通过本练习，学员可以掌握Spark SQL的使用方法

## 5.2 练习案例说明：

## 5.3 练习步骤:

1. 启动spark-sql

spark-sql --master spark://localhost:7077 --executor-memory 1g

1. 设置参数实例: SET spark.sql.shuffle.partitions=20;
2. 命令参数：通过 bin/spark-sql --help 可以查看 CLI 命令参数:
3. 使用 Case Class 定义 RDD 演示

val sqlContext=new org.apache.spark.sql.SQLContext(sc)

定义 Person 类,读入数据并注册为临时表

case class Person(name:String,age:Int)

val rddpeople=sc.textFile("hdfs://localhost:9000/people.txt").map(\_.split(",")).map(p =>Person(p(0),p(1).trim.toInt))

rddpeople.registerTempTable("rddTable")

在查询年纪在 13-19 岁之间的人员

sqlContext.sql("SELECT name FROM rddTable WHERE age >= 13 AND age <= 19").map(t => "Name: " + t(0)).collect().foreach(println)

1. parquet 演示

rddpeople.saveAsParquetFile("hdfs://localhost:9000/people.parquet")

val parquetpeople = sqlContext.parquetFile("hdfs://localhost:9000/people.parquet")

parquetpeople.registerTempTable("parquetTable")

sqlContext.sql("SELECT name FROM parquetTable WHERE age >= 25").map(t => "Name: " + t(0)).collect().foreach(println)

1. json 演示

sparkSQL1.1.0 开始提供对 json 文件格式的支持,这意味着开发者可以使用更多的数据源,如 鼎鼎大名的 NOSQL 数据库 MongDB 等。sqlContext 可以从 jsonFile 或 jsonRDD 获取 schema 信息,来构建 SchemaRDD,注册成表后就可以使用

val jsonpeople = sqlContext.jsonFile("hdfs://localhost:9000/people.json") jsonpeople.registerTempTable("jsonTable")

sqlContext.sql("SELECT name FROM jsonTable WHERE age >= 25").map(t => "Name: " + t(0)).collect().foreach(println)

1. sqlContext 中混合使用演示

sqlContext.sql("select a.name,a.age,b.age from rddTable a join parquetTable b on a.name=b.name").collect().foreach(println)

1. hiveContext 中混合使用演示

启动hive环境：nohup hive --service metastore > metastore.log 2>&1 &

修改hive配置文件：

<configuration>

<property>

<name>hive.metastore.uris</name>

<value>thrift://hadoop1:9083</value>

<description>Thrift URI for the remote metastore. Used by metastore client to

connect to remote metastore.</description> </property>

</configuration>

在spark conf下放置 hive-site.xml

val hiveContext = new org.apache.spark.sql.hive.HiveContext(sc)

hiveContext.sql("show tables").collect().foreach(println)

1. Cache 使用

sparkSQL 的 cache 可以使用两种方法来实现:

CacheTable()方法

CACHE TABLE 命令

第一步 对 rddTable 表进行缓存

sqlContext.cacheTable("rddTable")

sqlContext.sql("SELECT name FROM rddTable WHERE age >= 13 AND age <= 19").map(t => "Name: " + t(0)).collect().foreach(println)

在监控界面上看到该表数据已经缓存



解除缓存

sqlContext.uncacheTable("rddTable")

sqlContext.sql("UNCACHE TABLE parquetTable")

1. DSL 演示

val teenagers\_dsl = rddpeople.where('age >= 10).where('age <= 19).select('name)

teenagers\_dsl.map(t => "Name: " + t(0)).collect().foreach(println)

# Spark SQL Thrift练习

## 6.1 练习目的：

通过本练习，学员可以掌握Spark SQLThrift的使用方法

## 6.2 练习案例说明：

## 6.3 练习步骤:

**编辑hive配置文件**

<property>

<name>hive.server2.thrift.min.worker.threads</name> <value>5</value>

<description>Minimum number of Thrift worker threads</description>

</property>

<property>

<name>hive.server2.thrift.max.worker.threads</name> <value>500</value>

<description>Maximum number of Thrift worker threads</description>

</property>

<property>

<name>hive.server2.thrift.port</name>

<value>10000</value>

<description>Port number of HiveServer2 Thrift interface. Can be overridden by

setting $HIVE\_SERVER2\_THRIFT\_PORT</description> </property>

<property>

<name>hive.server2.thrift.bind.host</name>

<value>hadoop2</value>

<description>Bind host on which to run the HiveServer2 Thrift interface.Can be

overridden by setting$HIVE\_SERVER2\_THRIFT\_BIND\_HOST</description> </property>

</configuration>

# Spark Streaming 基础开发练习

## 7.1 练习目的：

通过本练习，学员可以掌握Spark Streaming基础的使用方法

## 7.2 练习案例说明：

## 7.3 练习步骤:

练习1:Spark Streaming HDFS WordCount

import org.apache.spark.SparkConf

import org.apache.spark.streaming.{Seconds, StreamingContext}

// Create the context

val ssc = new StreamingContext(sc, Seconds(2))

// Create the FileInputDStream on the directory and use the

// stream to count words in new files created

val lines = ssc.textFileStream("hdfs://localhost:9000/test.txt")

val words = lines.flatMap(\_.split(" "))

val wordCounts = words.map(x => (x, 1)).reduceByKey(\_ + \_)

wordCounts.print()

ssc.start()

ssc.awaitTermination()

练习2:读取文件演示

在该实例中 Spark Streaming 将监控某目录中的文件,获取在间隔时间段内变化的数据, 然后通过 Spark Streaming 计算出改时间段内单词统计数。

import org.apache.spark.SparkConf

import org.apache.spark.streaming.{Seconds, StreamingContext} import org.apache.spark.streaming.StreamingContext.\_

val sparkConf = new SparkConf().setAppName("FileWordCount").setMaster("local[2]")

// 创建 Streaming 的上下文,包括 Spark 的配置和时间间隔,这里时间为间隔 20 秒

val ssc = new StreamingContext(sparkConf, Seconds(20))

// 指定监控的目录,在这里为/home/hadoop/temp/

val lines = ssc.textFileStream("/home/hadoop/temp/")

// 对指定文件夹变化的数据进行单词统计并且打印

val words = lines.flatMap(\_.split(" "))

val wordCounts = words.map(x => (x, 1)).reduceByKey(\_ + \_) wordCounts.print()

// 启动 Streaming

ssc.start()

ssc.awaitTermination()

在 http://localhost:4040 监控 Spark Streaming 运行情况,可以观察到每 20 秒运行一次作业

练习3: 销售数据统计演示

import org.apache.log4j.{Level, Logger}

import org.apache.spark.{SparkContext, SparkConf}

import org.apache.spark.streaming.{Milliseconds, Seconds, StreamingContext} import org.apache.spark.streaming.StreamingContext.\_

import org.apache.spark.storage.StorageLevel

object SaleAmount {

def main(args: Array[String]) {

if (args.length != 2) {

System.err.println("Usage: SaleAmount <hostname> <port> ") System.exit(1)

}

Logger.getLogger("org.apache.spark").setLevel(Level.ERROR) Logger.getLogger("org.eclipse.jetty.server").setLevel(Level.OFF)

val conf = new SparkConf().setAppName("SaleAmount").setMaster("local[2]") val sc = new SparkContext(conf)

val ssc = new StreamingContext(sc, Seconds(5))

// 通过 Socket 获取数据,该处需要提供 Socket 的主机名和端口号,数据保存在内存和硬盘中

val lines = ssc.socketTextStream(args(0), args(1).toInt, StorageLevel.MEMORY\_AND\_DISK\_SER)

val words = lines.map(\_.split(",")).filter(\_.length == 6)

val wordCounts = words.map(x=>(1, x(5).toDouble)).reduceByKey(\_ + \_)

wordCounts.print() ssc.start() ssc.awaitTermination()

} }

java -cp LearnSpark.jar test.StreamingSimulation /home/hadoop/upload /saledata/tbStockDetail.txt 9999 1000

练习4: 数据流模拟器

import java.io.{PrintWriter}

import java.net.ServerSocket

import scala.io.Source

object StreamingSimulation {

// 定义随机获取整数的方法

def index(length: Int) = { import java.util.Random val rdm = new Random rdm.nextInt(length)

}

def main(args: Array[String]) {

// 调用该模拟器需要三个参数,分为为文件路径、端口号和间隔时间(单位:毫秒)

if (args.length != 3) {

System.err.println("Usage: <filename> <port> <millisecond>") System.exit(1)

}

// 获取指定文件总的行数

val filename = args(0)

val lines = Source.fromFile(filename).getLines.toList val filerow = lines.length

// 指定监听某端口,当外部程序请求时建立连接

val listener = new ServerSocket(args(1).toInt) while (true) {

val socket = listener.accept() new Thread() {

override def run = {

println("Got client connected from: " + socket.getInetAddress) val out = new PrintWriter(socket.getOutputStream(), true) while (true) {

Thread.sleep(args(2).toLong)

// 当该端口接受请求时,随机获取某行数据发送给对方

val content = lines(index(filerow)) println(content) out.write(content + '\n') out.flush()

}

socket.close() }

}.start() }

} }

练习5: 网络数据演示

在该实例中将由 4流数据模拟以 1 秒的频度发送模拟数据,Spark Streaming 通过 Socket 接收流数据并每 20 秒运行一次用来处理接收到数据,处理完毕后打印该时间段内数据出现的频 度,即在各处理段时间之间状态并无关系。

import org.apache.spark.{SparkContext, SparkConf}

import org.apache.spark.streaming.{Milliseconds, Seconds, StreamingContext} import org.apache.spark.streaming.StreamingContext.\_

import org.apache.spark.storage.StorageLevel

object NetworkWordCount { def main(args: Array[String]) {

val conf = new SparkConf().setAppName("NetworkWordCount").setMaster("local[2]")

val sc = new SparkContext(conf)

val ssc = new StreamingContext(sc, Seconds(20))

// 通过 Socket 获取数据,该处需要提供 Socket 的主机名和端口号,数据保存在内存和硬盘中

val lines = ssc.socketTextStream(args(0), args(1).toInt, StorageLevel.MEMORY\_AND\_DISK\_SER)

// 对读入的数据进行分割、计数

val words = lines.flatMap(\_.split(","))

val wordCounts = words.map(x => (x, 1)).reduceByKey(\_ + \_)

wordCounts.print() ssc.start() ssc.awaitTermination()

} }

实例 6:Stateful 演示

该实例为 Spark Streaming 状态操作,模拟数据由 4 流数据模拟以 1 秒的频度发送,Spark Streaming 通过 Socket 接收流数据并每 5 秒运行一次用来处理接收到数据,处理完毕后打印程 序启动后单词出现的频度,相比较前面实例在该实例中各时间段之间状态是相关的。

import org.apache.log4j.{Level, Logger}

import org.apache.spark.{SparkContext, SparkConf}

import org.apache.spark.streaming.{Seconds, StreamingContext} import org.apache.spark.streaming.StreamingContext.\_

object StatefulWordCount {

def main(args: Array[String]) {

if (args.length != 2) {

System.err.println("Usage: StatefulWordCount <filename> <port> ") System.exit(1)

}

Logger.getLogger("org.apache.spark").setLevel(Level.ERROR) Logger.getLogger("org.eclipse.jetty.server").setLevel(Level.OFF)

// 定义更新状态方法,参数 values 为当前批次单词频度,state 为以往批次单词频度

val updateFunc = (values: Seq[Int], state: Option[Int]) => { val currentCount = values.foldLeft(0)(\_ + \_)

val previousCount = state.getOrElse(0) Some(currentCount + previousCount)

}

val conf = new SparkConf().setAppName("StatefulWordCount").setMaster("local[2]")

val sc = new SparkContext(conf)

// 创建 StreamingContext,Spark Steaming 运行时间间隔为 5 秒

val ssc = new StreamingContext(sc, Seconds(5))

// 定义 checkpoint 目录为当前目录

ssc.checkpoint(".")

// 获取从 Socket 发送过来数据

val lines = ssc.socketTextStream(args(0), args(1).toInt) val words = lines.flatMap(\_.split(","))

val wordCounts = words.map(x => (x, 1))

// 使用 updateStateByKey 来更新状态,统计从运行开始以来单词总的次数

val stateDstream = wordCounts.updateStateByKey[Int](updateFunc) stateDstream.print()

ssc.start()

ssc.awaitTermination()

} }

实例 7:Window 演示

该实例为 Spark Streaming 窗口操作,模拟数据由4 流数据模拟以 1 秒的频度发送,Spark Streaming 通过 Socket 接收流数据并每 10 秒运行一次用来处理接收到数据,处理完毕后打印 程序启动后单词出现的频度。相比前面的实例,Spark Streaming 窗口统计是通过 reduceByKeyAndWindow()方法实现的,在该方法中需要指定窗口时间长度和滑动时间间隔。

import org.apache.log4j.{Level, Logger}

import org.apache.spark.{SparkContext, SparkConf} import org.apache.spark.storage.StorageLevel

import org.apache.spark.streaming.\_

import org.apache.spark.streaming.StreamingContext.\_

object WindowWordCount { def main(args: Array[String]) {

if (args.length != 4) {

System.err.println("Usage: WindowWorldCount <filename> <port> <windowDuration> <slideDuration>")

System.exit(1)

}

Logger.getLogger("org.apache.spark").setLevel(Level.ERROR) Logger.getLogger("org.eclipse.jetty.server").setLevel(Level.OFF)

val conf = new SparkConf().setAppName("WindowWordCount").setMaster("local[2]")

val sc = new SparkContext(conf)

// 创建 StreamingContext

val ssc = new StreamingContext(sc, Seconds(5))

// 定义 checkpoint 目录为当前目录

／／ssc.checkpoint(".")

// 通过 Socket 获取数据,该处需要提供 Socket 的主机名和端口号,数据保存在内存和硬盘中

val lines = ssc.socketTextStream(args(0), args(1).toInt, StorageLevel.MEMORY\_ONLY\_SER) val words = lines.flatMap(\_.split(","))

// windows 操作,第一种方式为叠加处理,第二种方式为增量处理

val wordCounts = words.map(x => (x , 1)).reduceByKeyAndWindow((a:Int,b:Int) => (a + b), Seconds(args(2).toInt), Seconds(args(3).toInt))

reduceByKeyAndWindow（func，3，15）

//val wordCounts = words.map(x => (x , 1)).reduceByKeyAndWindow(\_+\_, \_-\_,Seconds(args(2).toInt), Seconds(args(3).toInt))

wordCounts.print() ssc.start() ssc.awaitTermination()

} }

1. spark 序列化优化

spark.serializer org.apache.spark.serializer.KryoSerializer

2.

MEMORY\_AND\_DISK

如果数据可以放到内存，手动使用cache命令

1. 如何查看 executor，内存使用情况，并且GC 分析

executor – 设置告警机制（只能自己开发）

1. Yarn环境，必须spark 将Spark 提交到队列里面才可以，看到队列到内容存使用情况

yarn container -list application\_1462689750069\_0001（spark应用进行处理）

1. 非Yarn环境，8080 界面

executor 数量计算：

Yarn：

Node Manager：32G 内存

Container 内存 1～8G 32 ～ 4个

Executor 包含 N 个 Task（Map或者Reduce），Cache ～ Driver

假设，需要写入200G HDFS 数据 ～ Application ～ Driver ～ RDD

如果 MEM－DISK 级别的话，CPU＋内存的

64M ～ Par ～ MAP

（200\*1024 ／ 24 ） 200 core Executor （24core ＊ 10 ）

nodemanger = 3 Executor

3 \* 10 = 30 Executor ~4 vcore ~ 120 vcore

128G ~ Nodemanager = 96G

20台： 20 core , 64G mem ～ 48G spark

3／4＊（20 ＊ 20 ） ＝ 300cpu

3／4＊（48 ＊ 20） ＝ 720G

distinct([numTasks]))

--num-executors NUM 默认2个

distinct([numTasks]))

1. 将executors 的所有分配，交给yarn来负责
2. 重点关注程序里面task的数量
3. 监控RDD的分区情况， 然后调整参数进行测试

广播变量：

1. 手机号，URL，时间， 流量

1.使用程序，生成256的数据，放入hdfs

2.在Hbase中，增加一个表，url，keywords映射

2.使用spark 程序，计算如下需求：

1.找出流量异常的手机号，在单一URL上，流量超过30M的，

输出，url，手机号，流量，时间到hdfs上

2.对输入数据与Hbase表的表进行join，生成一个新的数据，包含分类

3.统计url的pv，并找出top 10