Introduction to Data Analysis with Spark

TANG Gen

Outline

- Spark programming model
- Installation
- Review of Spark shell and SparkContext
- Get familiar with data
- Structuring data
- Summary statistics of data

Spark programming model

- Spark program typically consists of the following three things
  - Define a set of transformations on input datasets
  - Invoke actions that output the transformed datasets to persistent storage or return results to driver's local memory
  - Run local computation that operate on the results computed in a distributed fashion

Installation

/Download data

- The data we will use in this case is German hospital data in 2010
- It contains pairs of the information of patient records with matching field assigned a numerical score from 0 to 1.0

Name	Address	City	State	Phone
Josh’s Coffee Shop	1234 Sunset Boulevard	West Hollywood	CA	(213)-555-1212
Josh Cofee	1234 Sunset Blvd West	Hollywood	CA	555-1212
Coffee Chain #1234	1400 Sunset Blvd #2	Hollywood	CA	206-555-1212
Coffee Chain Regional Office	1400 Sunset Blvd Suite 2	Hollywood	California	206-555-1212

Installation

/Download data

- You can download the data from

https://archive.ics.uci.edu/ml/machine-learning-databases/00210/donation.zip

- run

unzip donation.zip

unzip “block.zip”*

mkdir linkage

*mv block*csv linkage/*

Review of Spark shell and SparkContext

/Spark shell

- Enter “Spark” directory and run

  ./bin/spark-shell

- A SparkContext object is already created as sc

- Tips:
  - Type “:help” to get the help information
  - Type “:load” to load the script to Spark shell
  - Type the name of a variable and then type tab to get the methods of this variable

Review of Spark shell and SparkContext

/Create RDD from files

- In Spark shell, type

  val rawblocks = sc.textFile("…")

- rawblocks is "org.apache.spark.rdd.RDD[String]". It means that rawblocks is an RDD and the its element is "String"

- Thanks to Scala "type inference" feature, we don't need to specify type information in our variable declaration

- Whenever we create a new variable in Scala, we must preface the name of the variable with either *var* or *val*

- variable with *val* is immutable, with *var* is mutable

Get familiar with data

/Bring data from the workers to the driver

- When we want to take a look at the data, there are three common actions to use:

```
scala> rawblocks.first()
res3: String = "id_1","id_2","cmp_fname_c1","cmp_fname_c2","cmp_lname_c1","cmp_lname_c2","cmp_sex","cmp_bd","cmp_bm","
cmp_by","cmp_plz","is_match"
```

```
scala> rawblocks.take(10)
res5: Array[String] = Array("id_1","id_2","cmp_fname_c1","cmp_fname_c2","cmp_lname_c1","cmp_lname_c2","cmp_sex","cmp_b
d","cmp_bm","cmp_by","cmp_plz","is_match", 37291,53113,0.833333333333333,?,1,?,1,1,1,1,0,TRUE, 39086,47614,1,?,1,?,1,1
,1,1,1,TRUE, 70031,70237,1,?,1,?,1,1,1,1,1,TRUE, 84795,97439,1,?,1,?,1,1,1,1,1,TRUE, 36950,42116,1,?,1,1,1,1,1,1,1,TRU
E, 42413,48491,1,?,1,?,1,1,1,1,1,TRUE, 25965,64753,1,?,1,?,1,1,1,1,1,TRUE, 49451,90407,1,?,1,?,1,1,1,1,0,TRUE, 39932,4
0902,1,?,1,?,1,1,1,1,1,TRUE)
```

```
scala> rawblocks.collect()
```

- However, RDD.collect() will bring all the data to the driver program which usually will cause the crash of driver program

Get familiar with data

/Skip header

- As the file contains header and data, we need to create an RDD without header. It is can be done by RDD.filter()

```
def isHeader(line: String): Boolean = {
  line.contains("id_1")
}

val noheader = rawblocks.filter(x => !isHeader(x))
```

```
scala> noheader.first()
res6: String = 37291,53113,0.833333333333333,?,1,?,1,1,1,1,0,TRUE
```

Structuring data

/Tuples and case classes

- As we see, the structure of data is as follows:

1. The first two fields are integer IDs that represent the patients that were matched in the record.

2. The next nine values are (possibly missing) double values that represent match scores on different fields of the patient records, such as their names, birthdays, and location.

3. The last field is a boolean value (TRUE or FALSE) indicating whether or not the pair of patient records represented by the line was a match or not.

Structuring data

/Tuples and case classes

- We want to map the data to appropriate type:
  - The first two columns are mapped int
  - The column 3 to 12 to Array[Double]
  - The last column to Boolean

Structuring data

/Tuple

```
def toDouble(s: String): Double = {
if ("?".equals(s)) Double.NaN else s.toDouble
}

def parse(line: String) = {
  val pieces = line.split(',')
  val id1 = pieces(0).toInt
  val id2 = pieces(1).toInt
  val scores = pieces.slice(2, 11).map(toDouble)
  val matched = pieces(11).toBoolean
  (id1, id2, scores, matched)
}
```

- We can access the data stored in tuple by
  - tuple._1
  - tuple.productElement(0)
  - tuple.productArity

Structuring data

/Case classes

- However, use tuples in Spark sometime causes the problems of comprehension, as we address data by position instead of meaningful name

- Scala provides a convenient syntax for this problem, called *case class*

```
case class MatchData(id1: Int, id2: Int,
  scores: Array[Double], matched: Boolean)

def parse(line: String) = {
  val pieces = line.split(',')
  val id1 = pieces(0).toInt
  val id2 = pieces(1).toInt
  val scores = pieces.slice(2, 11).map(toDouble)
  val matched = pieces(11).toBoolean
  MatchData(id1, id2, scores, matched)
}
```

Structuring data

/Ship code to the workers

- Now, we can use tuples or case class to structure the data

```
scala> val parsed = noheader.map(line => parse(line))
parsed: org.apache.spark.rdd.RDD[(Int, Int, Array[Double], Boolean)] = MapPartitionsRDD[7] at map at <console>:31

scala> parsed.first()
res10: (Int, Int, Array[Double], Boolean) = (37291,53113,Array(0.833333333333333, NaN, 1.0, NaN, 1.0, 1.0, 1.0, 1.0, 0
.0),true)
```

```
scala> val parsed = noheader.map(line => parse(line))
parsed: org.apache.spark.rdd.RDD[MatchData] = MapPartitionsRDD[8] at map at <console>:33

scala> parsed.first()
res11: MatchData = MatchData(37291,53113,[D@376f3de6,true)
```

Summary statistics of data

/Histogram

- Let's start by creating a simple histogram on matched column

```
val matchCounts = parsed.map(md => md.matched).countByValue()

val matchCountsSeq = matchCounts.toSeq
matchCountsSeq.sortBy(_._2).foreach(println)
```

```
scala> matchCountsSeq.sortBy(_._2).foreach(println)
(true,20931)
(false,5728201)
```

Summary statistics of data

/Basic statistics for continuous variables

- For instances of RDD[Double], Spark provides stats API to calculate basic statistics of data

```
scala> parsed.map(md => md.scores(0)).stats()
res13: org.apache.spark.util.StatCounter = (count: 5749132, mean: NaN, stdev: NaN, max: NaN, min: NaN)
```

- Due to missing value, we need to clean data before calculate basic statistics

```
import java.lang.Double.isNaN
parsed.map(md => md.scores(0)).filter(!isNaN(_)).stats()
```

Summary statistics of data

/Basic statistics for continuous variables

```
val stats = (0 until 9).map(i => {
parsed.map(md => md.scores(i)).filter(!isNaN(_)).stats()
})
```

```
stats: scala.collection.immutable.IndexedSeq[org.apache.spark.util.StatCounter] = Vector((count: 5748125, mean: 0.712902, stdev: 0.388758, max: 1.000000, min: 0.000000), (count: 103698, mean: 0.900018, stdev: 0.271316, max: 1.000000, min: 0.000000), (count: 5749132, mean: 0.315628, stdev: 0.334234, max: 1.000000, min: 0.000000), (count: 2464, mean: 0.318413, stdev: 0.368492, max: 1.000000, min: 0.000000), (count: 5749132, mean: 0.955001, stdev: 0.207301, max: 1.000000, min: 0.000000), (count: 5748337, mean: 0.224465, stdev: 0.417230, max: 1.000000, min: 0.000000), (count: 5748337, mean: 0.488855, stdev: 0.499876, max: 1.000000, min: 0.000000), (count: 5748337, mean: 0.222749, stdev: 0.416091, max: 1.000000, min: 0.000000), (count: 5736289, mean: 0.005529, stdev: 0.074149, max: 1.000000,...
```

Case studies

/Spark at Typesafe

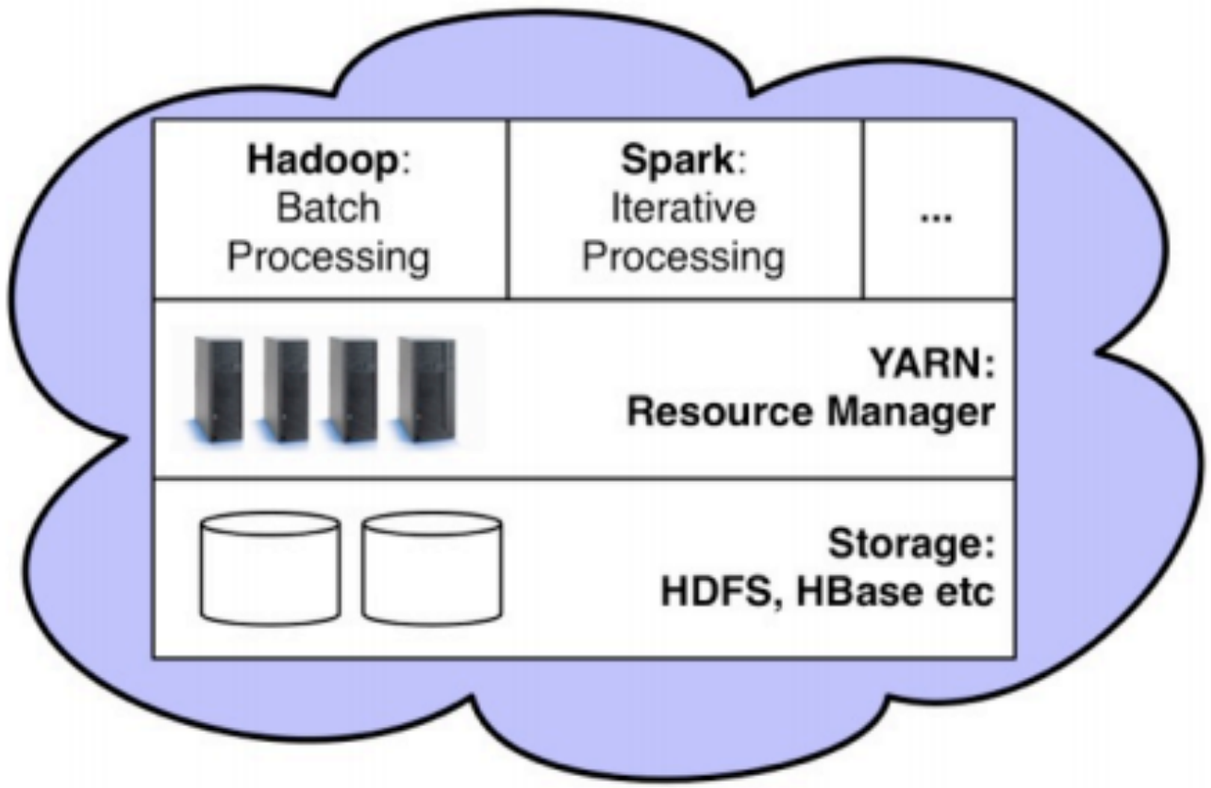
http://www.slideshare.net/deanwampler/spark-the-next-top-compute-model

- Hadoop: most algorithms are much harder to implement in this restrictive map-then-reduce model

- Spark: fine-grained “combinators” for composing algorithms

Case studies

/Spark at Yahoo



I. **science** ... Spark API & MLlib ease development of ML algorithms
II. **speed** ... Spark reduces latency of model training via in-memory RDD etc
III. **scale** ... YARN brings Hadoop datasets & servers at scientists' fingertips

http://spark-summit.org/talk/feng-hadoop-and-spark-join-forces-at-yahoo/