Simple Tutorial
CS-562
Introduction to data analysis using Apache Spark

Instructor: Vassilis Christophides
T.A.: Moustakas Serafeim

Outline
- What is Scala?
- Core Functionality
- Simple First Tour of Scala
- Crash Tests
What is Scala?

- Scala is a mixture. It is the place where 2 world connect:
  - Object oriented programming
  - Functional Programming

- This mixture is the strength of Scala.
- We can say that Scala is a better version of Java.
Core Functionality

var vs val

- Var -> Variable
- Val -> Variable + Final
Functions Vs Methods

- Function is a group of statements that perform a task

But what is the difference of a method and a function?

   Method: a function, which is defined as a member of some object
   Function: a group of statements that perform a task

Load our data

- `sc.parallelize()`

Parallelized collections are created by calling SparkContext's `parallelize` method on an existing collection.

The elements of the collection are copied to form a distributed dataset that can be operated on in parallel.
Lists & Arrays

- Scala Lists are quite similar to arrays

- All the elements of a list have the same type but...

- First, lists are immutable, which means elements of a list cannot be changed by assignment.

- Second, lists represent a linked list whereas arrays are flat.

example: val nums: List[Int] = List(1, 2, 3, 4)

Zip & ZipWithIndex

Zip

Returns a list formed from this list and another iterable collection by combining corresponding elements in pairs.

zipWithIndex

Zips this list with its indices. Returns: A new list containing pairs consisting of all elements of this list paired with their index. Indices start at 0.
Map

Spark RDD map function returns a new RDD by applying a function to all elements of source RDD.

```
Partition 1: RDD = X

map ( f(x) ) = (x, 1)

Partition 2: RDD = Y
```

Filter

Spark RDD filter function returns a new RDD containing only the elements that satisfy a predicate.

```
Partition 1: RDD = X

filter ( f(x) ) = (x | x > 0)

Partition 2: RDD = Y
```
Reduce
Spark RDD reduce function reduces the elements of this RDD using the specified commutative and associative binary operator.

GroupBy
Spark RDD groupBy function returns an RDD of grouped items.
Quick Tour of Scala Part 1

Scala functions (closures)

(x: Int) => x + 2 // full version
x => x + 2 // type inferred
_*_ + 2 // placeholder syntax (each argument must be used exactly once)
x => {_ // body is a block of code
  val numberToAdd = 2
  x + numberToAdd
}_{ // Regular functions
  def addTwo(x: Int): Int = x + 2
}
Quick Tour of Scala Part 2
(electric boogaloo)

Processing collections with functional programming

```scala
val list = List(1, 2, 3)
list.foreach(x => println(x)) // prints 1, 2, 3
list.foreach(printIn) // same

list.map(x => x + 2) // returns a new List(3, 4, 5)
list.map(_ + 2) // same

list.filter(x => x % 2 == 1) // returns a new List(1, 3)
list.filter(_. % 2 == 1) // same

list.reduce((x, y) => x + y) // => 6
list.reduce(_ + _) // same
```

All of these leave the list unchanged as it is immutable.

Functional methods on collections

There are a lot of methods on Scala collections, just google Scala Seq or [here](https://www.scala-lang.org/seq).

<table>
<thead>
<tr>
<th>Method on Seq[T]</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>map(f: T =&gt; U, Seq[U])</td>
<td>Each element is result of f</td>
</tr>
<tr>
<td>flatMap(f: T =&gt; Seq[U], Seq[U])</td>
<td>One to many map</td>
</tr>
<tr>
<td>filter(f: T =&gt; Boolean, Seq[T])</td>
<td>Keep elements passing f</td>
</tr>
<tr>
<td>exists(f: T =&gt; Boolean, Seq[T])</td>
<td>True if one element passes f</td>
</tr>
<tr>
<td>forall(f: T =&gt; Boolean, Seq[T])</td>
<td>True if all elements pass</td>
</tr>
<tr>
<td>reduce(f: (T, T) =&gt; T, Seq[T])</td>
<td>Merge elements using f</td>
</tr>
<tr>
<td>groupBy(key: K, Seq[T])</td>
<td>Group elements by f</td>
</tr>
<tr>
<td>sortBy(key: K, Seq[T])</td>
<td>Sort elements</td>
</tr>
</tbody>
</table>

...
Crash Tests

Lets Make our First Crash-Test

Our Data:

val data = 1 to 10000

create an RDD based on that data…

val distData = sc.parallelize(data)

then use a filter to select values less than 10…

distData.filter(_ < 10).collect()
Spark Deconstructed: Log Mining Example

We start with Spark running on a cluster... submitting code to be evaluated on it:

```scala
// load error messages from a log into memory
// then interactively search for various patterns
// https://gist.github.com/ceteri/8ae5b9509a08c68a1132
val lines = sc.textFile("hdfs:///...")
// base RDD
val errors = lines.filter(_.startsWith("ERROR"))
val messages = errors.map(_.split("\s")).map(r => r(1))
messages.cache()

// action 1
messages.filter(_.contains("mysql")).count()
// action 2
messages.filter(_.contains("php")).count()
```
Spark Deconstructed: Log Mining Example

```scala
// base RDD
val lines = sc.textFile("hdfs://...")

// transformed RDD
val errors = lines.filter(_.startsWith("ERROR"))
val messages = errors.map(_.split("""\""")_._3).map(x => x)
messages.cache()
```

Spark Deconstructed: Log Mining Example

```scala
// base RDD
val lines = sc.textFile("hdfs://...")

// transformed RDD
val errors = lines.filter(_.startsWith("ERROR"))
val messages = errors.map(_.split("""\""")_._3).map(x => x)
messages.cache()
```

**discussing the other part**
Spark Deconstructed: Log Mining Example

```scala
// base RDD
val lines = sc.textFile("hdfs://...")
val errors = lines.filter(_.startsWith("ERROR")
val messages = errors.map(_.split("\"\""\"\"\"""\"\"\"\"") :map(r => r))
messages.count()

// action 1
messages.filter(_.contains("mysql")).count()
```

Spark Deconstructed: Log Mining Example

```scala
// base RDD
val lines = sc.textFile("hdfs://...")
val errors = lines.filter(_.startsWith("ERROR")
val messages = errors.map(_.split("\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"") :map(r => r))
messages.count()

// action 1
messages.filter(_.contains("mysql")).count()
```
Spark Deconstructed: Log Mining Example

```scala
// base RDD
val lines = sc.textFile("hdfs://...")

// transformed RDD
val errors = lines.filter(_.startsWith("ERROR"))
val messages = errors.map(_.split(" ").map(_.toInt))
messages.collect()

// action 1
messages.filter(_.contains("myobj"))
```

Spark Deconstructed: Log Mining Example

```scala
// base RDD
val lines = sc.textFile("hdfs://...")

// transformed RDD
val errors = lines.filter(_.startsWith("ERROR"))
val messages = errors.map(_.split(" ").map(_.toInt))
message.collect()

// action 1
messages.filter(_.contains("myobj"))
```
Spark Deconstructed: Log Mining Example

// action 2
messages.filter(_.contains("php"))\n
// accessing the other part
messaging\n
Driver

Worker

process from cache

Worker

process from cache

Worker

process from cache

Worker

process from cache
Spark Deconstructed:

Looking at the RDD transformations and actions from another perspective...

```scala
// load event messages from a log into memory
// time-incorrectly match the various patterns
// https://spark.apache.org/docs/latest/understanding-rdd-operations.html

// base RDD
val lines = sc.textFile("hdfs://...")

// transformed RDD
val accessor = lines.filter(_.startsWith("name="))
val messages = accessor.map(_.split('|')(1).split(|)).collect

// action 1
messages.filter(_.contains("msg")) // collect

// action 2
messages.filter(_.contains("msg") & !it.contains)
```

Spark Deconstructed:

```
// base RDD
val lines = sc.textFile("hdfs://...")
```
Spark Deconstructed:

```
val errors = lines.filter(_.startsWith("ERROR: "))
val messages = errors.map(_.split("\n\n")) .map(_.split("\"\"")) .map(_.toString)
messages.foreach()
```

Spark Deconstructed:

```
// action 1
messages.filter(_.contains("ERROR") .count)
```
Special Thanks

- CS543 Presentations
- Coursera Introduction to Apache Spark, University of California, Databricks
- https://backtobazics.com/big-data/spark