BIG DATA

Scala Programming Language

K V Subramaniam
Computer Science and Engineering

BIG DATA

Hadoop Challenges

- In last lectures
  - Algorithms for relational / page rank
  - Require use of multiple files
  - Requires use of iteration
- No support from language
- Is there a language suitable for Big Data

BIG DATA

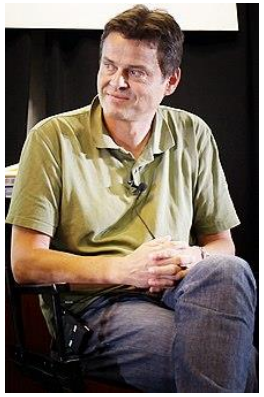
Overview of lecture

- Scala overview
- Scala Notation
- Features required for Big Data
- Functional programming
- Big Data and Scala

Scala Overview

What is Scala?

- JVM based language that can be called and call Java – SCAlable LAnguage
- A more concise, richer Java + Functional Programming
  - Blends OO + FP
- Strongly statically typed
  - But feels dynamically typed
  - Type inferencing saves typing
- Developed by Marvin Odersky at EPFL (Switzerland)
- Released in 2004



Roots in Java – Why move?

What's wrong with Java?

- Not designed for highly concurrent programs
  - The original Thread model was just *wrong* (it's been fixed)
  - Java 5+ helps by including java.util.concurrent
- Verbose
  - Too much of Thing thing = new Thing();
  - Too much "boilerplate," for example, getters and setters

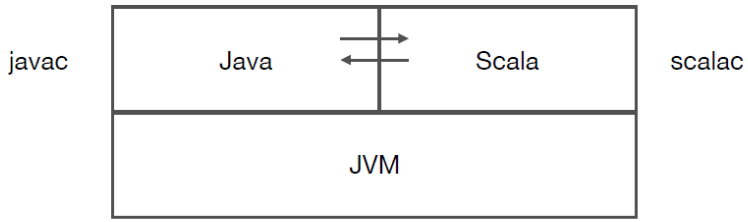
What's right with Java?

- Very popular
- Object oriented (mostly), which is important for large projects
- Strong typing (more on this later)
- The fine large library of classes
- **The JVM!** Platform independent, highly optimized

BIG DATA

Java and Scala: Spot the differences

- Java is a *good* language, and Scala is a lot like it
- For each difference, there is a *reason*--none of the changes are “just to be different”
- Scala and Java are (almost) completely interoperable
  - Call Java from Scala? No problem!
  - Call Scala from Java? Some restrictions, but mostly OK.
  - Scala compiles to .class files (a *lot* of them!), and can be run with either the scala command or the java command



Quick Tour of Scala – Differences with Java

BIG DATA

Java and Scala: Spot the differences

Declaring variables:

```
var x: Int = 7
var x = 7 // type inferred
val y = “hi” // read-only
```

Java equivalent:

```
int x = 7;

final String y = “hi”;
```

Functions:

```
def square(x: Int): Int = x*x
def square(x: Int): Int = {
  x*x
}
def announce(text: String) =
{
  println(text)
}
```

Java equivalent:

```
int square(int x) {
  return x*x;
}

void announce(String text) {
  System.out.println(text);
}
```

BIG DATA

Java and Scala: Spot the differences

- Minimal Verbosity
- Referential Transparency – type inferencing
- Concurrency
- Functional Programming

Minimal Verbosity

- Java:
  - ```
    class Person {
        private String firstName;
        private String lastName;
        private int    age;

        public Person(String firstName, String lastName, int age) {
            this.firstName = firstName;
            this.lastName  = lastName;
            this.age  = age;
        }

        public void setFirstName(String firstName) { this.firstName = firstName; }
        public void String getFirstName() { return this.firstName; }
        public void setLastName(String lastName) { this.lastName = lastName; }
        public void String getLastName() { return this.lastName; }
        public void setAge(int age) { this.age = age; }
        public void int getAge() { return this.age; }
    }
    ```
- Scala:
  - ```
    class Person(var firstName: String, var lastName: String, var age: Int)
    ```
- Source: http://blog.objectmentor.com/articles/2008/08/03/the-seductions-of-scala-part-i

Type Inferencing

- Java is statically typed--a variable has a type, and can hold only values of that type
  - You must specify the type of every variable
  - Type errors are caught by the compiler, not at runtime--this is a big win
  - However, it leads to a lot of typing (pun intended)
- Languages like Ruby and Python don’t make you declare types
  - Easier (and more fun) to write programs
  - Less fun to debug, especially if you have even slightly complicated types

Type Inferencing

- Scala is also statically typed, but it uses type inferencing--that is, it figures out the types, so you don’t have to
  - **The good news**: Less typing, more fun, type errors caught by the compiler
  - **The bad news**: More kinds of error messages to get familiar with

Consistency

- In Java, every value is an object--unless it's a primitive
  - Numbers and booleans are primitives for reasons of efficiency, so we have to treat them differently (you can't "talk" to a primitive)
  - In Scala, all values are objects. Period.
  - The compiler turns them into primitives, so no efficiency is lost (behind the scenes, there are objects like RichInt)

- Java has operators (+, <, ...) and methods, with different syntax
  - In Scala, operators are just methods, and in many cases you can use either syntax

Concurrency and Functional Programming

Concurrency

- Broadly speaking, concurrency can be either:
  - Fine-grained: Frequent interactions between threads working closely together (extremely challenging to get right)
  - Coarse-grained: Infrequent interactions between largely independent sequential processes (much easier to get right)
- Java 5 and 6 provide reasonable support for traditional fine-grained concurrency
  - Threads
- Scala has total access to the Java API
  - Hence, it can do anything Java can do
  - And it can do much more (see next slide)
- Scala also has Actors for coarse-grained concurrency
  - Sending messages (use the send ! Abstraction)

BIG DATA

Functional Programming

- The big nasty problem with concurrency is dealing with shared state--multiple threads all trying to read and maybe change the same variables
- If all data were immutable, then any thread could read it any time, no synchronization, no locks, no problem
- But if your program couldn’t ever change anything, then it couldn’t ever do anything, right?
- Wrong!
- There is an entire class of programming languages that use only immutable data, but work just fine: the functional languages

Concurrency

- The best-known functional languages are ML, OCaml, and Haskell
- Functional languages are regarded as:
  - "Ivory tower languages," used only by academics (mostly but not entirely true)
  - Difficult to learn (mostly true)
  - The solution to all concurrent programming problems everywhere (exaggerated, but not entirely wrong)
- Scala is an "impure" functional language--you can program functionally, but it isn't forced upon you

BIG DATA

Functional Programming

- Immutable – functional operations create new structures
  - Don’t modify existing structures
- Program implicitly captures data flow
- Order of operations not significant
- Functions
  - Are objects
  - can be passed as arguments to functions
  - can return functions
  - Can operate on collections

Functional Programming – differences with java

Quicksort program in Scala – java style

Observe

Focus on HOW?

Explicitly iterate

Determine when and how to swap()

```
def sort(xs: Array[Int]) {
  def swap(i: Int, j: Int) {
    val t = xs(i); xs(i) = xs(j); xs(j) = t
  }
  def sort1(l: Int, r: Int) {
    val pivot = xs((l + r) / 2)
    var i = l; var j = r
    while (i <= j) {
      while (xs(i) < pivot) i += 1
      while (xs(j) > pivot) j -= 1
      if (i <= j) {
        swap(i, j)
        i += 1
        j -= 1
      }
    }
    if (l < j) sort1(l, j)
    if (j < r) sort1(i, r)
  }
  sort1(0, xs.length - 1)
}
```

BIG DATA

Functional Programming – differences with java

Focus on solving the problem

Observe

Focus on WHAT, not HOW

```
def sort(xs: Array[Int]): Array[Int] = {
  if (xs.length <= 1) xs
  else {
    val pivot = xs(xs.length / 2)
    Array.concat(
      sort(xs filter (pivot >)),
           xs filter (pivot ==),
      sort(xs filter (pivot <)))
  }
}
```

Pick a pivot

Sort values smaller than the pivot

Sort values larger than the pivot

Concatenate the result

BIG DATA

Self Learning Exercise

- Does this sort array in ascending order or descending order?
- Consider the program with array
  - xs=3,1,2,0,7,6,4,5
- Write a program to sort in the reverse order (if ascending, sort descending)
- How can we parallelize this?

```
def sort(xs: Array[Int]): Array[Int] = {
  if (xs.length <= 1) xs
  else {
    val pivot = xs(xs.length / 2)
    Array.concat(
      sort(xs filter (pivot >)),
           xs filter (pivot ==),
      sort(xs filter (pivot <)))
  }
}
```

Functional programming and collections

Original List is left unchanged

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

Functional Programming and Big Data

BIG DATA

- Big data architectures leverage
  - Parallel disk, memory and CPU in clusters
- Operations consist of independently parallel operations
  - Similar to *map()* operator in a functional language
- Parallel operations have to be consolidated
  - Similar to *aggregation()* operators in functional languages
- Hence the fit

THANK YOU

K V Subramaniam, Usha Devi

Dept. of Computer Science and Engineering

subramaniamkv@pes.edu

ushadevibg@pes.edu