CSYE 7200

Big Data System Engineering using Scala

Syllabus

Prof. Robin Hillyard, Boston

Spring 2020

r.hillyard@neu.edu

This course concentrates more on Scala and functional programming than on Big Data. However, the goal of the class is to show that Scala has an important role to play in both Big Data systems and concurrent systems. We will cover Spark—currently one of the most important tools in the Big Data zoo—which itself is written in Scala. Above all, this is a *practical* class: you will learn many aspects of programming and software engineering that are useful whatever language you are using.

Scala is also important for concurrent systems and we will also talk about reactive systems and micro-services.

Recommended text(s):

***Programming in Scala*—Odersky, Spoon & Venners, Artima** (3rd edition)
***Functional Programming in Scala*—Chiusano & Bjarnason, Manning**

These are both excellent texts. The first is the definitive guide to Scala co-written by the originator of the language. The second is a beautifully written introduction to the concepts of functional programming, with the advantage that it uses Scala.

Course Objectives: Spark has revolutionized the approach to processing big data, abstracting away the details of map/reduce such that programmers are hardly aware of it. While much work with Spark can be programmed with Java, Python, R, or even plain old SQL, it is often the ETL (ingestion) phase of Big Data work which particularly requires Scala. Why should this be so? Most non-functional languages are oriented towards doing things as long as everything is working fine. However,

real life encounters nulls and occasionally causes exceptions to be thrown. These abnormal situations are very well handled using Scala. Secondly, it is when gathering data that it is most important to be protected by type-safety.

In any case, Spark is *implemented* in Scala. Thus, programming in Scala helps you not only with best practices, but also enables you to look "under the hood". But functional programming (*fp*) is not only ideal for parallel programming with Spark. *fp* is ideally suited to concurrent programming (all modern computing is potentially concurrent) because side effects and mutable state are either eliminated or carefully encapsulated.

Nevertheless, *fp* requires a different way of thinking from imperative programming (Java, C[++], etc.). This class aims to cover the fundamentals of *fp* (in Scala), and to provide a basic, practical foundation for many different types of programming. Topics to be covered, in addition to all basic programming techniques are: numerical programming, reactive programming (using Akka), parser-combinators and DSLs, testing frameworks and getting the job done. While *fp* has a solid mathematical foundation, the mathematics required is really just basic logic and axioms. You don't need a "higher math" background.

The last third of the class will be largely concerned with projects which will not only test your knowledge of Scala but will give you a great opportunity to tackle something really interesting and, hopefully, useful.

Prerequisite: none, although it is probably helpful to have had some experience with a programming language such as Java or Python.

Grading Breakdown: 20% mid, 25% final, 30% project, 25% homework.

Project Information: Projects will normally be worked on in pairs (or trios) and will implement some analysis of Big Data (possibly streaming) typically using Spark and maybe Zeppelin. Alternatively, a project might implement a reactive system. Projects must include some significant Scala coding, with unit tests, and demonstrate scalability. For more detail and ideas, see the Project page under Course Materials on Blackboard.

Course Schedule (may vary somewhat):

Week 1		**Introduction; Big Data systems; Spark overview; Looking under the hood: Scala; Scala and Functional Programming.**
Week 2		**Scala (continued); Important concepts. Parallel Processing and Mutable State.**
Week 3		**More Advanced Scala Concepts (REPL, substitution, type inference, lazy functions, lists and streams, generics and variance); Dealing with Exceptional Conditions.**
Week 4		**Collections; Streams; Managing State; Types.**
Week 5		**Functional composition and for comprehensions; recursion.**
Week 6		**Syntax; Type Declarations; Functions, methods & operators; Specifications & Unit Tests.**
Week 7		**Implicits; Serialization/de-serialization; Parallel Processing and Futures; Monoids, functors, and monads.**
Week 8		**Mid-term exam; Syntactic sugar; Repositories.**
Week 9		**Enumerated Types; Actors; Syntactic sugar (continued) and pattern matching; Tour of the API; Parsing and DSLs.**
Week 10		**Spark details; Zeppelin; GraphX, MLlib; Spark Streaming and Spark SQL.**
Week 11		**Play/Activator; Numerical Computing.**
Week 12 thru 13		**Projects and other topics not already covered.**
Week 14		**Project presentations**
Week 15		**Final exam and remaining Project presentations.**

Academic Integrity:

The Northeastern University academic integrity policy applies to your work in this course. All students are expected to adhere to this policy. I expect each and every one of you to familiarize yourself with this policy by visiting: http://www.northeastern.edu/osccr/academic-integrity-policy/

I want you to take this *very* seriously. Most of you will be in your second year of the program. If I give you an *F* because, for example, you cheated in an exam, you may not be able to recover. You will have wasted more than a year of academic study. It's just not worth it.