Scalable and Cloud Programming

Prof. Gianluigi Zavattaro
gianluigi.zavattaro@unibo.it

Logistics and web site

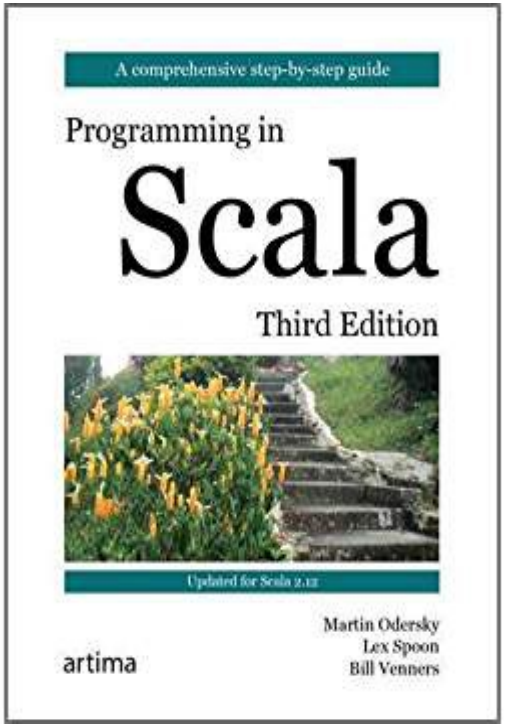
- Lectures:
  - Tuesday, E1, Mura A. Zamboni 2/A, 12:00 – 15:00
  - Wednesday, E2, Mura A. Zamboni 2/A, 15:00 – 17:00
- Course web site
  - available as an instance of the UniBO e-Learning service
    https://virtuale.unibo.it/course/view.php?id=58895
    - slides
    - examples of code used during the lectures
    - agenda with contents of each lecture
    - project work specification and submission
    - ...
  - registration password: ScalableZAV24
  - please register asap!
    - I will use the mailing list for internal communications
      (including messages with instructions to redeem your cloud credits)

This course..

- .. deals with programming with large datasets but ..
  - .. it is not a Data Analytics course
  - .. it is not a Machine Learning course
- Traditional approaches to programming usually do not scale when your dataset gets too large
  - You usually have to re-implement your system!
    - Re-organize and distribute your data
    - Distribute and coordinate your processes
- We will experiment an alternative programming style that ..
  - .. makes it easier to scale your small problem to the large ..
  - .. by leveraging on Scala and Spark

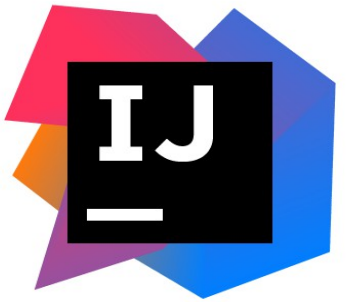
Scala = FP + OO

- "Scalable Language", i.e., able to grow with the demands of its users
- Design started in 2001 by Martin Odersky (professor at EPFL)
  - First release 2004, initially distributed by Typesafe Inc., recently rebranded as Lightbend Inc.
- Initial goal: design a language better than Java (but connected with it) that was both object-oriented and functional
- Scala programs are translated in Java bytecode and executed on JVM



Download & Install

- Scala can be downloaded and installed following the instructions at: https://scala-lang.org/download/
  - SBT: Scala Build Tool
  - IntelliJ IDEA:
    - IDE with a specific customization for Scala



First example

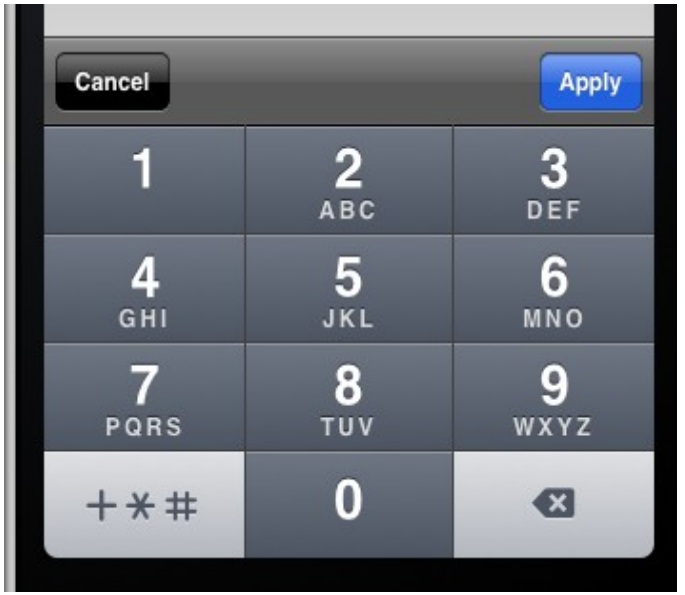
```
object QuickSort {

  // create an indexed sequence of 1000 random ints
  val r = scala.util.Random
  val randomArray = (for (i <- 1 to 1000) yield r.nextInt(100000))

  def main(args: Array[String]) = {
    // do the sorting
    val sortedArray = quickSort(randomArray)
    // print the ordered sequence
    sortedArray.foreach(println)
  }

  // the quicksort recursive algorithm
  def quickSort(xs: IndexedSeq[Int]): IndexedSeq[Int] = {
    if (xs.length <= 1) xs
    else {
      val pivot = xs(r.nextInt(xs.length))
      IndexedSeq.concat(
        quickSort(xs.filter((x)=>(x < pivot))),
        xs.filter ((x)=>(x == pivot)),
        quickSort(xs.filter((x)=>(x > pivot))))
    }
  }
}
```

Scala's collections

- Scala standard library offers a rich set of data structures, called collections
  - Collections are equipped with very expressive higher-order functions
  - Programming with collections means to take advantage of such higher-order functions

Second example

- There exists a traditional way to associate letters to digits:

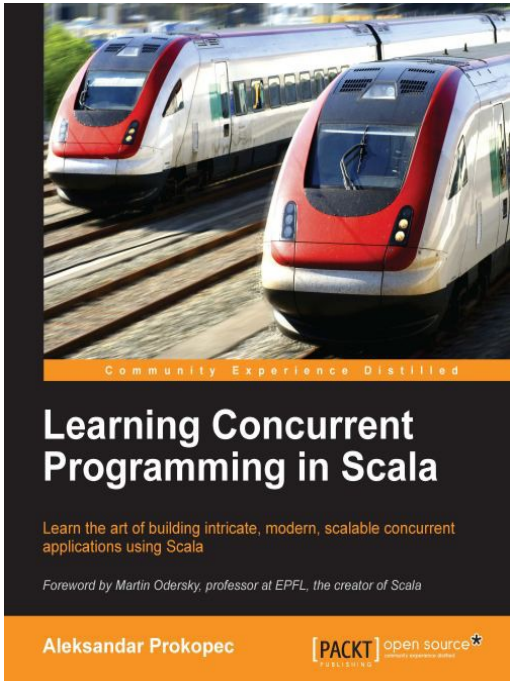


- Problem: write a program that, given a sequence of digits, returns the corresponding sentences (i.e. sequences of corresponding words taken from a given dictionary)
  - Ex. `7225247386` can generate `scala is fun`

```scala
import scala.io.Source

val in = Source.fromURL("http://cs.unibo.it/zavattar/words.txt")
val words = in.getLines.toList filter (w => w forall (c => c.isLetter))

val mnem = Map('2'->"ABC", '3'->"DEF", '4'->"GHI", '5'->"JKL",
  '6'->"MNO", '7'->"PQRS",'8'->"TUV", '9'->"WXYZ")

val charCode: Map[Char, Char] =
  for {
    (digit, str) <- mnem
    ltr <- str
  } yield ltr -> digit

def wordCode (word: String): String =
  word.toUpperCase map charCode

val wordsForNum: Map[String, Seq[String]] =
  (words groupBy wordCode) withDefaultValue Seq()

def encode(number: String): Set[List[String]] =
  if (number.isEmpty) Set(List())
  else (
    for {
      split <- 1 to number.length
      word <- wordsForNum(number take split)
      rest <- encode(number drop split)
    } yield word :: rest
    ).toSet
```

Concurrent programming in Scala

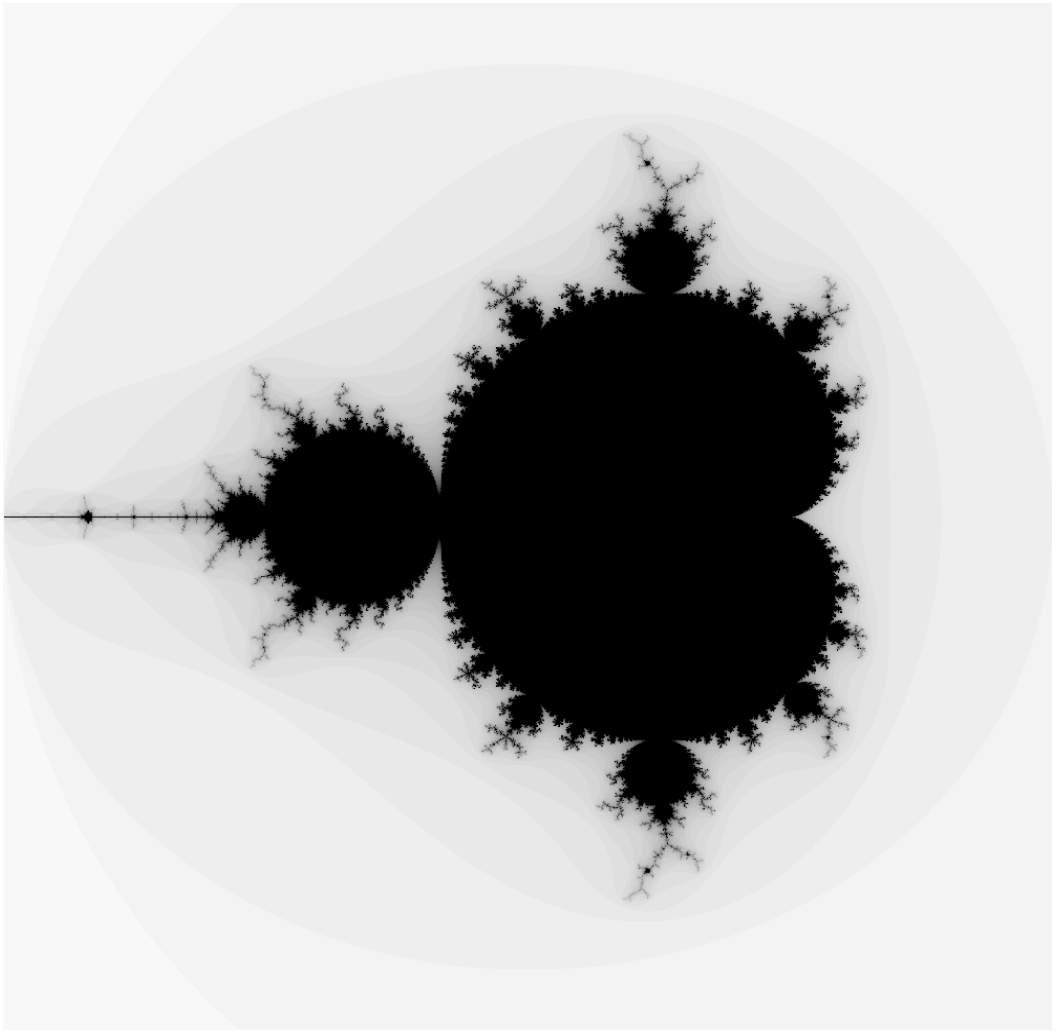
- Scala libraries also provide parallel collections:
  - map-like higher-order functions (acting pointwise on collections) can execute the “local” activities in parallel
  - this kind of parallelism is managed transparently:
    - Not explicitly managed by the programmer, but implemented in the parallel collection library



Third example

- Mandelbrot set image:
  - Mandelbrot set = $\{c \mid ..(..((c)^2+c)^2+c)^2..+c)^2.. \text{ is finite in abs value } \}$

    (considering complex numbers)
  - The graphical representation of the elements of the Mandelbrot set is a fractal (black part in the image)



```scala
import java.io._

class Complex(val a: Double, val b: Double){
  def +(that: Complex) = new Complex(this.a+that.a,this.b+that.b)
  def *(that: Complex) = new Complex(
    this.a*that.a-this.b*that.b,this.a*that.b+that.a*this.b)
  def abs() = Math.sqrt(this.a*this.a + this.b*this.b)
}

val fileName =
  "/Users/zavattar/IdeaProjects/IntroductionToScala/scalaimage.pgm"

def run(n:Int, level:Int) : Unit = {
  val out = new FileOutputStream(fileName)
  out.write(("P5\n"+n+" "+n+"\n255\n").getBytes())
  for (j <- (0 until n*n))
  { val x = -2.0 + (j%n)*3.0/n
    val y = -1.5 + (j/n)*3.0/n
    var z = new Complex(0,0)
    var c = new Complex(x,y)
    var i = 0
    while (z.abs < 2 && i < level) {z = z*z + c; i=i+1}
    out.write(255*(level-i)/level) }
  out.close()
}

run(1000,50)
```

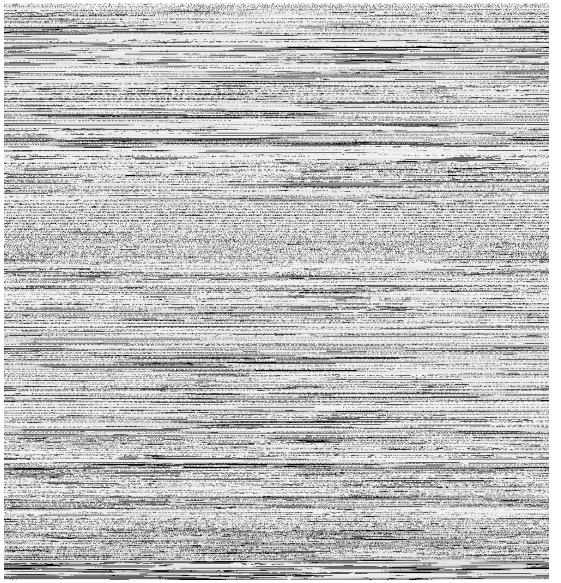
sequential version

```scala
import java.io._

class Complex(val a: Double, val b: Double){
  def +(that: Complex) = new Complex(this.a+that.a,this.b+that.b)
  def *(that: Complex) = new Complex(
    this.a*that.a-this.b*that.b,this.a*that.b+that.a*this.b)
  def abs() = Math.sqrt(this.a*this.a + this.b*this.b)
}

val fileName =
  "/Users/zavattar/IdeaProjects/IntroductionToScala/scalaimage.pgm"

def run(n:Int, level:Int) : Unit = {
  val out = new FileOutputStream(fileName)
  out.write(("P5\n"+n+" "+n+"\n255\n").getBytes())
  for (j <- (0 until n*n).par)
  { val x = -2.0 + (j%n)*3.0/n
    val y = -1.5 + (j/n)*3.0/n
    var z = new Complex(0,0)
    var c = new Complex(x,y)
    var i = 0
    while (z.abs < 2 && i < level) {z = z*z + c; i=i+1}
    out.write(255*(level-i)/level) }
  out.close()
}

run(1000,50)
```

parallel version

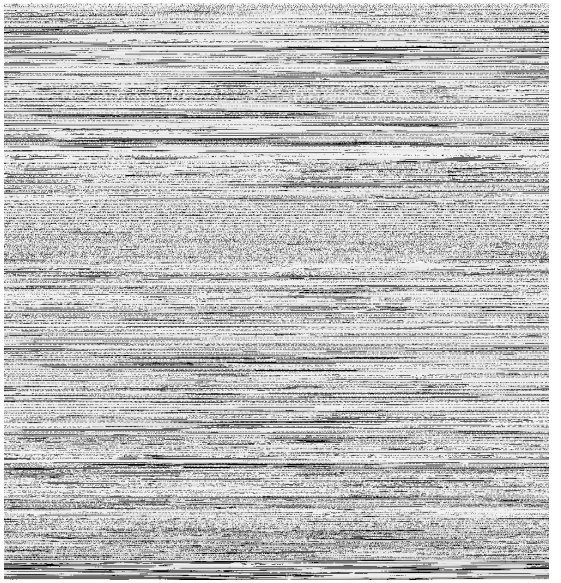
Parallel version (revisited)

- What is wrong with the parallel version?
  - Pixels are not in their expected position!



Parallel version (revisited)

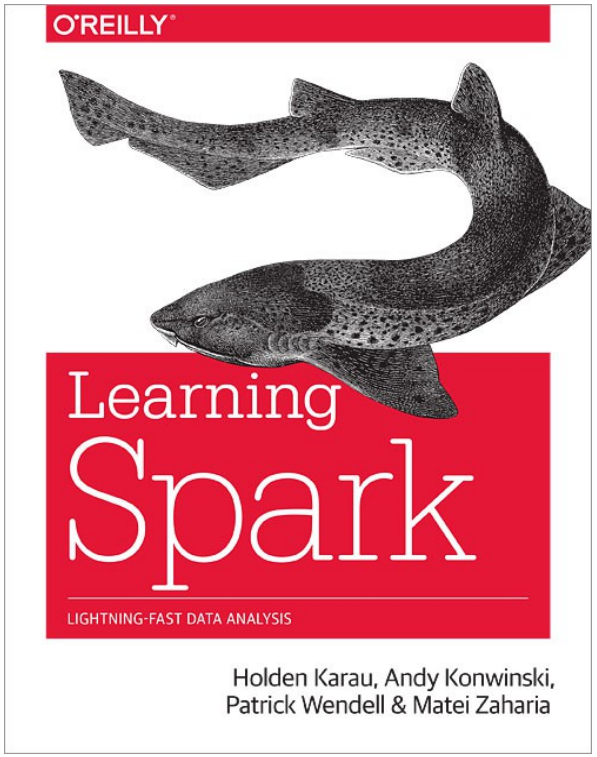
- What is wrong with the parallel version?
  - Pixels are not in their expected position!
  - Here is a revised parallel version:



```
def run(n:Int, level:Int) : Unit = {
  val out = new FileOutputStream(fileName)
  out.write(("P5\n"+n+" "+n+"\n255\n").getBytes())
  var a=new Array[Int](n * n)
  for (j <- (0 until n*n).par)
  { val x = -2.0 + (j%n)*3.0/n
    val y = -1.5 + (j/n)*3.0/n
    var z = new Complex(0,0)
    var c = new Complex(x,y)
    var i = 0
    while (z.abs < 2 && i < level) {z = z*z + c; i=i+1}
    a(j) = 255*(level-i)/level }
  for{k <- 0 until n*n} out.write(a(k))
  out.close()
}
```

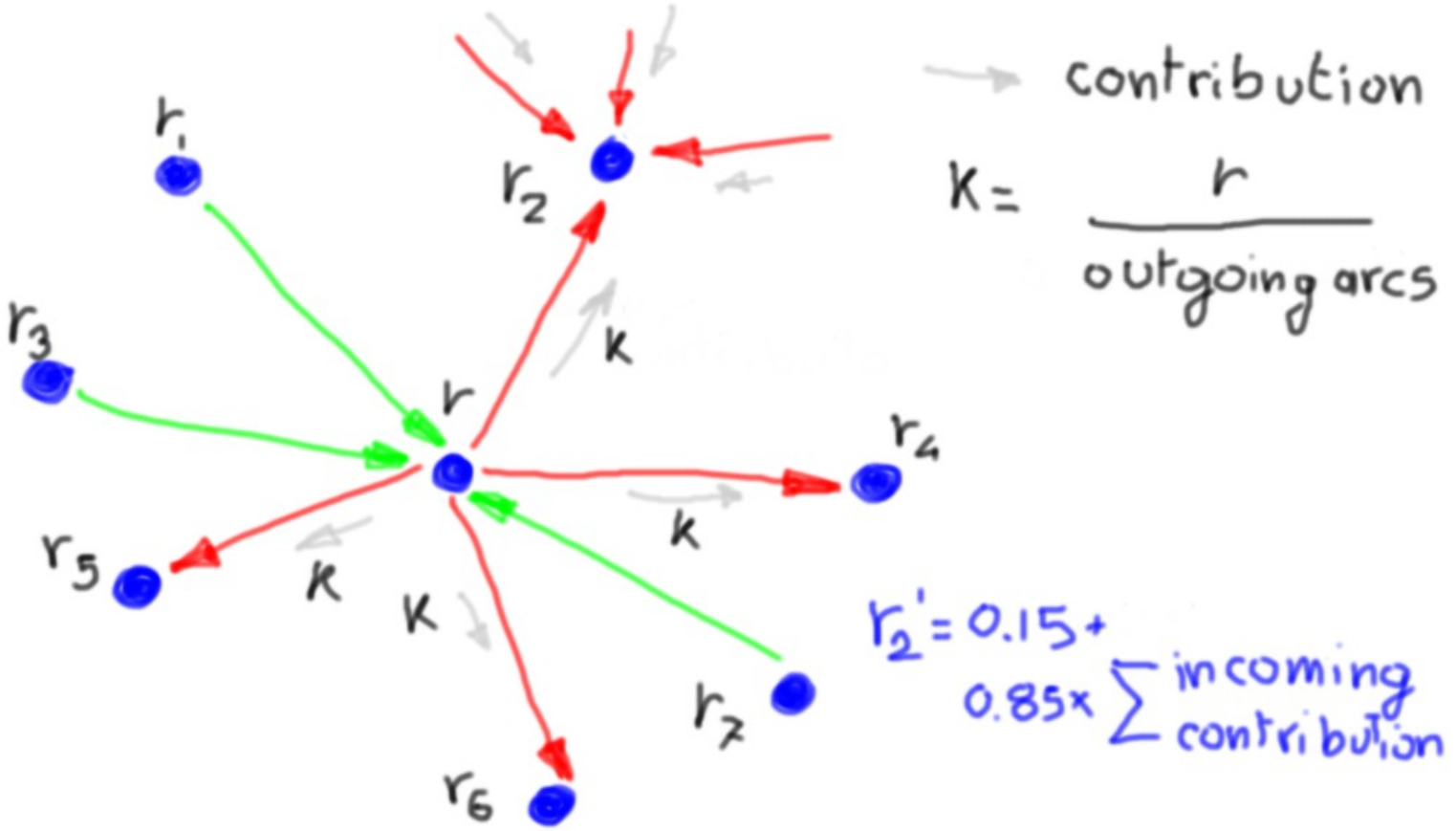
Spark

- Spark is a large-scale data processing framework
  - like other MapReduce platforms
  - but with an API which is almost 1-to-1 with Scala's collections
- We will see:
  - Spark's programming model
    - similar to Scala programming but..
    - ..you must know some Spark internals to avoid performance issues



Fourth example

- Page-rank: iterative ranking algorithm used by Google
  - At each iteration, a node sends contributions to its neighbors, and updates its rank depending on the incoming contributions



```scala
import org.apache.spark.SparkContext, org.apache.spark.SparkConf,
org.apache.commons.io.FileUtils, org.apache.spark.HashPartitioner, java.io._

object PageRank {
  def main(args: Array[String]) {
    val conf = new SparkConf().setAppName("pageRank").setMaster("local[*]")
    val sc = new SparkContext(conf)
    val inputFile = "/Users/zavattar/IdeaProjects/PageRank/soc_Epinions.txt"
    val outputFile = "/Users/zavattar/IdeaProjects/PageRank/pageRank"
    val input = sc.textFile(inputFile)
    val edges = input.map(s => (s.split("\t"))).
      map(a => (a(0).toInt,a(1).toInt))
    val links = edges.groupByKey().
      partitionBy(new HashPartitioner(4)).persist()
    var ranks = links.mapValues(v => 1.0)

    for(i <- 0 until 10) {
      val contributions = links.join(ranks).flatMap {
        case (u, (uLinks, rank)) =>
          uLinks.map(t => (t, rank / uLinks.size))
      }
      ranks = contributions.reduceByKey((x,y) => x+y).
        mapValues(v => 0.15+0.85*v)
    }

    FileUtils.deleteDirectory(new File(outputFile))
    ranks.saveAsTextFile(outputFile)
    sc.stop()
  }
}
```

Scala-Spark in the Cloud

- Big data Scala-Spark programs are rarely executed on a single node
- They are usually executed on a Spark cluster, possibly deployed in the cloud
- The main cloud providers offer services to easily acquire computing resources and execute on them Scala-Spark programs
  - GCP Dataproc, AWS EMR or Azure HDInsight
- The course will include:
  - Seminars on “introduction to cloud computing”
  - Presentation of cloud services useful to executed Scala-Spark programs

Google Cloud

Microsoft Azure

Exam

- Divided in two parts
  - Written exam:
    - Open-ended questions on the course contents
    - Dates, registration and evaluations on AlmaEsami
  - Project work:
    - Implement a Scala-Spark program and experiment its execution in the cloud (most probably Google Cloud Platform - GCP)
    - Project specifications will be published close to the end of the lectures ("Scala-Spark in the cloud" will be discussed during the last lectures)
    - Single (not group) activity
    - Submission via GitHub of code, documentation, deployment/execution scripts, ... and a short document describing the project activity
    - Project activities followed by the tutor (giuseppe.depalma2@unibo.it)
  - Final grade:
    - Weighted mean between the two evaluations (written exam has double weight w.r.t. project work)