2.4
Introduction to Scala

From the point of view of Spark

So, what actually *is* Spark?

- Spark is
  - A fast, general-purpose cluster-computing platform
  - Based on map/reduce paradigm
    - In that sense, similar to Hadoop.
  - Optimized for clusters with lots of memory
  - An open-source Apache project
  - Part of the JDK eco-system
  - Implemented in Scala

What would we need to implement such a framework?

- A cluster manager (execution system);
- An optimizing execution planner;
- A partitionable lazily-evaluated collection type;
- A way to compose and to serialize functions.

Details?

- Cluster manager: External (YARN, Mesos, Kubernetes, stand-alone, etc.)
- Execution planner: Internal (map/reduce-like); optimizable for some jobs, e.g. SQL
- Why do we say “Partitionable and lazily-evaluated collection type?”
    - More detail in next slide.
- A way to compose and to serialize functions.
    - More detail in subsequent slide.

Why do we say “Partitionable and lazily-evaluated collection type?”

- Unlike Hadoop, which tends to do each map/reduce as a separate task, with persistence and replication to the HDFS, the Spark execution planner builds a DAG (directed, acyclic graph) of stages, the whole of which is implemented in one remote “job.”
- In Spark, the map/reduce pipeline that corresponds to one “action” is represented by a directed acyclic graph of map and reduce stages.
- The map stage may actually invoke many different transformations, each represented by a function, and which can be *composed* together into a single transformation for “lazy*” evaluation.
- A stage is broken up into tasks, typically one per partition.
- The data structure which enables these operations is called a Resilient Distributed Dataset (RDD)

** lazy evaluation is an important aspect of functional programming: delaying evaluation until absolutely necessary.*

A way to compose and to serialize functions.

- Composing functions
  - Each task is physically executed with the appropriate partition, the appropriate function, and on the appropriate executor (i.e. worker node). We therefore prefer to compose functions together if possible to avoid extra performance overhead.
  - For example, suppose we wish to transform a partition with a function which doubles the value and then we wish to transform the same partition with a function which increments the result. Instead we can *compose* these two functions into one function of form *x => x * 2 + 1*
- Closures
  - A “closure” is a function which is partially applied so that all variables, except the one(s) expected as input, are “captured”, i.e. bound to values
  - For example, suppose we wish to transform a partition with a function which multiplies the value by a factor *f* and adds a constant *k* to the result: we will then need a function like: *x => x * f + k.*
  - But in order for this to be usable out of context (i.e. on a remote node), we must capture the values *f* and *k*. This is what a closure does.
- Serializing functions
  - These functions will do us no good at all if we can’t send them, like ordinary objects, over the network as part of the task definition.
  - We therefore need a way to serialize these functions. But the driver and the executors are all running the JVM which means that they all understand byte code. Pure functions can be serialized and deserialized as byte code!

An example Spark job

```
Robins-MacBook-Pro:spark-2.2.0-bin-hadoop2.7 scalaprof$ bin/spark-shell
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
18/01/02 20:28:25 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes
where applicable
18/01/02 20:28:31 WARN ObjectStore: Failed to get database global_temp, returning NoSuchObjectException
Spark context Web UI available at http://172.20.20.20:4040
Spark context available as 'sc' (master = local[*], app id = local-1514942907077).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.2.0
      /_/

Using Scala version 2.11.8 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_144)
Type in expressions to have them evaluated.
Type :help for more information.

scala> val rdd = sc.textFile("/Users/scalaprof/flatland.txt")
rdd: org.apache.spark.rdd.RDD[String] = /Users/scalaprof/flatland.txt MapPartitionsRDD[1] at textFile at <console>:24

scala> rdd.flatMap(_.split(" ")).map((_,1)).reduceByKey((x, y) => x + y).sortBy(-_._2).take(10).foreach (println _)
(the,15)
(of,14)
(and,10)
(a,9)
(to,9)
(will,8)
(you,8)
(have,6)
(at,5)
(I,5)

scala> :quit
```

Summary

- What we need to implement Spark:
  - lazy evaluation
  - functional composition
  - closures
  - serializable functions
  - JVM
- These all come built-in with **Scala** but not with Java (at least not until very recently).
- For another (better) take on this, please refer to: A Newbie's guide to Scala…