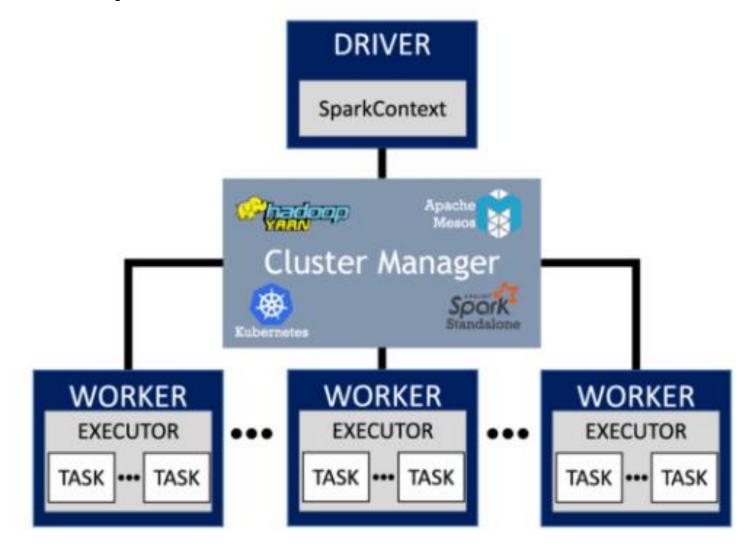
Systems, paradigms and algorithms for Big Data TD 1

Why Spark?

- Ease creation of complicated data analysis task
- Support iterative algorithms, like gradient descent
- Ease data exploration
- Make it easier to explore data interactively
 - Developer friendly, no need to create tons of classes and jobs...
 - Repl mode
 - Scala/Python API similar to functional programming

Main Components



How to run it

- Repl : Spark-shell, notebooks → exploratory mode
- Spark-submit: runs a script → scheduled job
- Api for Java, Scala, Python.
- Deploy Mode: client vs cluster

Dependencies

- Depends on Hadoop Libraries (HDFS and Yarn)
- Requires a Java Runtime Environment (need Java in your system path)

RDD

Resilient Distributed Dataset

- The base building block of your application
- Lazy: doesn't compute anything before it really needs to do so.
- Immutable: a transformation doesn't change the data set, it returns a new RDD.
- Fault Tolerant: partition can be recomputed in case of failure

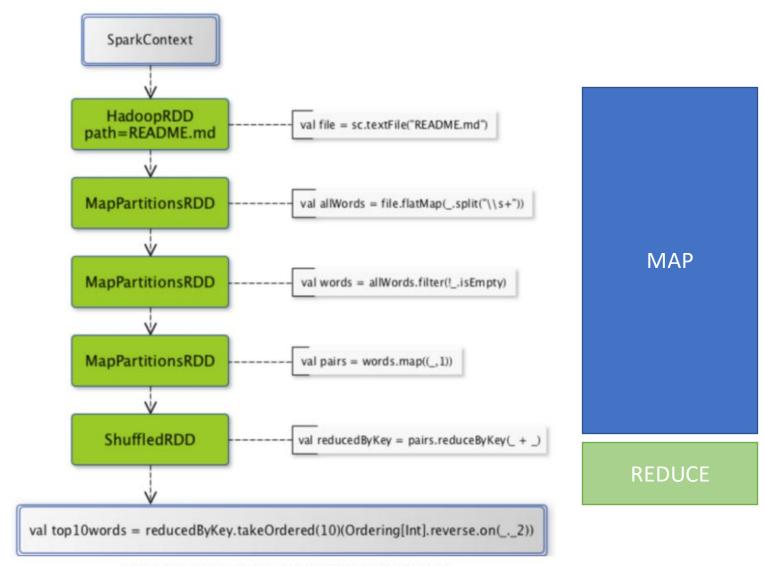
Graph

- Conceptually, an RDD is a node of a graph of functions.
- API is pretty much similar with **Functional Programming. Forget the for-loop.**

Inside the RDD:

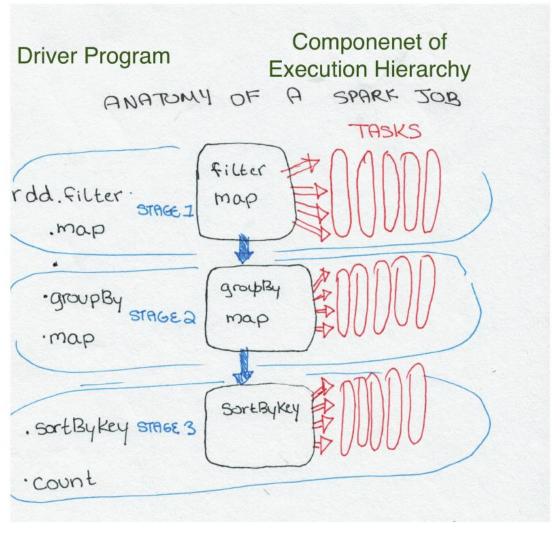
- Knows the partitions he is working-on, how data is partitionned
- Knows how to iterate over each partition to yield records
- Know RDD's it depends-on

RDD as a Directed Acyclic Graph



New RDD is created after every transformation.(DAG graph)

Jobs, Stages, Tasks



Source : high performance Spark

RDD API - Actions

RDD {"user":"John", "movie":"Blade Runner", "rating":5.0} count() 3 {"user":"Louise", "movie":"Dirty dancing", "rating":5.0} {"user": "Sam", "movie": "Blade Runner", "rating": 3.5} **RDD** list {"user":"John"} take(2) {"user":"John"} {"user":"Louise"} {"user":"Louise"} {"user":"Sam"} **RDD** list {"user":"John"} {"user":"John"} collect() {"user":"Sam"} {"user":"Sam"}

RDD API - Transformations

RDD RDD map(lambda x: {"user":"John", "movie":"Blade Runner", "rating":5.0} "Blade Runner" x['movie']) {"user":"Louise", "movie":"Dirty dancing", "rating":5.0} "Dirty Dancing" {"user": "Sam", "movie": "Blade Runner", "rating": 3.5} "Blade Runner" **RDD** RDD of (K, V) pairs keyBy(lambda x: ("John", ...) {"user":"John", "movie":"Blade Runner", "rating":5.0} x['user']) {"user":"Louise", "movie":"Dirty dancing", "rating":5.0} ("Louise", ...) {"user": "Sam", "movie": "Blade Runner", "rating": 3.5} ("Sam", ...) RDD of (K, V) pairs RDD of (K, V) pairs mapValues(lambda x: ("John", "Blade Runner") ("John", 12) len(x[1])) ("Louise", "Dirty dancing") ("Louise", 13) ("Sam", "Blade Runner") ("Sam", 12)

RDD API - Transformations

RDD

{"movie":"Blade Runner", "genres":"cyberpunk;scifi;action"}

{"movie":"Dirty dancing", "genres":"music;danse;romance"}

flatmap(lambda x:
x['genres'].split(';'))

RDD

"cyberpunk"

"scifi"

"action"

music"

"danse"

"romance"

RDD

{"user":"John", "movie":"Blade Runner", "rating":5.0}

{"user":"Louise", "movie":"Dirty dancing", "rating":5.0}

{"user": "Sam", "movie": "Blade Runner", "rating": 3.5}

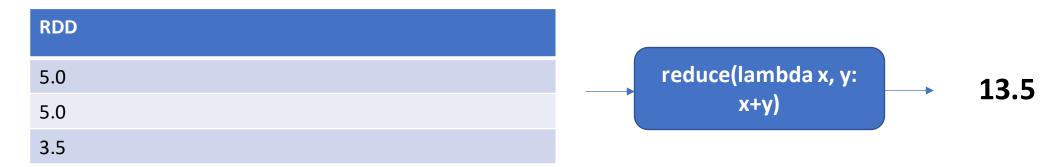
filter(lambda x: x['rating']>4.0)

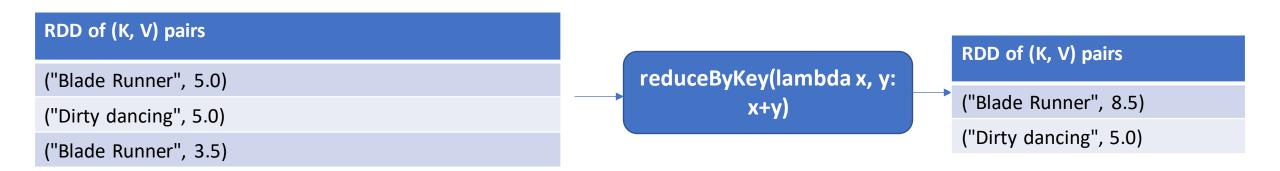
RDD

{"user":"John", "movie":"Bl...

{"user":"Louise", "movie":"Dir...

RDD API – Actions





Other useful functions

- Join (shuffle ? Lazy ?)
- Sample
- mappartitions
- zippartitions

Links

- https://spark.apache.org/docs/latest/api/scala/org/apache/spark/api/java/JavaPairRDD.html
- https://0x0fff.com/hadoop-mapreduce-comprehensive-description/