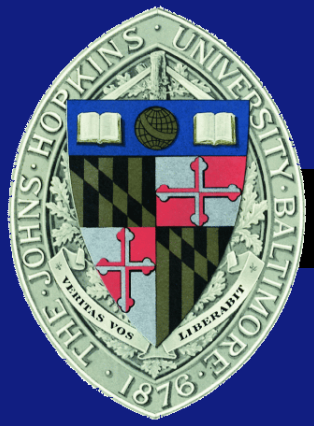
Lecture 16.1
Spark and RDDs

EN 600.320/420

Instructor: Randal Burns

5 April 2017



Department of Computer Science, *Johns Hopkins University*

Spark: Batch Computing Reload

- Map/Reduce style programming
  - Data-parallel, batch, restrictive model, functional
  - Abstractions to leverage distributed memory
- New interfaces to in-memory computations
  - Fault-tolerant
  - Lazy materialization (pipelined evaluation)
- Good support for *iterative computations* on *in-memory* data sets leads to good performance
  - 20x over Map/Reduce
  - No writing data to file system, loading data from file system

Lecture derived from: Zaharia et al. *Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing*. USENIX NSDI, 2012:

RDD: Resilient Distributed Dataset

- Read-only partitioned collection of records
- Created from:
  - Data in stable storage
  - Transformations on other RDDs
- Unit of parallelism in a data decomposition:
  - Automatic parallelization of transformation, such as *map, reduce, filter,* etc.
- RDDs are not data:
  - Not materialized. They are an abstraction.
  - Defined by lineage. Set of transformations on a original data set.

RDD Lineage

- Lines backed by HDFS
- Errors – filtered lines
- Times—collected makes real data

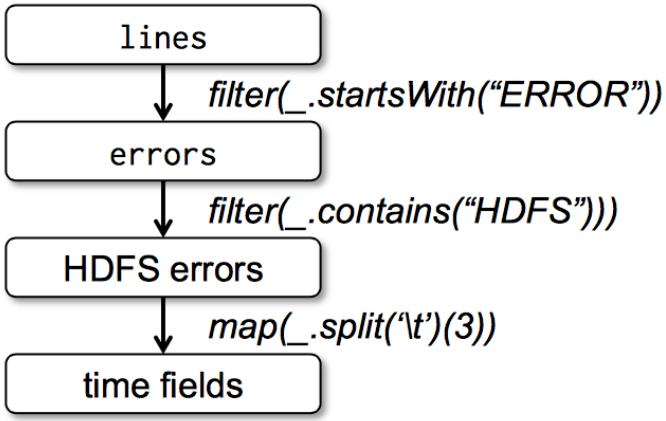


Figure 1: Lineage graph for the third query in our example. Boxes represent RDDs and arrows represent transformations.

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
errors.persist()

// Return the time fields of errors mentioning
// HDFS as an array (assuming time is field
// number 3 in a tab-separated format):
errors.filter(_.contains("HDFS"))
      .map(_.split('\t')(3))
      .collect()
```

Logistical Regression: A First Example:

```
val points = spark.textFile(...)
                  .map(parsePoint).persist()
var w = // random initial vector
for (i <- 1 to ITERATIONS) {
  val gradient = points.map{ p =>
    p.x * (1/(1+exp(-p.y*(w dot p.x)))-1)*p.y
  }.reduce((a,b) => a+b)
  w -= gradient
}
```

- Features:
  - Scala closures (on w), functions with free variables
  - points is a read-only RDD for each iteration
  - Only w (a scalar) gets updated

Managing the State of Data

- persist(): indicates desire to reuse an RDD, encourages Spark to keep it in memory
- RDD(): the representation of a logical data set
- sequence: a physical, materialized data set
- In Spark-land, RDDs and sequences are differentiated by the concepts of
  - Transformations: RDD->RDD
  - Actions: RDD->sequence/data
- RDDs define a pipeline of computations from data set (HDFS) to sequence/data
  - RDDs evaluated lazily as needed to build a sequence

Transformations and Actions

- Parallelized constructs in Spark
  - Transformations are lazy whereas actions launch computation

Transformations	$map(f : \mathrm{T} \Rightarrow \mathrm{U})$	:	RDD[T] $\Rightarrow$ RDD[U]
	$filter(f : \mathrm{T} \Rightarrow \mathrm{Bool})$	:	RDD[T] $\Rightarrow$ RDD[T]
	$flatMap(f : \mathrm{T} \Rightarrow \mathrm{Seq[U]})$	:	RDD[T] $\Rightarrow$ RDD[U]
	$sample(fraction : \mathrm{Float})$	:	RDD[T] $\Rightarrow$ RDD[T] (Deterministic sampling)
	$groupByKey()$	:	RDD[(K, V)] $\Rightarrow$ RDD[(K, Seq[V])]
	$reduceByKey(f : (\mathrm{V}, \mathrm{V}) \Rightarrow \mathrm{V})$	:	RDD[(K, V)] $\Rightarrow$ RDD[(K, V)]
	$union()$	:	(RDD[T], RDD[T]) $\Rightarrow$ RDD[T]
	$join()$	:	(RDD[(K, V)], RDD[(K, W)]) $\Rightarrow$ RDD[(K, (V, W))]
	$cogroup()$	:	(RDD[(K, V)], RDD[(K, W)]) $\Rightarrow$ RDD[(K, (Seq[V], Seq[W]))]
	$crossProduct()$	:	(RDD[T], RDD[U]) $\Rightarrow$ RDD[(T, U)]
	$mapValues(f : \mathrm{V} \Rightarrow \mathrm{W})$	:	RDD[(K, V)] $\Rightarrow$ RDD[(K, W)] (Preserves partitioning)
	$sort(c : \mathrm{Comparator[K]})$	:	RDD[(K, V)] $\Rightarrow$ RDD[(K, V)]
	$partitionBy(p : \mathrm{Partitioner[K]})$	:	RDD[(K, V)] $\Rightarrow$ RDD[(K, V)]
Actions	$count()$	:	RDD[T] $\Rightarrow$ Long
	$collect()$	:	RDD[T] $\Rightarrow$ Seq[T]
	$reduce(f : (\mathrm{T}, \mathrm{T}) \Rightarrow \mathrm{T})$	:	RDD[T] $\Rightarrow$ T
	$lookup(k : \mathrm{K})$	:	RDD[(K, V)] $\Rightarrow$ Seq[V] (On hash/range partitioned RDDs)
	$save(path : \mathrm{String})$	:	Outputs RDD to a storage system, *e.g.*, HDFS

Table 2: Transformations and actions available on RDDs in Spark. Seq[T] denotes a sequence of elements of type T.