



ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Resilient Distributed Datasets(RDDs), Spark's Distributed Collections

Big Data Analysis with Scala and Spark

Heather Miller

Resilient Distributed Datasets (RDDs)

RDDs seem a lot like *immutable* sequential or parallel Scala collections.

Resilient Distributed Datasets (RDDs)

RDDs seem a lot like *immutable* sequential or parallel Scala collections.

```
abstract class RDD[T] {  
  def map[U](f: T => U): RDD[U] = ...  
  def flatMap[U](f: T => TraversableOnce[U]): RDD[U] = ...  
  def filter(f: T => Boolean): RDD[T] = ...  
  def reduce(f: (T, T) => T): T = ...  
}
```

Resilient Distributed Datasets (RDDs)

RDDs seem a lot like *immutable* sequential or parallel Scala collections.

```
abstract class RDD[T] {  
  def map[U](f: T => U): RDD[U] = ...  
  def flatMap[U](f: T => TraversableOnce[U]): RDD[U] = ...  
  def filter(f: T => Boolean): RDD[T] = ...  
  def reduce(f: (T, T) => T): T = ...  
}
```

Most operations on RDDs, like Scala's immutable List, and Scala's parallel collections, are higher-order functions.

That is, methods that work on RDDs, taking a function as an argument, and which typically return RDDs.

Resilient Distributed Datasets (RDDs)

RDDs seem a lot like *immutable* sequential or parallel Scala collections.

Resilient Distributed Datasets (RDDs)

RDDs seem a lot like *immutable* sequential or parallel Scala collections.

Combinators on Scala parallel/sequential collections:

map
flatMap
filter
reduce
fold
aggregate

Combinators on RDDs:

map
flatMap
filter
reduce
fold
aggregate

Resilient Distributed Datasets (RDDs)

While their signatures differ a bit, their semantics (macroscopically) are the same:

```
map[B](f: A => B): List[B] // Scala List
```

```
map[B](f: A => B): RDD[B] // Spark RDD
```

```
flatMap[B](f: A => TraversableOnce[B]): List[B] // Scala List
```

```
flatMap[B](f: A => TraversableOnce[B]): RDD[B] // Spark RDD
```

```
filter(pred: A => Boolean): List[A] // Scala List
```

```
filter(pred: A => Boolean): RDD[A] // Spark RDD
```

Resilient Distributed Datasets (RDDs)

While their signatures differ a bit, their semantics (macroscopically) are the same:

```
reduce(op: (A, A) => A): A // Scala List
```

```
reduce(op: (A, A) => A): A // Spark RDD
```

```
fold(z: A)(op: (A, A) => A): A // Scala List
```

```
fold(z: A)(op: (A, A) => A): A // Spark RDD
```

```
aggregate[B](z: => B)(seqop: (B, A) => B, combop: (B, B) => B): B // Scala
```

```
aggregate[B](z: B)(seqop: (B, A) => B, combop: (B, B) => B): B // Spark RDD
```


Resilient Distributed Datasets (RDDs)

Using RDDs in Spark feels a lot like normal Scala sequential/parallel collections, with the added knowledge that your data is distributed across several machines.

Example:

Given, `val encyclopedia: RDD[String]`, say we want to search all of encyclopedia for mentions of EPFL, and count the number of pages that mention EPFL.

Resilient Distributed Datasets (RDDs)

Using RDDs in Spark feels a lot like normal Scala sequential/parallel collections, with the added knowledge that your data is distributed across several machines.

Example:

Given, `val encyclopedia: RDD[String]`, say we want to search all of encyclopedia for mentions of EPFL, and count the number of pages that mention EPFL.

```
val result = encyclopedia.filter(page => page.contains("EPFL"))  
                           .count()
```

Example: Word Count

The “Hello, World!” of programming with large-scale data.

```
// Create an RDD RDD[String]  
val rdd = spark.textFile("hdfs://...")
```

```
val count = ???
```

Example: Word Count

The “Hello, World!” of programming with large-scale data.

```
// Create an RDD
```

```
val rdd = spark.textFile("hdfs://...")
```

```
val count = rdd.flatMap(line => line.split(" ")) // separate lines into words
```

RDD[String] ← words

Example: Word Count

The “Hello, World!” of programming with large-scale data.

```
// Create an RDD
```

```
val rdd = spark.textFile("hdfs://...")
```

```
val count = rdd.flatMap(line => line.split(" ")) // separate lines into words  
                  .map(word => (word, 1))         // include something to count
```

Example: Word Count

The “Hello, World!” of programming with large-scale data.

```
// Create an RDD
```

```
val rdd = spark.textFile("hdfs://...")
```

```
val count = rdd.flatMap(line => line.split(" ")) // separate lines into words  
                  .map(word => (word, 1))           // include something to count  
                  .reduceByKey(_ + _)              // sum up the 1s in the pairs
```

That's it.

How to Create an RDD?

RDDs can be created in two ways:

How to Create an RDD?

RDDs can be created in two ways:

- ▶ Transforming an existing RDD.
- ▶ From a SparkContext (or SparkSession) object.

How to Create an RDD?

RDDs can be created in two ways:

- ▶ **Transforming an existing RDD.**
- ▶ From a `SparkContext` (or `SparkSession`) object.

Transforming an existing RDD.

Just like a call to `map` on a `List` returns a new `List`, many higher-order functions defined on `RDD` return a new `RDD`.


How to Create an RDD?

RDDs can be created in two ways:

- ▶ Transforming an existing RDD.
- ▶ **From a SparkContext (or SparkSession) object.**



From a SparkContext (or SparkSession) object.



The SparkContext object (renamed SparkSession) can be thought of as your handle to the Spark cluster. It represents the connection between the Spark cluster and your running application. It defines a handful of methods which can be used to create and populate a new RDD:

- ▶ parallelize: convert a local Scala collection to an RDD.
- ▶ textFile: read a text file from HDFS or a local file system and return an RDD of String