cloudera®

Working With RDDs in Spark

Chapter 11

201509

Course Chapters

Working With RDDs

In this chapter you will learn

- **How RDDs are created from files or data in memory**
- **How to handle file formats with multi-line records**
- **How to use some additional operations on RDDs**

Chapter Topics

Working With RDDs in Spark	Distributed Data Processing with Spark

- **Creating RDDs**
- Other General RDD Operations
- Conclusion
- Homework: Process Data Files with Spark

RDDs

- **RDDs can hold any type of element**
  - Primitive types: integers, characters, booleans, etc.
  - Sequence types: strings, lists, arrays, tuples, dicts, etc. (including nested data types)
  - Scala/Java Objects (if serializable)
  - Mixed types
- **Some types of RDDs have additional functionality**
  - Pair RDDs
    - RDDs consisting of Key-Value pairs
  - Double RDDs
    - RDDs consisting of numeric data

Creating RDDs From Collections

- **You can create RDDs from collections instead of files**
  - `sc.parallelize(`*`collection`*`)`

```
> myData = ["Alice","Carlos","Frank","Barbara"]
> myRdd = sc.parallelize(myData)
> myRdd.take(2)
['Alice', 'Carlos']
```

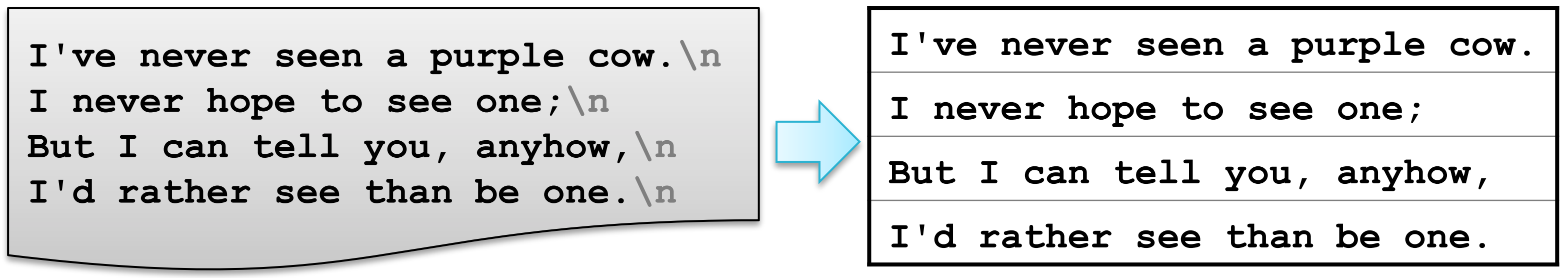
- **Useful when**
  - Testing
  - Generating data programmatically
  - Integrating

Creating RDDs from Files (1)

- **For file-based RDDs, use `SparkContext.textFile`**
  - Accepts a single file, a wildcard list of files, or a comma-separated list of files
  - Examples
    - `sc.textFile("myfile.txt")`
    - `sc.textFile("mydata/*.log")`
    - `sc.textFile("myfile1.txt,myfile2.txt")`
  - Each line in the file(s) is a separate record in the RDD
- **Files are referenced by absolute or relative URI**
  - Absolute URI:
    - `file:/home/training/myfile.txt`
    - `hdfs://localhost/loudacre/myfile.txt`
  - Relative URI (uses default file system): `myfile.txt`

Creating RDDs from Files (2)

- **`textFile` maps each line in a file to a separate RDD element**



```
I've never seen a purple cow.\n
I never hope to see one;\n
But I can tell you, anyhow,\n
I'd rather see than be one.\n
```

```
I've never seen a purple cow.
I never hope to see one;
But I can tell you, anyhow,
I'd rather see than be one.
```

- **`textFile` only works with line-delimited text files**
- **What about other formats?**

Input and Output Formats (1)

- **Spark uses Hadoop `InputFormat` and `OutputFormat` Java classes**
  - Some examples from core Hadoop
    - **`TextInputFormat`** / **`TextOutputFormat`** – newline delimited text files
    - **`SequenceInputFormat`** / **`SequenceOutputFormat`**
    - **`FixedLengthInputFormat`**
  - Many implementations available in additional libraries
    - e.g. **`AvroInputFormat`** / **`AvroOutputFormat`** in the Avro library

Input and Output Formats (2)

- **Specify any input format using `sc.hadoopFile`**
  - or `newAPIhadoopFile` for New API classes
- **Specify any output format using `rdd.saveAsHadoopFile`**
  - or `saveAsNewAPIhadoopFile` for New API classes
- **`textFile` and `saveAsTextFile` are convenience functions**
  - `textFile` just calls `hadoopFile` specifying `TextInputFormat`
  - `saveAsTextFile` calls `saveAsHadoopFile` specifying `TextOutputFormat`

Whole File-Based RDDs (1)

- **`sc.textFile` maps each line in a file to a separate RDD element**
  - What about files with a multi-line input format, e.g. XML or JSON?
- **`sc.wholeTextFiles(`*`directory`*`)`**
  - Maps entire contents of each file in a directory to a single RDD element
  - Works only for small files (element must fit in memory)

file1.json

```
{
 "firstName":"Fred",
 "lastName":"Flintstone",
 "userid":"123"
}
```

file2.json

```
{
 "firstName":"Barney",
 "lastName":"Rubble",
 "userid":"234"
}
```

```
(file1.json,{"firstName":"Fred","lastName":"Flintstone","userid":"123"} )
(file2.json,{"firstName":"Barney","lastName":"Rubble","userid":"234"} )
(file3.xml,… )
(file4.xml,… )
```

Whole File-Based RDDs (2)

```
> import json
> myrdd1 = sc.wholeTextFiles(mydir)
> myrdd2 = myrdd1
  .map(lambda (fname,s): json.loads(s))
> for record in myrdd2.take(2):
>     print record["firstName"]
```

```
> import scala.util.parsing.json.JSON
> val myrdd1 = sc.wholeTextFiles(mydir)
> val myrdd2 = myrdd1
  .map(pair => JSON.parseFull(pair._2).get.
           asInstanceOf[Map[String,String]])
> for (record <- myrdd2.take(2))
    println(record.getOrElse("firstName",null))
```

Output:

```
Fred
Barney
```

Chapter Topics

Working With RDDs in Spark	Distributed Data Processing with Spark

- Creating RDDs
- **Other General RDD Operations**
- Conclusion
- Homework: Process Data Files with Spark

Some Other General RDD Operations

- **Single-RDD Transformations**
  - **`flatMap`** – maps one element in the base RDD to multiple elements
  - **`distinct`** – filter out duplicates
  - **`sortBy`** – use provided function to sort
- **Multi-RDD Transformations**
  - **`intersection`** – create a new RDD with all elements in both original RDDs
  - **`union`** – add all elements of two RDDs into a single new RDD
  - **`zip`** – pair each element of the first RDD with the corresponding element of the second

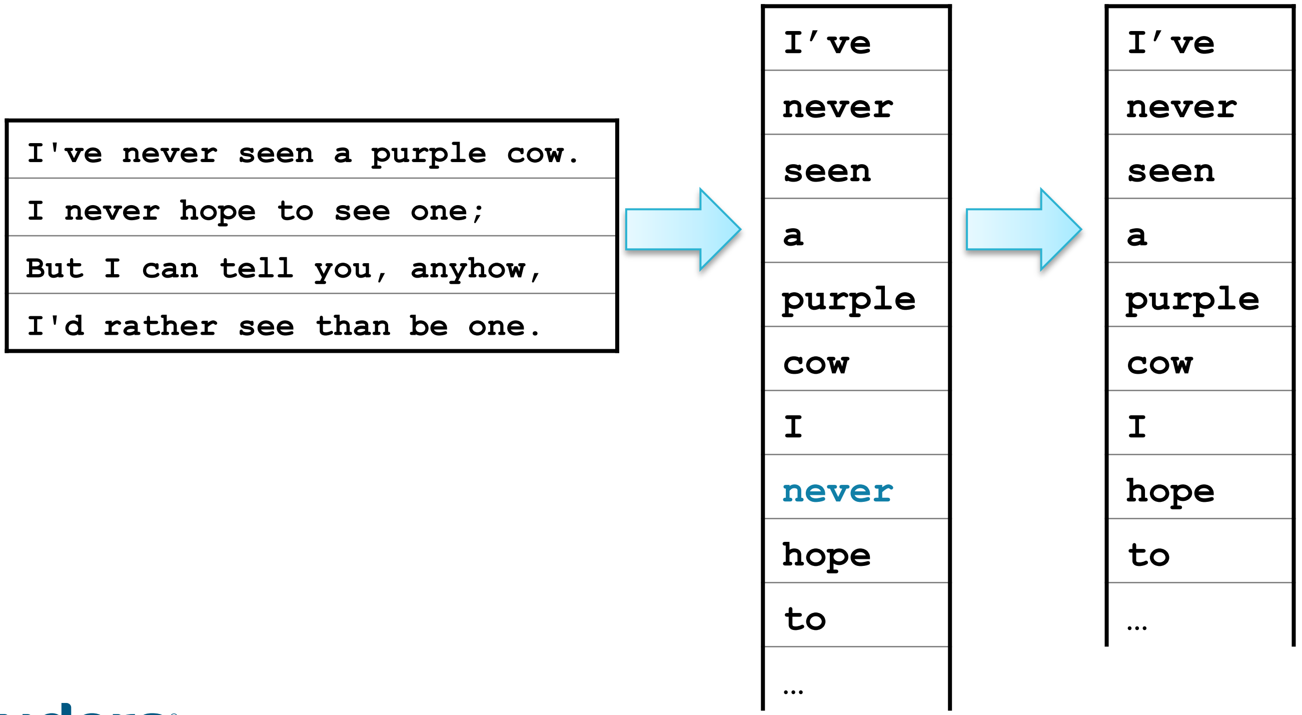
Example: **`flatMap`** and **`distinct`**

Python

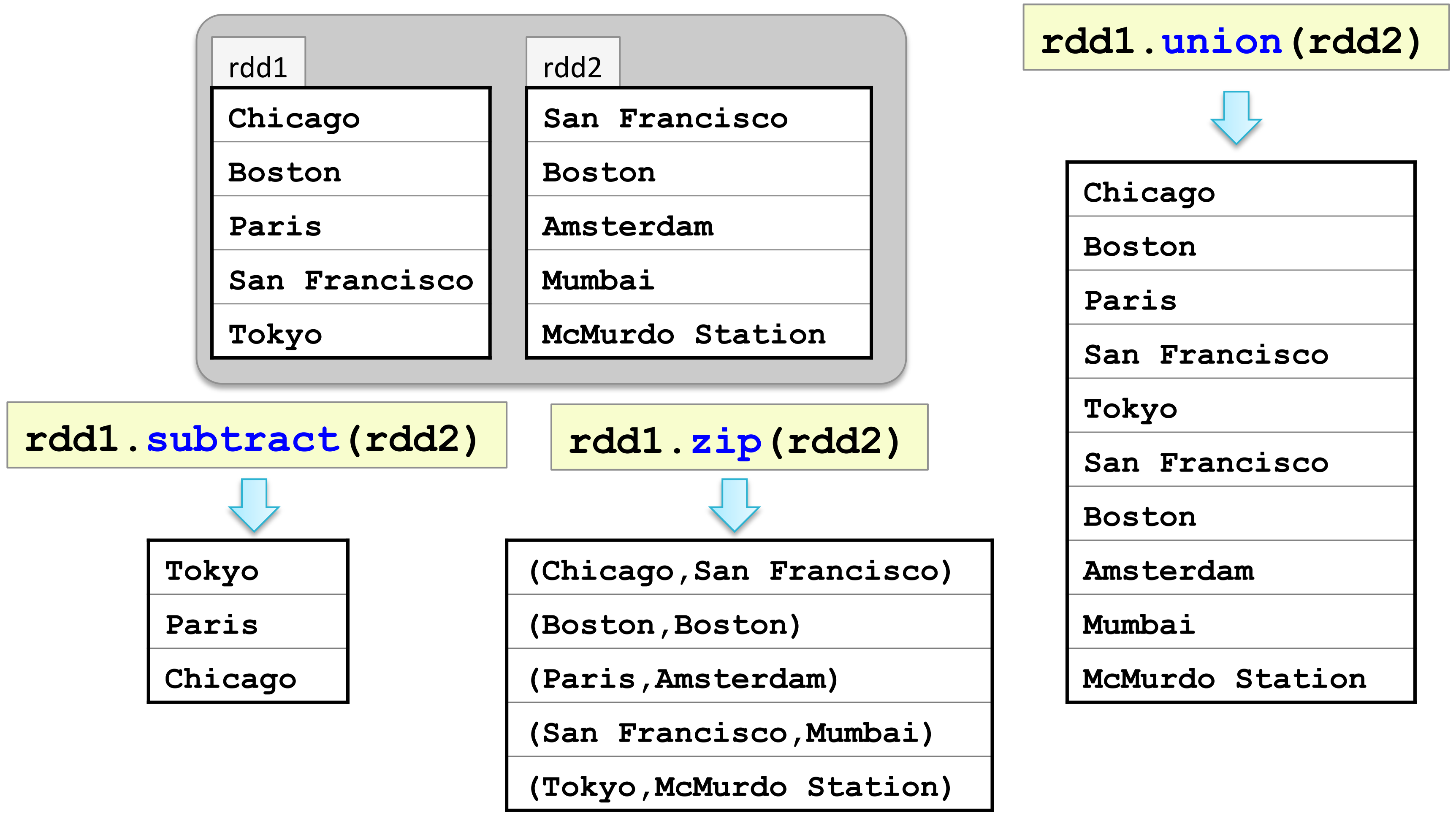
```
> sc.textFile(file) \
  .flatMap(lambda line: line.split()) \
  .distinct()
```

Scala

```
> sc.textFile(file).
   flatMap(line => line.split(' ')).
   distinct()
```



Examples: Multi-RDD Transformations



Some Other General RDD Operations

- **Other RDD operations**
  - **`first`** – return the first element of the RDD
  - **`foreach`** – apply a function to each element in an RDD
  - **`top(`*`n`*`)`** – return the largest *n* elements using natural ordering
- **Sampling operations**
  - **`sample`** – create a new RDD with a sampling of elements
  - **`takeSample`** – return an array of sampled elements
- **Double RDD operations**
  - Statistical functions, e.g., **`mean`**, **`sum`**, **`variance`**, **`stdev`**

Chapter Topics

Working With RDDs in Spark | **Distributed Data Processing with Spark**

- Creating RDDs
- Other General RDD Operations
- **Conclusion**
- Homework: Process Data Files with Spark

Essential Points

- RDDs can be created from files, parallelized data in memory, or other RDDs
- `sc.textFile` reads newline delimited text, one line per RDD record
- `sc.wholeTextFile` reads entire files into single RDD records
- Generic RDDs can consist of any type of data
- Generic RDDs provide a wide range of transformation operations

Chapter Topics

Working With RDDs in Spark | **Distributed Data Processing with Spark**

- Creating RDDs
- Other General RDD Operations
- Conclusion
- **Homework: Process Data Files with Spark**

Homework: Process Data Files with Spark

- **In this homework assignment you will**
  - Process a set of XML files using **`wholeTextFiles`**
  - Reformat a dataset to standardize format (bonus)
- **Please refer to the Homework description**