Notes on Learning Spark

Weikai Mao

Contents

CHAPTER 3. Programming with RDDs

RDD Basics

To summarize, every Spark program and shell session will work as follows:

1. Create some input RDDs from external data.
2. Transform them to define new RDDs using transformations like filter() .
3. Ask Spark to persist() any intermediate RDDs that will need to be reused.
4. Launch actions such as count() and first() to kick off a parallel computation, which is then optimized and executed by Spark.

Creating RDDs

Spark provides two ways to create RDDs: loading an external dataset and parallelizing a collection in your driver program.

Example 3-8. textFile() method in Python

```
lines = sc.textFile("/path/to/README.md")
```

Example 3-9. textFile() method in Scala

```
val lines = sc.textFile("/path/to/README.md")
```

RDD Operations

RDDs support two types of operations: transformations and actions. Transformations are operations on RDDs that return a new RDD, such as map() and filter() . Actions are operations that return a result to the driver program or write it to storage, and kick off a computation, such as count() and first() .

Transformations

Example 3-11. `filter()` and `union()` transformation in Python

```
inputRDD = sc.textFile("log.txt")
errorsRDD = inputRDD.filter(lambda x: "error" in x)
warningsRDD = inputRDD.filter(lambda x: "warning" in x)
badLinesRDD = errorsRDD.union(warningsRDD)
```

Example 3-12. `filter()` transformation in Scala

```
val inputRDD = sc.textFile("log.txt")
val errorsRDD = inputRDD.filter(line => line.contains("error"))
```

Actions

Example 3-15. Python error count using actions

```
print "Input had " + badLinesRDD.count() + " concerning lines"
print "Here are 10 examples:"
for line in badLinesRDD.take(10):
    print line
```

Example 3-16. Scala error count using actions

```
println("Input had " + badLinesRDD.count() + " concerning lines")
println("Here are 10 examples:")
badLinesRDD.take(10).foreach(println)
```

Lazy Evaluation

Transformations on RDDs are lazily evaluated, meaning that Spark will not begin to execute until it sees an action.

Lazy evaluation means that when we call a transformation on an RDD (for instance, calling map()), the operation is not immediately performed. Instead, Spark internally records metadata to indicate that this operation has been requested. Rather than thinking of an RDD as containing specific data, it is best to think of each RDD as consisting of instructions on how to compute the data that we build up through transformations. Loading data into an RDD is lazily evaluated in the same way transformations are. So, when we call `sc.textFile()`, the data is not loaded until it is necessary. As with transformations, the operation (in this case, reading the data) can occur multiple times.

Passing Functions to Spark

Most of Spark's transformations, and some of its actions, depend on passing in functions that are used by Spark to compute data.

Python

In Python, we have three options for passing functions into Spark.

1. pass in lambda expressions (Example 3-2, Example 3-18)
2. pass in top-level functions
3. pass in locally defined functions.

Example 3-18. Passing functions in Python

```
word = rdd.filter(lambda s: "error" in s)
def containsError(s):
return "error" in s
word = rdd.filter(containsError)
```

Example 3-20. Python function passing without field references

```
class WordFunctions(object):
...
def getMatchesNoReference(self, rdd):
# Safe: extract only the field we need into a local variable
query = self.query
return rdd.filter(lambda x: query in x)
```

Scala

In Scala, we can pass in functions defined inline, references to methods, or static functions as we do for Scala's other functional APIs.

Example 3-21. Scala function passing

```
```

```
class SearchFunctions(val query: String) {
    def isMatch(s: String): Boolean = {
        s.contains(query)
    }
    def getMatchesFunctionReference(rdd: RDD[String]): RDD[String] = {
        // Problem: "isMatch" means "this.isMatch", so we pass all of "this"
        rdd.map(isMatch)
    }
    def getMatchesFieldReference(rdd: RDD[String]): RDD[String] = {
        // Problem: "query" means "this.query", so we pass all of "this"
        rdd.map(x => x.split(query))
    }
    def getMatchesNoReference(rdd: RDD[String]): RDD[String] = {
        // Safe: extract just the field we need into a local variable
        val query_ = this.query
        rdd.map(x => x.split(query_))
    }
}
```

Common Transformations and Actions

Basic RDDs

Element-wise transformations

The two most common transformations you will likely be using are `map()` and `filter()`.

The map() transformation takes in a function and applies it to each element in the RDD with the result of the function being the new value of each element in the resulting RDD. The filter() transformation takes in a function and returns an RDD that only has elements that pass the filter() function.

It is useful to note that map() 's return type does not have to be the same as its input type.

Let's look at a basic example of map() that squares all of the numbers in an RDD (Examples 3-26 through 3-28).

Example 3-26. Python squaring the values in an RDD

```
nums = sc.parallelize([1, 2, 3, 4])
squared = nums.map(lambda x: x * x).collect()
for num in squared:
print "%i " % (num)
```

Example 3-27. Scala squaring the values in an RDD

```
val input = sc.parallelize(List(1, 2, 3, 4))
val result = input.map(x => x * x)
println(result.collect().mkString(","))
```

Sometimes we want to produce multiple output elements for each input element. The operation to do this is called `flatMap()`.

Example 3-29. flatMap() in Python, splitting lines into words

```
lines = sc.parallelize(["hello world", "hi"])
words = lines.flatMap(lambda line: line.split(" "))
words.first() # returns "hello"
```

Example 3-30. flatMap() in Scala, splitting lines into multiple words

```
val lines = sc.parallelize(List("hello world", "hi"))
val words = lines.flatMap(line => line.split(" "))
words.first() // returns "hello"
```

The difference between `flatMap()` and `map()`:

Pseudo set operations

Four operations (`distinct()`, `union()`, `intersection()`, `subtract()`) are shown in Figure 3-4. It's important to note that all of these operations require that the RDDs being operated on are of the same type.

Note that distinct() is expensive, however, as it requires shuffling all the data over the network to ensure that we receive only one copy of each element. Shuffling, and how to avoid it, is discussed in more detail in Chapter 4.

We can also compute a Cartesian product between two RDDs, as shown in Figure 3-5.

Actions

The most common action on basic RDDs you will likely use is `reduce()`.

Example 3-32. `reduce()` in Python

```
sum = rdd.reduce(lambda x, y: x + y)
```

Example 3-33. `reduce()` in Scala

```
val sum = rdd.reduce((x, y) => x + y)
```

Reference

Karau, Holden, et al. *Learning spark: lightning-fast big data analysis*. " O'Reilly Media, Inc.", 2015.