Programming with Apache Spark

Lesson 3

Hortonworks

Learning Objectives

- After you complete this lesson you should be able to:
  - Start the spark shell
  - Understand what an RDD is
  - Load data from the HDFS and perform a word count
  - Know the differences between Transformation and Action
  - Explain Lazy Evaluation

Hortonworks

How to start using Apache Spark?

- The Spark Shell provides an interactive way to learn Spark and explore data
- Available for python and scala
  - pyspark
  - spark-shell
- REPL

The Spark Context

- Main entry point for Spark Applications
- All Spark Applications require one
- The Spark Context tells Spark how to access a cluster
- The REPLs automatically create one for you
  - In Spark 1.3 and on, the shell creates a SQL context too

Working with the Spark Context

Attributes:

- sc.appName: Spark application name
- sc.master: Spark Master (local, yarn-client, etc)
- sc.version: Version of Spark being used

Functions:

- sc.parallelize(): create an RDD from local data
- sc.textfile(): create RDD from a text file in HDFS
- sc.stop(): stop the spark context

Hortonworks

The Resilient Distributed Dataset

- An *Immutable* collection of objects (or records) that can be operated on in parallel
  - **Resilient:** can be recreated from parent RDDs - An RDD keeps its lineage information
  - **Distributed:** partitions are distributed across nodes in the cluster
  - **Dataset**: a set of data that can be accessed
  - Each RDD is composed of 1 or more partitions - The user can control the number of partitions - More partitions => more parallelism

Hortonworks

Create an RDD

- Load data from HDFS – or any other file system (S3, Local, etc)

```
rdd1 = sc.textFile("/path/to/file.txt")
rdd2 = sc.textFile("hdfs://namenode:8020/mydata/")
```

- With parallelize() function in driver – useful for learning Spark

```
rdd3 = sc.parallelize([1, 2, 3, 4, 5])
```

Hortonworks

Working with RDDs

- RDDs have two types of operations
  - Transformations: the RDD is transformed into a new RDD
  - Actions: an action is performed on the RDD and a result is returned to the driver, or data is saved somewhere
- Transformations are lazy: they do not compute until an action is performed

Hortonworks

Actions – count()

- The count() action returns the number of elements in the RDD

```
data= [5, 12, -4 , 7, 20]
rdd= sc.parallelize(data)
rdd.count()
```

The output is: 5

Actions – reduce()

- The reduce() action has a lot of use cases in Spark
  - Aggregating elements of an RDD using a defined function
  - That function must be commutative and associative
    - a+b = b+a and a+(b+c)=(a+b)+c

```
rdd.reduce(lambda a, b : a+b)
40

rdd.reduce(lambda a, b: a if (a>b) else b)
20
```

Other Useful Spark Actions

- first(): return the first element in the RDD
- take(n): return the first n elements of the RDD
- collect(): return all the elements in the RDD to the driver
    - Make sure you only call this on small datasets or risk crashing your driver!
- saveAsTextFile(path): write the RDD to a file

Spark Actions: Examples

rdd.first(): 5

rdd.take(3): [5, 12, -4]

rdd.saveAsTextFile("myfile")

Spark Transformations

- Spark Transformations create new RDD’s from existing ones
- The transformation is lazy, and doesn’t occur until an action is called on the rdd, or subsequent rdd
  - Transformation create a recipe, or lineage, for processing
  - The actions trigger data to flow through the transformation and create the result

Hortonworks

Transformations: map()

- Map applies a function to each element of the RDD

```
rdd=sc.parallelize([1, 2, 3, 4, 5])
rdd.map(lambda x: x*2+1).collect()
[3, 5, 7, 9, 11]
```

Hortonworks

Transformations: flatMap()

- Map applies a function to each element of the RDD

```
rdd.map(lambda x: [x, x*2]).collect()
[(1,2), (2, 4), (3,6), (4,8), (5,10)]

rdd.flatMap(lambda x: [x, x*2]).collect()
[1, 2, 2, 4, 3, 6, 4, 8, 5, 10]
```

Hortonworks

Transformation: filter()

- Keep some elements based on a predicate

```
rdd.filter( lambda x:  x%2 == 0).collect()
[2, 4]

rdd.filter( lambda x: x<3).collect()
[1, 2]
```

Hortonworks

Transformation: distinct()

- Remove all duplicate elements

```
rdd.flatMap( lambda x: [x, x*2]).distinct().collect()
[8, 4, 1, 5, 2, 10, 6, 3]
```

Key Value Pair Intro (Pair RDDs)

- A Key/Value RDD is an RDD whose elements comprise a pair of values – key and value
- Pair-RDDs are very useful for many applications
  - Allow to group operations by key
  - For example: join(), groupByKey(), or reduceByKey

Creating Pair RDDs

- Pair RDDs are often created from regular RDDs by using the map() transformation:

```
wordlist = ‘this is my list and it is a nice list’
rdd1 = sc.parallelize([wordlist])
kv_rdd = rdd1.flatMap(lambda x: x.split(‘ ‘)).
        .map(lambda x: (x,1))
kv_rdd.collect()
[(this, 1), (is, 1), (my, 1), (list, 1), (and, 1), … (list,1)]
```

Hortonworks

Pair RDD Transformation: reduceByKey()

- reduceByKey performs a reduce function on all elements of a key/value pair RDD that share a key

```
kv_rdd.reduceByKey(lambda a,b: a+b).collect()
[('this', 1), ('my', 1), ('and', 1), ('list', 2), ('a', 1), ('it', 1),
 ('is', 2), ('nice', 1)]
```

Hortonworks

Conclusion and Key Points

- There are two* types of operations
  - Transformation which returns a new RDD
  - Action which returns a result
- Spark is lazy, it only does work when it has too
- RDD’s are in your mind
  - They’re just a set of directions to transform data, the data is never stored in the RDD

Hortonworks