Apache Spark Introduction

Introduction

You can find the material for this course available at

databricks.com/spark-training-resources#itas

Download the presentation, code and data here:

training.databricks.com/workshop/itas_workshop.pdf
training.databricks.com/workshop/usb.zip

Overview

Overview

At the end of this course you might be able to

- Run shell Spark
- Explore data through HDFS
- Use spark to deal with typical cases
- Understand the functional programming fundamentals
- Be familiar with Spark API
- Use Spark, Spark SQL and Spark Streaming

Installing Spark

Installation:

Let’s start using spark in 3 steps

databricks.com/spark-training-resources#itas

https://spark.apache.org/downloads.html

Attention: Do not install or run Spark using:

- Homebrew or ports on MacOS
- Cygwin on Windows

Step 1:

Verify if you already have Java 7 installed

oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html

- Follow the instructions and accept the license
- Select the version according to your OS

Step 2: *Download Spark*

We will use Spark 1.5.0 (+)

1. Download Spark (http://spark.apache.org/downloads.html)
2. $ tar xvzf spark-{version}.tar.gz
3. $ cd spark-{version}/*

*** In this directory you can find code examples**

Step 3: *Run Spark (spark-shell)*

Run pyspark on directory “spark”

```
./bin/pyspark
```

And use the prompt python “>>>"

Step 3: *Run Spark (spark-shell)*

You can also use Scala
Run Spark-shell on “spark” directory:

```
./bin/spark-shell
```

We will use the prompt “scala>"

Functional programming

Functional Programming

- It is a declarative programming paradigm in which the algorithm is described in terms of functions.

- Functions are evaluated as mathematical functions, it means that functions return the same value for the same input.

- Lambda function:

It is an anonymous function (no identifier)

```
lambda x: x + 1
```

Typically used with **map(), filter(), reduce()**

Functional Programming

- Defining a regular function (types are statically defined)

```
>>> def f (x): return x**2
>>> print f(8)
64
```

- Defining a lambda function

```
>>> g = lambda x: x**2
>>>
>>> print g(8)
64
```

Closure function:

The return of a closure function depends on a variable defined in an enclosing scope

```
>>> multiplier = lambda i: i * 10


>>> factor = 3
>>> multiplier = lambda i: i * factor
```

Resilient Distributed Dataset (RDD)

Resilient Distributed Dataset (RDD)

- RDD is a distributed immutable collection of objects.
- RDDs are divided in multiples partitions, that can be operated on in parallel.
- RDDs can hold any object type: Python, Java, Scala, including classes defined by the user.

Resilient Distributed Dataset (RDD)

- RDDs can be created in two ways:
  - Loading external object

```
>>> lines = sc.textFile("README.md")
```

  - Collection of objects (e.g, list)

```
>>> data = range(1,10000)
>>> distData = sc.parallelize(data)
```

Resilient Distributed Dataset (RDD)

- Spark provides special operations for RDDs containing pairs <key, value>.
- These RDDs are called *pair RDD.*

Resilient Distributed Dataset (RDD)

- **Pair RDD** supports parallel operations over the keys and/or rejoin data based on keys.
- For example:
  - *reduceByKey*() merges the values for each key using an associative reduce function,
  - *join*() returns an RDD containing all pairs of elements with matching keys.

Resilient Distributed Dataset (RDD)

- Creating a pair RDD (Scala)

```
>>> pairs = lines.map(lambda x: (x.split(" ")[0], x))
```

ELT (Extract-Load-Transform)

ELT (Extract, Load, and Transform)

- ELT is the process that performs:
  - Data extraction
  - Loading
  - Transformation when the data is needed .

ELT (Extract, Load, and Transform)

- Extract
  - Data is extracted and load to Spark by using the SparkContext.
  - The method *textFile()* reads a file as a collection of lines.

```
textFileHDFS = sc.textFile("hdfs://...")
textFileLocal = sc.textFile("/home/...")
textFileAmazon = sc.textFile("s3n//...")
```

ELT (Extract, Load, and Transform)

- Load (https://spark.apache.org/docs/1.1.0/sql-programming-guide.html#loading-data-programmatically)

```
// sqlContext from the previous example is used in this example.
// createSchemaRDD is used to implicitly convert an RDD to a SchemaRDD.
import sqlContext.createSchemaRDD

people: RDD[Person] = ... // An RDD of case class objects, from the previous example.

// The RDD is implicitly converted to a SchemaRDD by createSchemaRDD, allowing it to be stored using Parquet.
people.write.mode('overwrite').parquet("people.parquet")

// Read in the parquet file created above.  Parquet files are self-describing so the schema is preserved.
// The result of loading a Parquet file is also a SchemaRDD.
parquetFile = sqlContext.read.parquet("people.parquet")

//Parquet files can also be registered as tables and then used in SQL statements.
parquetFile.registerTempTable("parquetFile")
teenagers = sqlContext.sql("SELECT name FROM parquetFile WHERE age >= 13 AND age <= 19")
teenagers.map(lambda t: "Name: " + t(0)).collect()
```

ELT (Extract, Load, and Transform)

- Once the RDD is created it is possible to apply two operation types
  - **Transformations** to create a new RDD.
  - **Actions** to launch a job.

  - http://spark.apache.org/docs/latest/programming-guide.html#transformations

ELT (Extract, Load, and Transform)

- **Transformations** (20):

(map, filter, flatMap, mapPartitions, mapPartitionsWithIndex, sample, union, intersection, distinct, groupByKey, reduceByKey, aggregateByKey, sortByKey, join, cogroup, cartesian, pipe, coalesce, repartition, repartitionAndSortWithinPartitions).

- **Actions** (11):

(reduce, collect, count, first, take, takeSample, saveAsTextFile, saveAsSequenceFile, saveAsObjectFile, countByKey, foreach).

- http://spark.apache.org/docs/latest/programming-guide.html#transformations

ELT (Extract, Load, and Transform)

- Python code in red.
- Transformations in blue.
- Actions in purple.

ELT (Extract, Load, and Transform)

- filter(): Returns a new RDD with a sub set of items.

```
textFile = sc.textFile("hdfs://...")
errors = textFile.filter(lambda line: line.contains("ERROR"))
```

ELT (Extract, Load, and Transform)

- count(): Returns the number of elements of the dataset

```
textFile = sc.textFile("hdfs://...")
errors = textFile.filter(lambda line: line.contains("ERROR"))
// Count all the errors
errors.count()
// Count errors mentioning MySQL
errors.filter(lambda line: line.contains("MySQL")).count()
```

ELT (Extract, Load, and Transform)

- collect(): Returns the RDD elements as a list

```
textFile = sc.textFile("hdfs://...")
errors = textFile.filter(lambda line: line.contains("ERROR"))
// Count all the errors
errors.count()
// Count errors mentioning MySQL
errors.filter(lambda line: line.contains("MySQL")).count()
// Fetch the MySQL errors as an array of strings
errors.filter(lambda line: line.contains("MySQL")).collect()
```

ELT (Extract, Load, and Transform)

- map(): Maps an item as a list <key, value>.

```
textFile = sc.textFile("hdfs://...")
errors = textFile.filter(lambda line: line.contains("ERROR"))
// Count all the errors
errors.count()
// Count errors mentioning MySQL
errors.filter(lambda line: line.contains("MySQL")).count()
// Fetch the MySQL errors as an array of strings
errors.filter(lambda line: line.contains("MySQL")).collect()
errors.map(_.split("\t")).map(lambda r: r(1))
```

ELT (Extract, Load, and Transform)

- RDDs can persist in different levels (disk, memory, or replicated through cluster nodes

```
errors.persist(StorageLevel.MEMORY_ONLY)
errors.persist(StorageLevel.MEMORY_AND_DISK)
errors.persist(StorageLevel.MEMORY_ONLY_SER)
errors.persist(StorageLevel.MEMORY_AND_DISK_SER)
errors.persist(StorageLevel.DISK_ONLY)
errors.persist(StorageLevel.OFF_HEAP) @Experimental
errors.persist(StorageLevel.MEMORY_ONLY_2)
```

- http://spark.apache.org/docs/latest/programming-guide.html#rdd-persistence

ELT (Extract, Load, and Transform)

- The method cache() is a shortcut to use the default storage level.

  StorageLevel.MEMORY_ONLY (stores objects in memory).

```
errors.cache() // stores the variable errors in memory
```

DataFrame

DataFrame

- DataFrames
  - DataFrame is a distributed collection of data organized in columns.
  - It is equivalent to a table in a Relational Database.
  - DataFrames are constructed from different sources of data, such as files, database tables or RDDs.

DataFrame

- DataFrames provide a domain-specific language for structured data manipulation in Scala, Java, Python and R.

```
// in Python
people = sqlContext.read.parquet("...")
// in Java
DataFrame people = sqlContext.read().parquet("...")
```

DataFrame

- Picking a column

```
// in Scala
people = sqlContext.read.parquet("...")
ageCol = people.select("age")
```