L6: Introduction to Spark (Spark 基础)

Feng Li
feng.li@cufe.edu.cn

School of Statistics and Mathematics
Central University of Finance and Economics

Revised on December 20, 2017

Today we are going to learn... (本节知识要点)

Introduction (简介)

- Apache Spark is a fast and general-purpose cluster computing system. It provides high-level APIs in Java, Scala and Python, and an optimized engine that supports general execution graphs.
- It also supports a rich set of higher-level tools including
    - **Spark SQL** for SQL and structured data processing
    - **MLlib** for machine learning
    - **GraphX** for graph processing
    - **Spark Streaming**.

Cluster Mode （服务器模式） I

- Spark applications run as independent sets of processes on a cluster, coordinated by the **SparkContext** object in your main program (called the **driver program**).

1. Specifically, to run on a cluster, the SparkContext can connect to several types of cluster managers
   - Spark's own `standalone` cluster manager
   - `Hadoop YARN`)
   - `Apache Mesos`

   which allocate resources across applications.
2. Once connected, Spark acquires executors on nodes in the cluster, which are processes that run computations and store data for your application.
3. Next, it sends your application code (defined by **JAR** or **Python files** passed to SparkContext) to the executors
4. Finally, SparkContext sends tasks for the executors to run.

Cluster Mode （服务器模式） II

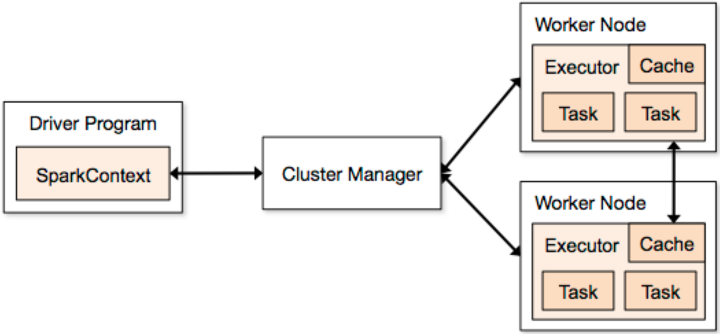

Figure: Spark Cluster Components

Install and Configure Spark (安装和配置 Spark)

- Install Spark
  - Download the source code.
  - Compile it with JDK
    `http://spark.apache.org/docs/latest/building-with-maven.html`
  - Link it with Hadoop
    `http://spark.apache.org/docs/latest/running-on-yarn.html`
  - Link it with Mahout
    `http://mahout.apache.org/users/sparkbindings/home.html`
- Configure Spark
  `http://spark.apache.org/docs/latest/configuration.html`

Interactive Analysis with the Spark Shell （利用 Spark Shell 交互分析） I

- **Spark's shell** provides a simple way to learn the API, as well as a powerful tool to analyze data interactively. It is available in either **Scala** (which runs on the Java VM and is thus a good way to use existing Java libraries) or `Python`.
- Start the Python version with exactly 4 cores by running the following in the Spark directory:

  ```
  ./bin/pyspark --master local[4]
  ```

  To find a complete list of options, run `pyspark --help`.
- Start the Scala version by running the following in the Spark directory:

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

- All examples based on this section will be based on Python. One may also check out the **Scala/Python/R** version at `http://spark.apache.org/docs/latest/programming-guide.html`

Interactive Analysis with the Spark Shell （利用 Spark Shell 交互分析） II

- Spark's primary abstraction is a distributed collection of items called a `Resilient Distributed Dataset` (RDD). RDDs can be created from Hadoop InputFormats (such as HDFS files) or by transforming other RDDs.
- To make a new RDD from the text of the README file in the Spark source directory:

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

- RDDs have actions, which return values, and transformations, which return pointers to new RDDs.

```
>>> textFile.count() # Number of items in this RDD
126

>>> textFile.first() # First item in this RDD
# Apache Spark'
```

- RDD actions and transformations can be used for more complex computations. Let's say we want to find the line with the most words:

Interactive Analysis with the Spark Shell （利用 Spark Shell 交互分析） III

```
>>> textFile.map(lambda line: len(line.split())). \
    reduce(lambda a, b: a if (a > b) else b)
15
```

- Spark also supports pulling data sets into a cluster-wide in-memory cache. This is very useful when data is accessed repeatedly

```
>>> linesWithSpark.cache()

>>> linesWithSpark.count()
15

>>> linesWithSpark.count()
15
```

Standalone Applications (独立应用) I

- Assume we like to write a program that just counts the number of lines containing 'a' and the number containing 'b' in the Spark README.
- The Python version

```
"""SimpleApp.py"""
from pyspark import SparkContext

logFile = "YOUR_SPARK_HOME/README.md"  # some file on system
sc = SparkContext("local", "Simple App")
logData = sc.textFile(logFile).cache()

numAs = logData.filter(lambda s: 'a' in s).count()
numBs = logData.filter(lambda s: 'b' in s).count()

print "Lines with a: %i, lines with b: %i" % (numAs, numBs)
```

- The Java version

Standalone Applications (独立应用) II

```
/* SimpleApp.java */
import org.apache.spark.api.java.*;
import org.apache.spark.SparkConf;
import org.apache.spark.api.java.function.Function;

public class SimpleApp {
  public static void main(String[] args) {
    String logFile = "YOUR_SPARK_HOME/README.md"; // Should be s
    SparkConf conf = new SparkConf().setAppName("Simple Applicat
    JavaSparkContext sc = new JavaSparkContext(conf);
    JavaRDD<String> logData = sc.textFile(logFile).cache();

    long numAs = logData.filter(new Function<String, Boolean>()
      public Boolean call(String s) { return s.contains("a"); }
    }).count();

    long numBs = logData.filter(new Function<String, Boolean>()
```

Standalone Applications (独立应用) III

```
      public Boolean call(String s) { return s.contains("b"); }
    }).count();

    System.out.println("Lines with a: " + numAs + ", lines with
  }
}
```

- The Scala version

```
/* SimpleApp.scala */
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark.SparkConf

object SimpleApp {
  def main(args: Array[String]) {
    val logFile = "YOUR_SPARK_HOME/README.md" // Should be some
    val conf = new SparkConf().setAppName("Simple Application")
    val sc = new SparkContext(conf)
```

Standalone Applications (独立应用) IV

```
    val logData = sc.textFile(logFile, 2).cache()
    val numAs = logData.filter(line => line.contains("a")).count
    val numBs = logData.filter(line => line.contains("b")).count
    println("Lines with a: %s, Lines with b: %s".format(numAs, n
  }
}
```

Submitting Applications to Spark （向 Spark 提交应用任务） I

- Bundling Your Application's Dependencies
  - If your code depends on other projects, you will need to package them alongside your application in order to distribute the code to a Spark cluster.
  - To do this, to create an assembly `jar` containing your code and its dependencies. When creating assembly jars, list Spark and Hadoop as provided dependencies; these need not be bundled since they are provided by the cluster manager at runtime.
  - For Python, you can use the `--py-files` argument of spark-submit to add .py, .zip or .egg files to be distributed with your application. If you depend on multiple Python files, pack them into a .zip or .egg.
- Once a user application is bundled, it can be launched using the `bin/spark-submit` script.
- Here are a few examples
  - Run application locally on 8 cores

Submitting Applications to Spark （向 Spark 提交应用任务） II

```
./bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
  --master local[8] \
  /path/to/examples.jar \
  100
```

- Run on a Spark standalone cluster

```
./bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
  --master spark://207.184.161.138:7077 \
  --executor-memory 20G \
  --total-executor-cores 100 \
  /path/to/examples.jar \
  1000
```

- Run on a Hadoop YARN cluster

Submitting Applications to Spark (向 Spark 提交应用任务) III

```
export HADOOP_CONF_DIR=XXX
./bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
  --master yarn-cluster \  # can also be `yarn-client` for client m
  --executor-memory 20G \
  --num-executors 50 \
  /path/to/examples.jar \
  1000
```

- Run a Python application on a cluster

```
./bin/spark-submit \
  --master spark://207.184.161.138:7077 \
  examples/src/main/python/pi.py \
  1000
```

More PySpark exmaples

- https://github.com/apache/spark/tree/master/examples/src/main/python

Spark Machine Learning Library I
↛ Introduction

- MLlib is Spark's machine learning (ML) library. Its goal is to make practical machine learning scalable and easy. At a high level, it provides tools such as:
  - ML Algorithms: common learning algorithms such as classification, regression, clustering, and collaborative filtering
  - Featurization: feature extraction, transformation, dimensionality reduction, and selection
  - Pipelines: tools for constructing, evaluating, and tuning ML Pipelines [1]
  - Persistence: saving and load algorithms, models, and Pipelines
  - Utilities: linear algebra, statistics, data handling, etc.
- To use `MLlib` in Python, you will need `NumPy` version 1.4 or newer.

Spark Machine Learning Library II
↛ Introduction

- As of Spark 2.0, The primary Machine Learning API for Spark is now the **DataFrame-based** API in the `spark.ml` package.
  - DataFrames provide a more user-friendly API than RDDs. The many benefits of DataFrames include Spark Datasources, SQL/DataFrame queries, Tungsten and Catalyst optimizations, and uniform APIs across languages.
  - The DataFrame-based API for MLlib provides a uniform API across ML algorithms and across multiple languages.
  - DataFrames facilitate practical ML Pipelines, particularly feature transformations

[1] `http://spark.apache.org/docs/latest/ml-pipeline.html`

Spark Machine Learning Library I
↛ Data Types

- MLlib supports **local vectors** and **matrices** stored on a single machine, as well as distributed matrices backed by one or more RDDs. Local vectors and local matrices are simple data models that serve as public interfaces. The underlying linear algebra operations are provided by Breeze[2].
- **Local vector** MLlib recognizes the following types as dense vectors:
  - NumPy's array
  - Python's list, e.g., [1, 2, 3]
  - MLlib's SparseVector.
  - SciPy's `csc_matrix` with a single column

```
import numpy as np
import scipy.sparse as sps
from pyspark.mllib.linalg import Vectors

# Use a NumPy array as a dense vector.
dv1 = np.array([1.0, 0.0, 3.0])
# Use a Python list as a dense vector.
dv2 = [1.0, 0.0, 3.0]
# Create a SparseVector.
```

Spark Machine Learning Library II

↬ Data Types

```
sv1 = Vectors.sparse(3, [0, 2], [1.0, 3.0])
# Use a single-column SciPy csc_matrix as a sparse vector.
sv2 = sps.csc_matrix((np.array([1.0, 3.0]),
                      np.array([0, 2]),
                      np.array([0, 2])), shape=(3, 1))
```

We recommend using NumPy arrays over lists for efficiency, and using the factory methods implemented in Vectors to create sparse vectors.

Spark Machine Learning Library III
↛ Data Types

- **Labeled point** A labeled point is a local vector, either dense or sparse, associated with a label/response.
  - In MLlib, labeled points are used in supervised learning algorithms.
  - We use a double to store a label, so we can use labeled points in both regression and classification.
  - For binary classification, a label should be either 0 (negative) or 1 (positive). For multiclass classification, labels should be class indices starting from zero: 0, 1, 2, ...

```
from pyspark.mllib.linalg import SparseVector
from pyspark.mllib.regression import LabeledPoint

# Create a labeled point with a positive label and
# a dense feature vector.
pos = LabeledPoint(1.0, [1.0, 0.0, 3.0])

# Create a labeled point with a negative label and
# a sparse feature vector.
neg = LabeledPoint(0.0, SparseVector(3, [0, 2], [1.0, 3.0]))
```

Spark Machine Learning Library IV
↛ Data Types

- **Sparse data** It is very common in practice to have sparse training data. MLlib supports reading training examples stored in `LIBSVM` format
  - `MLUtils.loadLibSVMFile` reads training examples stored in `LIBSVM` format.
  - Refer to the MLUtils Python docs for more details on the API.

```
from pyspark.mllib.util import MLUtils
examples = MLUtils.loadLibSVMFile(sc,
           "data/mllib/sample_libsvm_data.txt")
```

- **Local matrix** A local matrix has integer-typed row and column indices and double-typed values, stored on a single machine.
  - MLlib supports dense matrices, whose entry values are stored in a single double array in column-major order, and sparse matrices, whose non-zero entry values are stored in the Compressed Sparse Column (CSC) format in column-major order.
  - The base class of local matrices is `Matrix`, and Spark ML provides two implementations: `DenseMatrix`, and `SparseMatrix`. We recommend using the factory methods implemented in `Matrices` to create local matrices. Remember, local matrices in MLlib are stored in column-major order.

Spark Machine Learning Library V
↬ Data Types

```
from pyspark.mllib.linalg import Matrix, Matrices
# Create a dense matrix ((1.0, 2.0), (3.0, 4.0), (5.0, 6.0))
dm2 = Matrices.dense(3, 2, [1, 2, 3, 4, 5, 6])
# Create a sparse matrix ((9.0, 0.0), (0.0, 8.0), (0.0, 6.0))
sm = Matrices.sparse(3, 2, [0, 1, 3], [0, 2, 1], [9, 6, 8])
```

- **Distributed matrix** A distributed matrix has long-typed row and column indices and double-typed values, stored distributively in one or more RDDs. It is very important to choose the right format to store large and distributed matrices. Converting a distributed matrix to a different format may require a global shuffle, which is quite expensive. Four types of distributed matrices have been implemented so far.
  - The basic type is called **RowMatrix**. A RowMatrix is a row-oriented distributed matrix without meaningful row indices, e.g., a collection of feature vectors. It is backed by an RDD of its rows, where each row is a local vector. We assume that the number of columns is not huge for a RowMatrix so that a single local vector can be reasonably communicated to the driver and can also be stored / operated on using a single node.

Spark Machine Learning Library VI
↬ Data Types

```
from pyspark.mllib.linalg.distributed import RowMatrix
# Create an RDD of vectors.
rows = sc.parallelize([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12
# Create a RowMatrix from an RDD of vectors.
mat = RowMatrix(rows)
# Get its size.
m = mat.numRows()  # 4
n = mat.numCols()  # 3
# Get the rows as an RDD of vectors again.
rowsRDD = mat.rows
```

- An **IndexedRowMatrix** is similar to a RowMatrix but with row indices, which can be used for identifying rows and executing joins.

Spark Machine Learning Library VII
↬ Data Types

```
from pyspark.mllib.linalg.distributed import IndexedRow, IndexedRow
# Create an RDD of indexed rows.
#   - This can be done explicitly with the IndexedRow class:
indexedRows = sc.parallelize([IndexedRow(0, [1, 2, 3]),
                              IndexedRow(1, [4, 5, 6]),
                              IndexedRow(2, [7, 8, 9]),
                              IndexedRow(3, [10, 11, 12])])
#   - or by using (long, vector) tuples:
indexedRows = sc.parallelize([(0, [1, 2, 3]), (1, [4, 5, 6]),
                              (2, [7, 8, 9]), (3, [10, 11, 12])])
# Create an IndexedRowMatrix from an RDD of IndexedRows.
mat = IndexedRowMatrix(indexedRows)
# Get its size.
m = mat.numRows()  # 4
n = mat.numCols()  # 3
# Get the rows as an RDD of IndexedRows.
rowsRDD = mat.rows
# Convert to a RowMatrix by dropping the row indices.
rowMat = mat.toRowMatrix()
```

Spark Machine Learning Library VIII
↛ Data Types

- A **CoordinateMatrix** is a distributed matrix stored in coordinate list (COO) format, backed by an RDD of its entries.

```
from pyspark.mllib.linalg.distributed import CoordinateMatrix, Matr
# Create an RDD of coordinate entries.
#   - This can be done explicitly with the MatrixEntry class:
entries = sc.parallelize([MatrixEntry(0, 0, 1.2), MatrixEntry(1, 0,
#   - or using (long, long, float) tuples:
entries = sc.parallelize([(0, 0, 1.2), (1, 0, 2.1), (2, 1, 3.7)])
# Create an CoordinateMatrix from an RDD of MatrixEntries.
mat = CoordinateMatrix(entries)
# Get its size.
m = mat.numRows()  # 3
n = mat.numCols()  # 2
# Get the entries as an RDD of MatrixEntries.
entriesRDD = mat.entries
# Convert to a RowMatrix.
rowMat = mat.toRowMatrix()
# Convert to an IndexedRowMatrix.
indexedRowMat = mat.toIndexedRowMatrix()
```

Spark Machine Learning Library IX
↛ Data Types

```
# Convert to a BlockMatrix.
blockMat = mat.toBlockMatrix()
```

- A **BlockMatrix** is a distributed matrix backed by an RDD of MatrixBlock which is a tuple of (Int, Int, Matrix).

```
from pyspark.mllib.linalg import Matrices
from pyspark.mllib.linalg.distributed import BlockMatrix
# Create an RDD of sub-matrix blocks.
blocks = sc.parallelize([((0, 0), Matrices.dense(3, 2, [1, 2, 3, 4,
                         ((1, 0), Matrices.dense(3, 2, [7, 8, 9, 1(
# Create a BlockMatrix from an RDD of sub-matrix blocks.
mat = BlockMatrix(blocks, 3, 2)
# Get its size.
m = mat.numRows()  # 6
n = mat.numCols()  # 2
# Get the blocks as an RDD of sub-matrix blocks.
blocksRDD = mat.blocks
# Convert to a LocalMatrix.
localMat = mat.toLocalMatrix()
# Convert to an IndexedRowMatrix.
```

Spark Machine Learning Library X
↬ Data Types

```
indexedRowMat = mat.toIndexedRowMatrix()
# Convert to a CoordinateMatrix.
coordinateMat = mat.toCoordinateMatrix()
```

[2]http://www.scalanlp.org/

Spark Machine Learning Library I
↛ Basic Statistics

- **Summary statistics** colStats() returns an instance of MultivariateStatisticalSummary, which contains the column-wise max, min, mean, variance, and number of nonzeros, as well as the total count.

```
import numpy as np
from pyspark.mllib.stat import Statistics
mat = sc.parallelize(
    [np.array([1.0, 10.0, 100.0]),
     np.array([2.0, 20.0, 200.0]),
     np.array([3.0, 30.0, 300.0])]
)  # an RDD of Vectors
# Compute column summary statistics.
summary = Statistics.colStats(mat)
print(summary.mean())
print(summary.variance())
print(summary.numNonzeros())
```

Spark Machine Learning Library II
⇝ Basic Statistics

- **Correlations** The supported correlation methods are currently Pearson′ s and Spearman′ s correlation.

```
from pyspark.mllib.stat import Statistics
seriesX = sc.parallelize([1.0, 2.0, 3.0, 3.0, 5.0])  # a series
seriesY = sc.parallelize([11.0, 22.0, 33.0, 33.0, 555.0])
print("Correlation is: "
      + str(Statistics.corr(seriesX, seriesY, method="pearson"))
data = sc.parallelize(
    [np.array([1.0, 10.0, 100.0]),
     np.array([2.0, 20.0, 200.0]),
     np.array([5.0, 33.0, 366.0])]
)  # an RDD of Vectors
print(Statistics.corr(data, method="pearson"))
```

- **Random data generation** is useful for randomized algorithms, prototyping, and performance testing. spark.mllib supports generating random RDDs with i.i.d. values drawn from a given distribution: uniform, standard normal, or Poisson.

Spark Machine Learning Library III
↛ Basic Statistics

```
from pyspark.mllib.random import RandomRDDs
sc = ... # SparkContext
# Generate a random double RDD that contains 1 million i.i.d.
# values drawn from the standard normal distribution `N(0, 1)`,
# evenly distributed in 10 partitions.
u = RandomRDDs.normalRDD(sc, 1000000L, 10)
# Apply a transform to get a random double RDD following
# `N(1, 4)`.
v = u.map(lambda x: 1.0 + 2.0 * x)
```

Spark Programming API

- **R**
  http://spark.apache.org/docs/latest/api/R/index.html
- **Python**
  http://spark.apache.org/docs/latest/api/python/index.html
- **Scala**
  http://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.package