Quick Start

This tutorial provides a quick introduction to using Spark. We will first introduce the API through Spark's interactive shell (in Python or Scala), then show how to write applications in Java, Scala, and Python. See the programming guide for a more complete reference.

To follow along with this guide, first download a packaged release of Spark from the Spark website. Since we won't be using HDFS, you can download a package for any version of Hadoop.

Interactive Analysis with the Spark Shell

Basics

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 it by running the following in the Spark directory:

Scala Python

```
./bin/pyspark
```

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. Let's 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. Let's start with a few actions:

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

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

Now let's use a transformation. We will use the `filter` transformation to return a new RDD with a subset of the items in the file.

```
>>> linesWithSpark = textFile.filter(lambda line: "Spark" in line)
```

We can chain together transformations and actions:

```
>>> textFile.filter(lambda line: "Spark" in line).count() # How many lines contain "Spark"?
15
```

More on RDD Operations

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

Scala | Python

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

This first maps a line to an integer value, creating a new RDD. `reduce` is called on that RDD to find the largest line count. The arguments to `map` and `reduce` are Python anonymous functions (lambdas), but we can also pass any top-level Python function we want. For example, we'll define a `max` function to make this code easier to understand:

```
>>> def max(a, b):
...     if a > b:
...         return a
...     else:
...         return b
...

>>> textFile.map(lambda line: len(line.split())).reduce(max)
15
```

One common data flow pattern is MapReduce, as popularized by Hadoop. Spark can implement MapReduce flows easily:

```
>>> wordCounts = textFile.flatMap(lambda line: line.split()).map(lambda word: (word, 1)).reduceByKey(lambda a, b: a+b)
```

Here, we combined the `flatMap`, `map`, and `reduceByKey` transformations to compute the per-word counts in the file as an RDD of (string, int) pairs. To collect the word counts in our shell, we can use the `collect` action:

```
>>> wordCounts.collect()
[(u'and', 9), (u'A', 1), (u'webpage', 1), (u'README', 1), (u'Note', 1), (u'"local"', 1), (u'variable', 1), ...]
```

Caching

Spark also supports pulling data sets into a cluster-wide in-memory cache. This is very useful when data is accessed repeatedly, such as when querying a small "hot" dataset or when running an iterative algorithm like PageRank. As a simple example, let's mark our `linesWithSpark` dataset to be cached:

Scala | Python

```
>>> linesWithSpark.cache()

>>> linesWithSpark.count()
19

>>> linesWithSpark.count()
19
```

It may seem silly to use Spark to explore and cache a 100-line text file. The interesting part is that these same functions can be used on very large data sets, even when they are striped across tens or hundreds of nodes. You can also do this interactively by connecting `bin/pyspark` to a cluster, as described in the programming guide.

Self-Contained Applications

Suppose we wish to write a self-contained application using the Spark API. We will walk through a simple application in Scala (with sbt), Java (with Maven), and Python.

Scala Java Python

Now we will show how to write an application using the Python API (PySpark).

As an example, we'll create a simple Spark application, `SimpleApp.py`:

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

logFile = "YOUR_SPARK_HOME/README.md"  # Should be some file on your 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))
```

This program just counts the number of lines containing 'a' and the number containing 'b' in a text file. Note that you'll need to replace YOUR_SPARK_HOME with the location where Spark is installed. As with the Scala and Java examples, we use a SparkContext to create RDDs. We can pass Python functions to Spark, which are automatically serialized along with any variables that they reference. For applications that use custom classes or third-party libraries, we can also add code dependencies to `spark-submit` through its `--py-files` argument by packaging them into a .zip file (see `spark-submit --help` for details). `SimpleApp` is simple enough that we do not need to specify any code dependencies.

We can run this application using the `bin/spark-submit` script:

```
# Use spark-submit to run your application
$ YOUR_SPARK_HOME/bin/spark-submit \
  --master local[4] \
  SimpleApp.py
...
Lines with a: 46, Lines with b: 23
```

Where to Go from Here

Congratulations on running your first Spark application!

- For an in-depth overview of the API, start with the Spark programming guide, or see "Programming Guides" menu for other components.
- For running applications on a cluster, head to the deployment overview.
- Finally, Spark includes several samples in the `examples` directory (Scala, Java, Python, R). You can run them as follows:

```
# For Scala and Java, use run-example:
./bin/run-example SparkPi

# For Python examples, use spark-submit directly:
./bin/spark-submit examples/src/main/python/pi.py

# For R examples, use spark-submit directly:
./bin/spark-submit examples/src/main/r/dataframe.R
```