

Apache Spark Setup

```
rock — curl - ruby -W0 --disable=gems,did_you_mean,rubyopt /usr/local/Homebrew/Library/Homebrew/brew.rb upgrade — 204x54
(base) rocks-Air:~ rock$ brew upgrade && brew update
Updating Homebrew...
==> Upgrading 4 outdated packages:
little-cms2 2.9 -> 2.11
cmake 3.17.3 -> 3.18.4
laptonica 1.79.0 -> 1.80.0
automake 1.16.2 -> 1.16.2.1
==> Upgrading little-cms2 2.9 -> 2.11
==> Downloading https://homebrew.bintray.com/bottles/little-cms2-2.11.mojave.bottle.tar.gz
==> Downloading from https://d29vzk4ow0wi7.cloudfront.net/e05f0a487d224341eeb9fd9909f87517d7b27feb3cb914117acd9c60b76fcc?response-content-disposition=attachment%3Bfile
##### 100.0%
==> Pouring little-cms2-2.11.mojave.bottle.tar.gz
==> /usr/local/Cellar/little-cms2/2.11: 21 files, 1MB
==> 'brew cleanup' has not been run in 30 days, running now...
Removing: /Users/rock/Library/Caches/Homebrew/autoconf--2.69.mojave.bottle.4.tar.gz... (874.7KB)
Removing: /Users/rock/Library/Caches/Homebrew/automake--1.16.2.mojave.bottle.tar.gz... (948.7KB)
Removing: /Users/rock/Library/Caches/Homebrew/libtool--2.4.6.2.mojave.bottle.tar.gz... (1010.6KB)
Removing: /usr/local/Cellar/little-cms2/2.9... (18 files, 1MB)
Removing: /usr/local/Cellar/openssl@2.1/1.1.1g... (8,059 files, 18MB)
Removing: /Users/rock/Library/Caches/Homebrew/pkg-config--0.29.2.3.mojave.bottle.tar.gz... (234.2KB)
Removing: /Users/rock/Library/Caches/Homebrew/portable-ruby-2.6.3.2.yosemite.bottle.tar.gz... (9.1MB)
Removing: /Users/rock/Library/Caches/Homebrew/Cask/android-sdk--4333796.zip... (98.2MB)
Removing: /Users/rock/Library/Logs/Homebrew/jansson... (64B)
Removing: /Users/rock/Library/Logs/Homebrew/putty... (64B)
Removing: /Users/rock/Library/Logs/Homebrew/nghttp2... (64B)
Removing: /Users/rock/Library/Logs/Homebrew/apr-util... (64B)
Removing: /Users/rock/Library/Logs/Homebrew/jomallloc... (64B)
Removing: /Users/rock/Library/Logs/Homebrew/brotli... (64B)
Removing: /Users/rock/Library/Logs/Homebrew/c-ares... (64B)
Removing: /Users/rock/Library/Logs/Homebrew/httpd... (64B)
Removing: /Users/rock/Library/Logs/Homebrew/apr... (64B)
Removing: /Users/rock/Library/Logs/Homebrew/libevent... (64B)
Removing: /Users/rock/Library/Logs/Homebrew/openssl@1.1... (64B)
Removing: /Users/rock/Library/Logs/Homebrew/pcre... (64B)
Removing: /Users/rock/Library/Logs/Homebrew/libav... (64B)
Pruned 0 symbolic links and 2 directories from /usr/local
==> Upgrading cmake 3.17.3 -> 3.18.4
==> Downloading https://homebrew.bintray.com/bottles/cmake-3.18.4.mojave.bottle.tar.gz
==> Downloading from https://d29vzk4ow0wi7.cloudfront.net/a0b167ad7f2fbf6f6dbcca9d74cb09acbd7822c54873803e940abf04272f8028?response-content-disposition=attachment%3Bfile
```

```
Terminal Shell Edit View Window Help
rock — -bash — 204x54
(base) rocks-Air:~ rock$ Java -version
java version "14.0.2" 2020-07-14
Java(TM) SE Runtime Environment (build 14.0.2+12-46)
Java HotSpot(TM) 64-Bit Server VM (build 14.0.2+12-46, mixed mode, sharing)
(base) rocks-Air:~ rock$
```

```
rock — ruby - ruby -W0 --disable=gems,did_you_mean,rubyopt /usr/local/Homebrew/Library/Homebrew/brew.rb install scala — 204x54
(base) rocks-Air:~ rock$ brew install scala
Updating Homebrew...
==> Auto-updated Homebrew!
Updated 1 tap (homebrew/core).
==> Updated Formulae
Updated 3 formulae.

==> Downloading https://homebrew.bintray.com/bottles/openjdk-15.0.1.mojave.bottle.tar.gz
==> Downloading from https://d29vzk4ow0wi7.cloudfront.net/a4f00dc0b4c0bffe3828f32c82b0a6be41b23a69a7775a95cdbe9e01d9bdb68?response-content-disposition=attachment%3Bfile
##### 100.0%
==> Downloading https://downloads.lightbend.com/scala/2.13.4/scala-2.13.4.tgz
##### 100.0%
==> Installing dependencies for scala: openjdk
==> Installing scala dependency: openjdk
==> Pouring openjdk-15.0.1.mojave.bottle.tar.gz
```

```
Terminal Shell Edit View Window Help
rock — -bash — 204x54
(base) rocks-Air:~ rock$ scala -version
Scala code runner version 2.13.4 -- Copyright 2002-2020, LAMP/EPFL and Lightbend, Inc.
(base) rocks-Air:~ rock$
```



```
Terminal Shell Edit View Window Help
rock — java + python — 204x54

(base) rocks-Air:~ rock$ pyspark
Python 3.7.4 (default, Aug 13 2019, 15:17:50)
[Clang 4.0.1 (tags/RELEASE_401/final)] :: Anaconda, Inc. on darwin
Type "help", "copyright", "credits" or "license" for more information.
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by org.apache.spark.unsafe.Platform (file:/usr/local/Cellar/apache-spark/3.0.1/libexec/jars/spark-unsafe-3.0.1.jar) of method java.lang.ProcessImpl.()
WARNING: Please consider reporting this to the maintainers of org.apache.spark.unsafe.Platform
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations
WARNING: All illegal access operations will be denied in a future release
20/12/24 00:04:48 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Welcome to

  ____      _
 / ___|  __| | | |
| |  | |__| | | |
| |  |  __| | | |
| |  | |__| | | |
|_|  |____|_|_|_|

version 3.0.1

Using Python version 3.7.4 (default, Aug 13 2019 15:17:50)
SparkSession available as 'spark'.
>>> █
```

To be able to use PyPark locally on your machine you need to install findspark and pyspark If you use anaconda use the below commands:

```
#Find Spark Option 1:
    conda install -c conda-forge findspark
#Find Spark Option 2:
    conda install -c conda-forge/label/gcc7 findspark
#PySpark:
    conda install -c conda-forge pyspark

If you use regular python use pip install as:
    pip install findspark
    pip install pyspark
```

```
rock — -bash — 204x54
(base) rocks-Air:~ rock$ conda install -c conda-forge findspark
Collecting package metadata (current_repodata.json): done
Solving environment: done

==> WARNING: A newer version of conda exists. <==
  current version: 4.8.3
  latest version: 4.9.2

Please update conda by running

  $ conda update -n base -c defaults conda

## Package Plan ##

  environment location: /opt/anaconda3

  added / updated specs:
    - findspark

The following packages will be downloaded:



| package         | build          |        |             |
|-----------------|----------------|--------|-------------|
| conda-4.9.2     | py37hf985489_0 | 3.0 MB | conda-forge |
| findspark-1.3.0 | py_1           | 6 KB   | conda-forge |
| Total:          |                | 3.0 MB |             |



The following NEW packages will be INSTALLED:

  findspark                conda-forge/noarch::findspark-1.3.0-py_1

The following packages will be UPDATED:

  conda                    pkgs/main::conda-4.8.3-py37_0 --> conda-forge::conda-4.9.2-py37hf985489_0

Proceed ([y]/n)? y

Downloading and Extracting Packages
findspark-1.3.0           | 6 KB | #####
conda-4.9.2               | 3.0 MB | #####
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
(base) rocks-Air:~ rock$
```

```
rock — -bash — 204x54
(base) rocks-Air:~ rock$ conda install -c conda-forge/label/gcc7 findspark
Collecting package metadata (current_repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: /opt/anaconda3

  added / updated specs:
    - findspark

The following packages will be downloaded:



| package         | build |      |                        |
|-----------------|-------|------|------------------------|
| findspark-1.3.0 | py_1  | 6 KB | conda-forge/label/gcc7 |
| Total:          |       | 6 KB |                        |



The following packages will be SUPERSEDED by a higher-priority channel:

  findspark                conda-forge --> conda-forge/label/gcc7

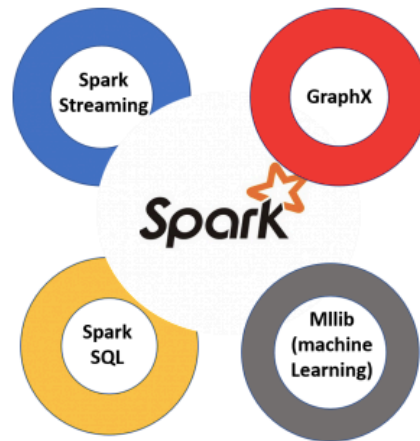
Proceed ([y]/n)? y

Downloading and Extracting Packages
findspark-1.3.0           | 6 KB | #####
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
(base) rocks-Air:~ rock$
```

What is Apache Spark?

[Apache Spark](#) is one of the hottest new trends in the technology domain. It is the framework with probably the highest potential to realize the fruit of the marriage between Big Data and Machine Learning.

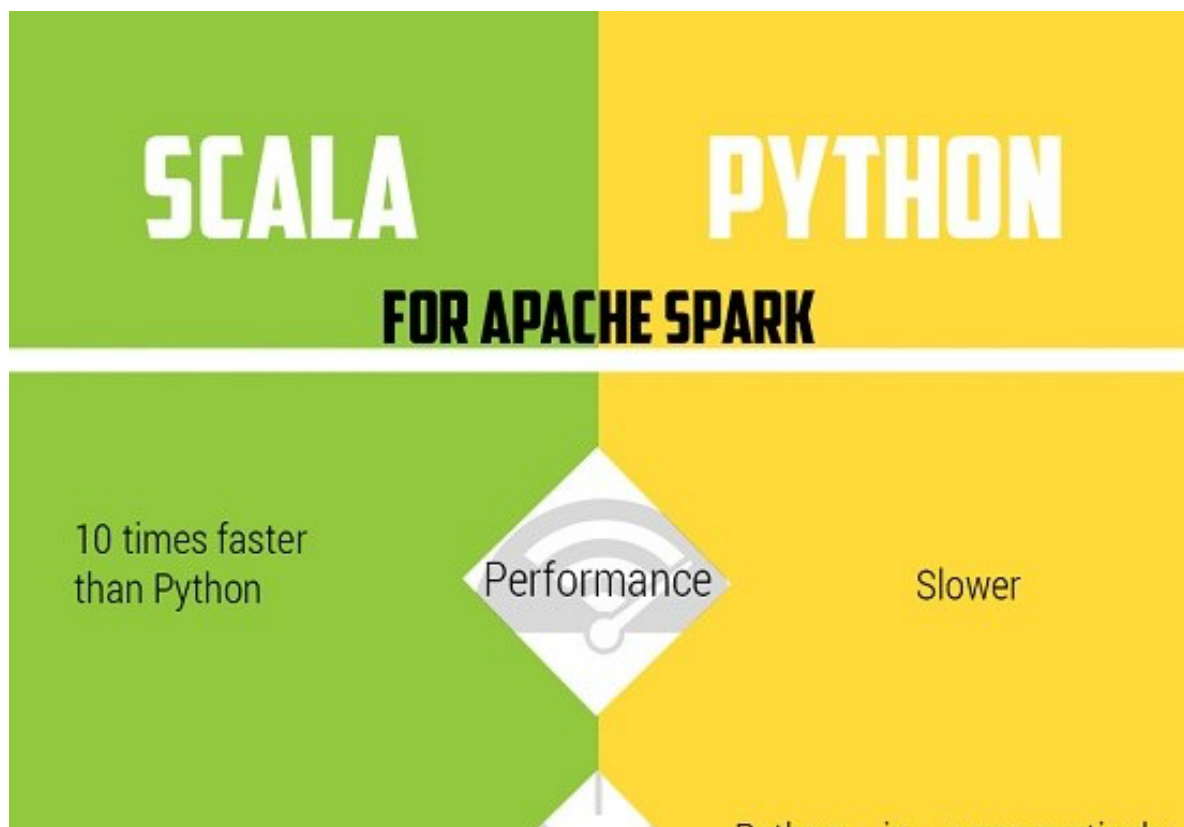
It runs fast (up to 100x faster than traditional [Hadoop MapReduce](#) due to in-memory operation, offers robust, distributed, fault-tolerant data objects (called [RDD](#)), and integrates beautifully with the world of machine learning and graph analytics through supplementary packages like [Mlib](#) and [GraphX](#).

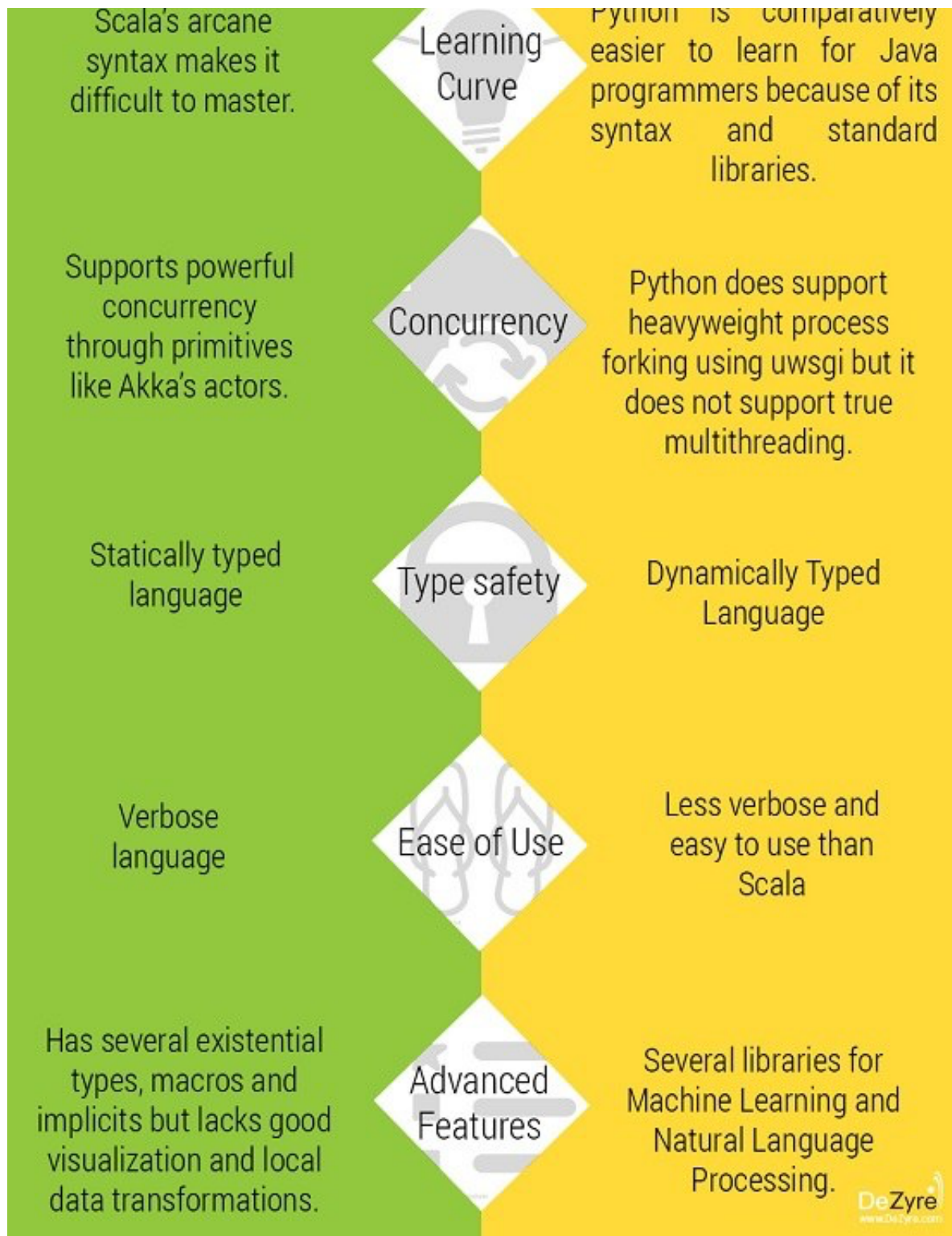


What is Scala used for?




What is **Scala** used for? A lot of things, ranging from machine learning to web apps. As a high-level general purpose language, **Scala** boasts an extensive range of possible applications. **Scala** allows developers to make good **use** of standard JVM features and Java libraries.





Python vs Scala

Comparison Chart

Python	Scala
It is a dynamically typed language in which the type checking is done at run-time.	It is a statically typed language in which the type checking is done at compile-time.
It does not support heavyweight process forking so it is not the preferred choice of language for highly concurrent and scalable systems.	It offers multiple asynchronous libraries and reactive cores that help in quick integration of databases in highly scalable systems.
It was originally conceived as an object-oriented language & can be used as a procedural language.	It is the mix of object-oriented and functional programming language.
It is generally easier to learn and use than other programming languages.	It is less difficult to use and learn than Python. 

Spark is implemented on [Hadoop/HDFS](#) and written mostly in [Scala](#), a functional programming language, similar to Java. In fact, Scala needs the latest Java installation on your system and runs on JVM. However, for most beginners, Scala is not a language that they learn first to venture into the world of data science. Fortunately, Spark provides a wonderful Python integration, called **PySpark**, which lets Python programmers to interface with the Spark framework and learn how to manipulate data at scale and work with objects and algorithms over a distributed file system.

In this article, we will learn the basics of PySpark. There are a lot of concepts (constantly evolving and introduced), and therefore, we just focus on fundamentals with a few simple examples. Readers are encouraged to build on these and explore more on their own.

The Short History of Apache Spark

Apache Spark started as a research project at the UC Berkeley AMPLab in 2009, and was open sourced in early 2010. It was a class project at UC Berkeley. Idea was to build a cluster management framework, which can support different kinds of cluster computing systems. Many of the ideas behind the system were presented in various research papers over the years. After being released, Spark grew into a broad developer community, and moved to the Apache Software Foundation in 2013. Today, the project is developed collaboratively by a community of hundreds of developers from hundreds of organizations.

Spark is Not a Programming Language

One thing to remember is that Spark is not a programming language like Python or Java. It is a general-purpose distributed data processing engine, suitable for use in a wide range of circumstances. It is particularly useful for big data processing both at scale and with high speed.

Application developers and data scientists generally incorporate Spark into their applications to rapidly query, analyze, and transform data at scale. Some of the tasks that are most frequently associated with Spark, include, – ETL and SQL batch jobs across large data sets (often of terabytes of size), – processing of streaming data from IoT devices and nodes, data from various sensors, financial and transactional systems of all kinds, and – machine learning tasks for e-commerce or IT applications.

At its core, Spark builds on top of the Hadoop/HDFS framework for handling distributed files. It is mostly implemented with Scala, a functional language variant of Java. There is a core Spark data processing engine, but on top of that, there are many libraries developed for SQL-type query analysis, distributed machine learning, large-scale graph computation, and streaming data processing. Multiple programming languages are supported by Spark in the form of easy interface libraries: Java, Python, Scala, and R.

Spark Uses the MapReduce Paradigm for Distributed Processing

The basic idea of distributed processing is to divide the data chunks into small manageable pieces (including some filtering and sorting), bring the computation close to the data i.e. use small nodes of a large cluster for specific jobs and then re-combine them back. The dividing portion is called the 'Map' action and the recombination is called the 'Reduce' action. Together, they make the famous 'MapReduce' paradigm, which was introduced by Google around 2004 (see the [original paper here](#)).

For example, if a file has 100 records to be processed, 100 mappers can run together to process one record each. Or maybe 50 mappers can run together to process two records each. After all the mappers complete processing, the framework shuffles and sorts the results before passing them on to the reducers. A reducer cannot start while a mapper is still in progress. All the map output values that have the same key are assigned to a single reducer, which then aggregates the values for that key.

How to Setup PySpark

If you're already familiar with Python and libraries such as Pandas and Numpy, then PySpark is a great extension/framework to learn in order to create more scalable, data-intensive analyses and pipelines by utilizing the power of Spark in the background.

The exact process of installing and setting up PySpark environment (on a standalone machine) is somewhat involved and can vary slightly depending on your system and environment. The goal is to get your regular Jupyter data science environment working with Spark in the background using the PySpark package.

[This article](#) on Medium provides more details on the step-by-step setup process.



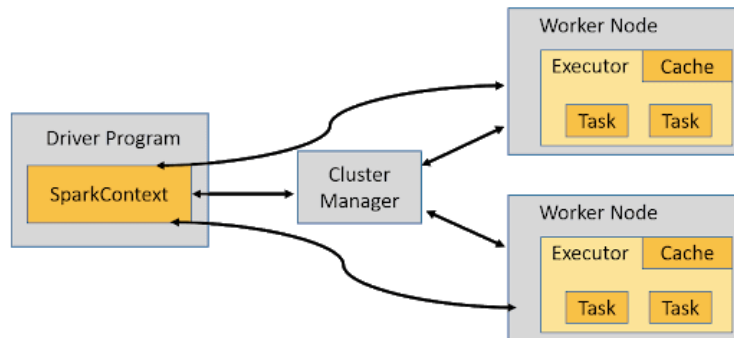
Alternatively, you can use Databricks setup for practicing Spark. This company was created by the original creators of Spark and have an excellent ready-to-launch environment to do distributed analysis with Spark.

But the idea is always the same. You are distributing (and replicating) your large dataset in small fixed chunks over many nodes. You then bring the compute engine close to them so that the whole operation is parallelized, fault-tolerant and scalable.

By working with PySpark and Jupyter notebook, you can learn all these concepts without spending anything on AWS or Databricks platform. You can also easily interface with SparkSQL and MLlib for database manipulation and machine learning. It will be much easier to start working with real-life large clusters if you have internalized these concepts beforehand!

Resilient Distributed Dataset (RDD) and SparkContext

Many Spark programs revolve around the concept of a resilient distributed dataset (RDD), which is a fault-tolerant collection of elements that can be operated on in parallel. SparkContext resides in the Driver program and manages the distributed data over the worker nodes through the cluster manager. The good thing about using PySpark is that all this complexity of data partitioning and task management is handled automatically at the back and the programmer can focus on the specific analytics or machine learning job itself.



There are two ways to create RDDs—parallelizing an existing collection in your driver program, or referencing a dataset in an external storage system, such as a shared file- system, HDFS, HBase, or any data source offering a Hadoop InputFormat.

For illustration with a Python-based approach, we will give examples of the first type here. We can create a simple Python array of 20 random integers (between 0 and 10), using Numpy random.randint(), and then create an RDD object as following,

```
1. from pyspark import SparkContext
2. import numpy as np
3. sc=SparkContext(master="local[4]")
4. lst=np.random.randint(0,10,20)
5. A=sc.parallelize(lst)
```

Note the '4' in the argument. It denotes 4 computing cores (in your local machine) to be used for this SparkContext object. If we check the type of the RDD object, we get the following,

```
1. type(A)
2. >> pyspark.rdd.RDD
```

Opposite to parallelization is the collection (with collect()) which brings all the distributed elements and returns them to the head node.

```
1. A.collect()
2. >> [4, 8, 2, 2, 4, 7, 0, 3, 3, 9, 2, 6, 0, 0, 1, 7, 5, 1, 9, 7]
```

But A is no longer a simple Numpy array. We can use the `glom()` method to check how the partitions are created.

```
1. A.glom().collect()
2. >> [[4, 8, 2, 2, 4], [7, 0, 3, 3, 9], [2, 6, 0, 0, 1], [7, 5, 1, 9, 7]]
```

Now stop the SC and reinitialize it with 2 cores and see what happens when you repeat the process.

```
1. sc.stop()
2. sc=SparkContext(master="local[2]")
3. A = sc.parallelize(lst)
4. A.glom().collect()
5. >> [[4, 8, 2, 2, 4, 7, 0, 3, 3, 9], [2, 6, 0, 0, 1, 7, 5, 1, 9, 7]]
```

The RDD is now distributed over two chunks, not four!

You have learned about the first step in distributed data analytics i.e. controlling how your data is partitioned over smaller chunks for further processing