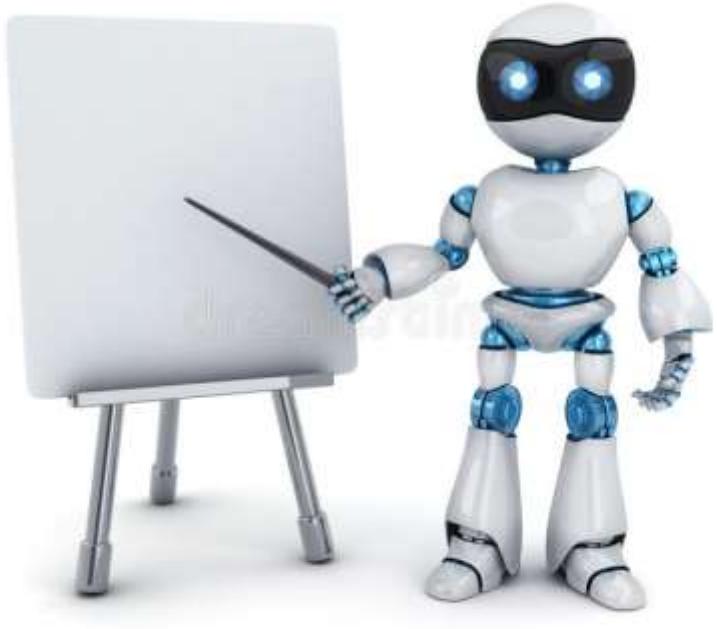


PySpark

Overview

Datamites

PySpark Introduction

PySpark is an interface for Spark in Python. It provides excellent API support to PySpark shell for interactively analysing the data in a distributed condition.

PySpark supports DataFrame, Streaming, MLlib (Machine Learning) ,Spark Core,Spark SQL

© DataMites®. All Rights Reserved | www.datamites.com

Pyspark Configuration

```
>>> from pyspark.conf import SparkConf
>>> from pyspark.context import SparkContext
>>> conf = SparkConf()
>>> conf.setMaster("local").setAppName("My app")
<pyspark.conf.SparkConf object at ...>
>>> conf.get("spark.master")
'local'
>>> conf.get("spark.app.name")
'My app'
>>> sc = SparkContext(conf=conf)
>>> sc.master
'local'
>>> sc.appName
'My app'
>>> sc.sparkHome is None
True
```

Datamites

Pyspark Configuration

```
>>> conf = SparkConf(loadDefaults=False)
>>> conf.setSparkHome("/path")
<pyspark.conf.SparkConf object at ...>
>>> conf.get("spark.home")
'/path'
>>> conf.setExecutorEnv("VAR1", "value1")
<pyspark.conf.SparkConf object at ...>
>>> conf.setExecutorEnv(pairs = [("VAR3", "value3"), ("VAR4", "value4")])
<pyspark.conf.SparkConf object at ...>
>>> conf.get("spark.executorEnv.VAR1")
'value1'
>>> print(conf.toDebugString())
spark.executorEnv.VAR1=value1
spark.executorEnv.VAR3=value3
spark.executorEnv.VAR4=value4
spark.home=/path
>>> for p in sorted(conf.getAll(), key=lambda p: p[0]):
...     print(p)
('spark.executorEnv.VAR1', 'value1')
('spark.executorEnv.VAR3', 'value3')
('spark.executorEnv.VAR4', 'value4')
('spark.home', '/path')
>>> conf._jconf.setExecutorEnv("VAR5", "value5")
JavaObject id...
>>> print(conf.toDebugString())
spark.executorEnv.VAR1=value1
spark.executorEnv.VAR3=value3
spark.executorEnv.VAR4=value4
spark.executorEnv.VAR5=value5
spark.home=/path
```

Datamites

© DataMites®. All Rights Reserved | www.datamites.com

Resilient distributed datasets (RDD)

RDD stands for Resilient Distributed Dataset in PySpark. It is a fundamental data structure in Spark and the core of the Spark programming model. It represents an immutable, distributed collection of objects that can be processed in parallel.

RDDs are created by parallelizing an existing collection of objects or by loading an external dataset. Once created, RDDs can be transformed and processed in parallel using a variety of operations such as filter, map, reduce, and join. RDDs can also be cached in memory for faster access.

RDDs are fault-tolerant, meaning that they can recover from node failures. When a node fails, the data stored on that node is automatically replicated to other nodes in the cluster, so that the RDD can continue to be processed without interruption.

© DataMites®. All Rights Reserved | www.datamites.com

Resilient distributed datasets (RDD)

RDDs have been replaced by DataFrames and Dataset API in Spark 2.0. But RDD API is still supported for backward compatibility.

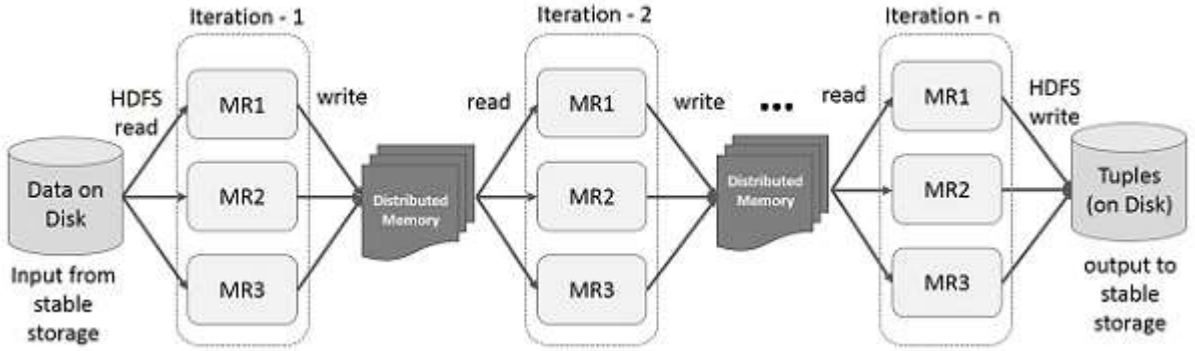
RDDs have the following characteristics:

Resilient: Able to recover from node failures
Distributed: Stored across multiple nodes in a cluster
Dataset: Immutable collection of objects
Parallel: Able to be processed in parallel
You can use the PySpark library to create RDDs and perform operations on them.

Datamites

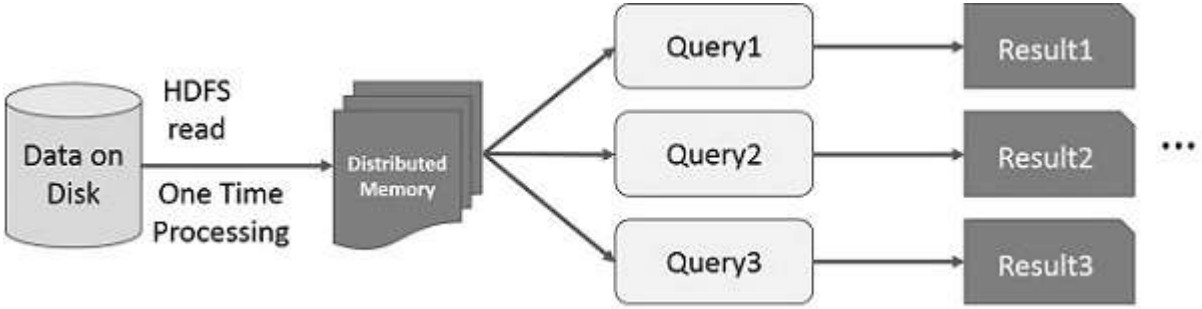
Resilient distributed datasets (RDD)

Iterative Operations on Spark RDD



Resilient distributed datasets (RDD)

Interactive Operations on Spark RDD



Datamites