Spark Context (sc) helps in :

-> Task Creator

-> Scheduler

-> Data Locality

-> Fault Tolerance

Get JDK :

<http://download.oracle.com/otn-pub/java/jdk/8u152-b16/aa0333dd3019491ca4f6ddbe78cdb6d0/jdk-8u152-linux-x64.rpm>

[root@my ~]# rpm -ivh jdk-8u152-linux-x64.rpm

[root@my ~]# export JAVA\_HOME=/usr/java/jdk1.8.0\_152/

[root@my ~]# export PATH=/usr/java/jdk1.8.0\_152/bin/:$PATH

[root@my ~]# java -version

java version "1.8.0\_152"

[root@my ~]# cat /root/.bashrc

export JAVA\_HOME=/usr/java/jdk1.8.0\_152/

export PATH=/usr/java/jdk1.8.0\_152/bin/:$PATH

Get Scala:

<https://downloads.lightbend.com/scala/2.12.4/scala-2.12.4.rpm>

[root@my ~]# rpm -ivh scala-2.12.4.rpm

Get spark :

<https://d3kbcqa49mib13.cloudfront.net/spark-2.2.0-bin-hadoop2.7.tgz>

[root@my ~]# tar -xvzf spark-2.2.0-bin-hadoop2.7.tgz

[root@my ~]# mv spark-2.2.0-bin-hadoop2.7 /spark

# export SPARK\_LOCAL\_IP=127.0.0.1

[root@my bin]# /spark/bin/spark-shell

**scala> val myFile = sc.textFile("file:///etc/passwd")**

myFile: org.apache.spark.rdd.RDD[String] = file:///etc/passwd MapPartitionsRDD[3] at textFile at <console>:24

**scala> myFile.first**

res1: String = root:x:0:0:root:/root:/bin/bash

**scala> val myFile = sc.textFile("file:///spark/README.md")**

myFile: org.apache.spark.rdd.RDD[String] = file:///spark/README.md MapPartitionsRDD[1] at textFile at <console>:24

**scala> myFile.first**

res0: String = # Apache Spark

**scala> val tokenizedFileData = myFile.flatMap(line => line.split(" "))**

tokenizedFileData: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[2] at flatMap at <console>:26

**scala> val countPrep = tokenizedFileData.map(word => (word , 1))**

countPrep: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[3] at map at <console>:28

**scala> val counts = countPrep.reduceByKey((accumValue, newValue)=> accumValue + newValue)**

counts: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[5] at reduceByKey at <console>:30

**scala> val sortedCounts = counts.sortBy(kvPair => kvPair.\_2 , false)**

sortedCounts: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[9] at sortBy at <console>:32

**scala> sortedCounts.saveAsTextFile("file:///tmp/WordCountOutput")**

All the above map reduce can be done by single “countByValue” api method :

**scala> tokenizedFileData.countByValue**

res6: scala.collection.Map[String,Long] = Map(site, -> 1, Please -> 4, Contributing -> 1, GraphX -> 1, project. -> 1, "" -> 71, for -> 12, find -> 1, Apache -> 1, package -> 1, Hadoop, -> 2, review -> 1, Once -> 1, For -> 3, name -> 1, this -> 1, protocols -> 1, Hive -> 2, in -> 6, "local[N]" -> 1, MASTER=spark://host:7077 -> 1, have -> 1, your -> 1, are -> 1, is -> 6, HDFS -> 1, Data. -> 1, built -> 1, thread, -> 1, examples -> 2, developing -> 1, using -> 5, system -> 1, than -> 1, Shell -> 2, mesos:// -> 1, 3"](https://cwiki.apache.org/confluence/display/MAVEN/Parallel+builds+in+Maven+3). -> 1, easiest -> 1, This -> 2, -T -> 1, [Apache -> 1, N -> 1, <class> -> 1, different -> 1, "local" -> 1, README -> 1, online -> 1, spark:// -> 1, return -> 2, Note -> 1, if -> 4, project -> 1, Scal...

Run Spark prog directly from scala :

[root@my ~]# scalac -classpath /spark/jars/spark-core\_2.11-2.2.0.jar my.scala

scala> **sc.parallelize(1 to 100)**

res0: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[0] at parallelize at <console>:25

**scala> res0.collect**

res2: Array[Int] = Array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100)

**scala> sc.parallelize**

def parallelize[T](seq: Seq[T],**numSlices: Int**)(implicit evidence$1: scala.reflect.ClassTag[T]): org.apache.spark.rdd.RDD[T]

**scala> sc.range**

def range(start: Long,end: Long,**step: Long,numSlices: In**t): org.apache.spark.rdd.RDD[Long]

Load all the files from directory :

**scala> sc.wholeTextFiles**

def wholeTextFiles(path: String,minPartitions: Int): org.apache.spark.rdd.RDD[(String, String)]

[root@workstation ~]# **export SPARK\_LOCAL\_IP=127.0.0.1**

scala> val t = sc.textFile("/etc/passwd")

scala> t.toDebugString

No. of cpu :

scala> sc.defaultParallelism

scala> val x = sc.parallelize(1 to 1000, 5)

scala> x.collect()

**scala> val data=sc.textFile("../../spark/examples/src/main/resources/people.json")**

data: org.apache.spark.rdd.RDD[String] = ../../spark/examples/src/main/resources/people.json MapPartitionsRDD[4] at textFile at <console>:24

**scala> data.take(10)**

res3: Array[String] = Array({"name":"Michael"}, {"name":"Andy", "age":30}, {"name":"Justin", "age":19})

**scala> data.count()**

res4: Long = 3

**scala> data.first()**

res5: String = {"name":"Michael"}

**scala> data.collect()**

res6: Array[String] = Array({"name":"Michael"}, {"name":"Andy", "age":30}, {"name":"Justin", "age":19})

**scala> val user=sc.textFile("/etc/passwd")**

user: org.apache.spark.rdd.RDD[String] = /etc/passwd MapPartitionsRDD[8] at textFile at <console>:24

**scala> val bashuser=user.filter(line => line.contains("bash"))**

bashuser: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[9] at filter at <console>:26

**scala> bashuser.collect()**

**scala> val numbersRDD = sc.parallelize(List(1,2,3,4))**

numbersRDD: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[10] at parallelize at <console>:24

**scala> val squareRDD = numbersRDD.map(x => x\*x).collect()**

squareRDD: Array[Int] = Array(1, 4, 9, 16)

**scala> squareRDD foreach println**

1

4

9

16

**scala> val numbersRDD = sc.parallelize(List(1,2,3,4))**

numbersRDD: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[12] at parallelize at <console>:24

**scala> val filterRDD = numbersRDD.filter(x => x != 2 ).collect()**

filterRDD: Array[Int] = Array(1, 3, 4)

**scala> filterRDD foreach println**

1

3

4

**scala> val linesRDD = sc.parallelize(List("hello world", "how are you"))**

linesRDD: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[14] at parallelize at <console>:24

**scala> linesRDD.collect()**

res12: Array[String] = Array(hello world, how are you)

**scala> val wordsRDD = linesRDD.flatMap(x => x.split(" ")).collect()**

wordsRDD: Array[String] = Array(hello, world, how, are, you)

**scala> wordsRDD foreach println**

hello

world

how

are

you

**[root@workstation ~]# /spark/bin/spark-shell --master local[2]**

**scala> sc.master**

--------------------------------------

**scala> val path = "file:///root/nyc.csv"**

path: String = file:///root/nyc.csv

**scala> val data = sc.textFile(path)**

data: org.apache.spark.rdd.RDD[String] = file:///root/nyc.csv MapPartitionsRDD[1] at textFile at <console>:26

**scala> data.take(2)**

res3: Array[String] = Array(OBJECTID,Identifier,Occurrence Date,Day of Week,Occurrence Month,Occurrence Day,Occurrence Year,Occurrence Hour,CompStat Month,CompStat Day,CompStat Year,Offense,Offense Classification,Sector,Precinct,Borough,Jurisdiction,XCoordinate,YCoordinate,Location 1, 1,f070032d,09/06/1940 07:30:00 PM,Friday,Sep,6,1940,19,9,7,2010,BURGLARY,FELONY,D,66,BROOKLYN,N.Y. POLICE DEPT,987478,166141,"(40.6227027620001, -73.9883732929999)")

**scala> val header = data.first()**

header: String = OBJECTID,Identifier,Occurrence Date,Day of Week,Occurrence Month,Occurrence Day,Occurrence Year,Occurrence Hour,CompStat Month,CompStat Day,CompStat Year,Offense,Offense Classification,Sector,Precinct,Borough,Jurisdiction,XCoordinate,YCoordinate,Location 1

**scala> val dataWithoutHeader = data.filter(x => x != header)**

dataWithoutHeader: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[2] at filter at <console>:30

**scala> dataWithoutHeader.first()**

res4: String = 1,f070032d,09/06/1940 07:30:00 PM,Friday,Sep,6,1940,19,9,7,2010,BURGLARY,FELONY,D,66,BROOKLYN,N.Y. POLICE DEPT,987478,166141,"(40.6227027620001, -73.9883732929999)"

**scala> dataWithoutHeader.map(x=> x.split(",")).take(10)**

**scala> val fields = header.replace(" " , "\_").replace("/", "\_").split(",")**

fields: Array[String] = Array(OBJECTID, Identifier, Occurrence\_Date, Day\_of\_Week, Occurrence\_Month, Occurrence\_Day, Occurrence\_Year, Occurrence\_Hour, CompStat\_Month, CompStat\_Day, CompStat\_Year, Offense, Offense\_Classification, Sector, Precinct, Borough, Jurisdiction, XCoordinate, YCoordinate, Location\_1)

DF Scala :

----------

scala> val df = spark.read.json("/spark/examples/src/main/resources/people.json")

scala> df.show()

*// This import is needed to use the $-notation*

scala> import spark.implicits.\_

scala> df.printSchema()

scala> df.select("name","age").show()

scala> df.select($"name",$"age" + 1).show()

scala> df.filter($"age" > 21).show()

scala> df.groupBy("age").count().show()

scala> df.createOrReplaceTempView("people")

scala> val sqlDF = spark.sql("SELECT \* FROM people")

scala> sqlDF.show()

**val** textFile **=** sc.textFile("hdfs://...")  
  
*// Creates a DataFrame having a single column named "line"*  
**val** df **=** textFile.toDF("line")  
**val** errors **=** df.filter(col("line").like("%ERROR%"))  
*// Counts all the errors*  
errors.count()  
*// Counts errors mentioning MySQL*  
errors.filter(col("line").like("%MySQL%")).count()  
*// Fetches the MySQL errors as an array of strings*  
errors.filter(col("line").like("%MySQL%")).collect()

scala> case class User(name: String, password: String, uid:Int, gid:Int, comment:String , homedir:String, shell:String)

scala> val userDF = sc.textFile("/etc/passwd").map(\_.split(":")).map( fields => User(fields(0),fields(1), fields(2).toInt, fields(3).toInt, fields(4), fields(5), fields(6) )).toDF()

scala> userDF.groupBy("shell").agg(max("uid"))

scala> userDF.groupBy("shell").agg(max("uid") , sum("gid") ).show()

scala> userDF.createOrReplaceTempView("people")

scala> val userbasicDF = spark.sql("SELECT name,uid FROM user WHERE uid BETWEEN 1000 AND 1002")

scala> userbasicDF.show()

scala> val usersDF = spark.read.load("/spark/examples/src/main/resources/users.parquet")

scala> usersDF.select("name", "favorite\_color").write.save("namesAndFavColors.parquet")

scala> val peopleDF = spark.read.format("json").load("/spark/examples/src/main/resources/people.json")

scala> peopleDF.select("name", "age").write.format("parquet").save("namesAndAges.parquet")

scala> val sqlDF = spark.sql("SELECT \* FROM parquet.`/spark/examples/src/main/resources/users.parquet`")

scala> import spark.implicits.\_

scala> val peopleDF = spark.read.json("/spark/examples/src/main/resources/people.json")

scala> peopleDF.write.parquet("people.parquet")

scala> val parquetFileDF = spark.read.parquet("people.parquet")

scala> parquetFileDF.createOrReplaceTempView("parquetFile")

scala> val namesDF = spark.sql("SELECT name FROM parquetFile WHERE age BETWEEN 13 AND 19")

scala> namesDF.map(attributes => "Name: " + attributes(0)).show()

+------------+

| value|

+------------+

|Name: Justin|

+------------+

**val** people = spark.read.parquet("...")  
**val** department = spark.read.parquet("...")  
  
people.filter("age > 30")  
 .join(department, people("deptId") === department("id"))  
 .groupBy(department("name"), "gender")  
 .agg(avg(people("salary")), max(people("age")))

# yum install mariadb-server

<https://dev.mysql.com/get/Downloads/Connector-J/mysql-connector-java-5.1.44.tar.gz>

scala> val dataframe\_mysql = spark.read.format("jdbc").option("url", "jdbc:mysql://localhost/lw").option("driver", "com.mysql.jdbc.Driver").option("dbtable", "student").option("user", "root").option("password", "redhat").load().show()

<https://jaceklaskowski.gitbooks.io/mastering-apache-spark/spark-sql-udfs.html>

You define a new UDF by defining a Scala function as an input parameter of [udf function](https://jaceklaskowski.gitbooks.io/mastering-apache-spark/spark-sql-udfs.html#udf-function). It accepts Scala functions of up to 10 input parameters.

val dataset = Seq((0, "hello"), (1, "world")).toDF("id", "text")  
  
// Define a regular Scala function  
val upper: String => String = \_.toUpperCase  
  
// Define a UDF that wraps the upper Scala function defined above  
// You could also define the function in place, i.e. inside udf  
// but separating Scala functions from Spark SQL's UDFs allows for easier testing  
import org.apache.spark.sql.functions.udf  
val upperUDF = udf(upper)  
  
// Apply the UDF to change the source dataset  
scala> dataset.withColumn("upper", upperUDF('text)).show

Performance tuning :

<https://jaceklaskowski.gitbooks.io/mastering-apache-spark/spark-sql-performance-tuning-groupBy-aggregation.html>

Joined :

----------

scala> case class Person(id: Long, name: String, cityId: Long)

defined class Person

scala> case class City(id: Long, name: String)

defined class City

scala> val people = Seq(Person(0, "Agata", 0), Person(1, "Iweta", 0)).toDS

people: org.apache.spark.sql.Dataset[Person] = [id: bigint, name: string ... 1 more field]

scala> val cities = Seq(City(0, "Warsaw"), City(1, "Washington")).toDS

cities: org.apache.spark.sql.Dataset[City] = [id: bigint, name: string]

scala> val joined = people.joinWith(cities, people("cityId") === cities("id"))

joined: org.apache.spark.sql.Dataset[(Person, City)] = [\_1: struct<id: bigint, name: string ... 1 more field>, \_2: struct<id: bigint, name: string>]

scala> joined.printSchema

root

|-- \_1: struct (nullable = false)

| |-- id: long (nullable = false)

| |-- name: string (nullable = true)

| |-- cityId: long (nullable = false)

|-- \_2: struct (nullable = false)

| |-- id: long (nullable = false)

| |-- name: string (nullable = true)

scala> joined.show

+-----------+----------+

| \_1| \_2|

+-----------+----------+

|[0,Agata,0]|[0,Warsaw]|

|[1,Iweta,0]|[0,Warsaw]|

+-----------+----------+

scala> val left = Seq((0, "zero"), (1, "one")).toDF("id", "left")

left: org.apache.spark.sql.DataFrame = [id: int, left: string]

scala> val right = Seq((0, "zero"), (2, "two"), (3, "three")).toDF("id", "right")

right: org.apache.spark.sql.DataFrame = [id: int, right: string]

scala> left.join(right, "id").show

+---+----+-----+

| id|left|right|

+---+----+-----+

| 0|zero| zero|

+---+----+-----+

scala> left.join(right, "id").explain

== Physical Plan ==

\*Project [id#63, left#64, right#74]

+- \*BroadcastHashJoin [id#63], [id#73], Inner, BuildRight

:- LocalTableScan [id#63, left#64]

+- BroadcastExchange HashedRelationBroadcastMode(List(cast(input[0, int, false] as bigint)))

+- LocalTableScan [id#73, right#74]

scala> left.join(right, Seq("id"), "fullouter").show

[Stage 14:==============> [Stage 14:====================> [Stage 14:==========================> [Stage 14:==================================> [Stage 14:=========================================[Stage 14:=========================================[Stage 14:========================================= [Stage 17:===========================> [Stage 17:=======================================> [Stage 17:========================================= +---+----+-----+

| id|left|right|

+---+----+-----+

| 1| one| null|

| 3|null|three|

| 2|null| two|

| 0|zero| zero|

+---+----+-----+

<https://jaceklaskowski.gitbooks.io/mastering-apache-spark/spark-sql-joins.html>

<https://jaceklaskowski.gitbooks.io/mastering-apache-spark/spark-sql-Expression-WindowExpression.html>

Implicit Conversion :

**scala> 1.plus(1)**

<console>:25: error: value plus is not a member of Int

1.plus(1)

^

**scala> case class IntExtensions(value:Int ){**

**| def plus(operand: Int) = value + operand**

**| }**

defined class IntExtensions

**scala> IntExtensions(1).plus(1)**

res8: Int = 2

**scala> import scala.language.implicitConversions**

import scala.language.implicitConversions

**scala> implicit def intToIntExtensions(value:Int) = {**

**| IntExtensions(value)**

**| }**

intToIntExtensions: (value: Int)IntExtensions

**scala> 1.plus(1)**

res9: Int = 2

-----------------------------------------------

https://www.balabit.com/blog/wp-content/uploads/2016/12/StarWars.csv

**case** **class** **Characters**(name**:** String,   
 height**:** Integer,   
 weight**:** Option[Integer],   
 eyecolor**:** Option[String],   
 haircolor**:** Option[String],   
 jedi**:** String,  
 species**:** String)  
org.apache.spark.sql.catalyst.encoders.**OuterScopes**.addOuterScope(**this**)

Columns weight, eyecolor and haircolor are Option types, so they are allowed to contain missing values. All the other fields must have values from the type defined in the case class.

**scala> val characters\_ds = spark.read.option("header", "true").option("delimiter",";").option("inderSchema","true").csv("/root/StarWars.csv")**

characters\_ds.show()

scala> case class Characters\_BadType(name: String,

| height: Integer,

| weight: Integer,

| eyecolor: String,

| haircolor: String,

| jedi: String,

| species: String)

defined class Characters\_BadType

scala> org.apache.spark.sql.catalyst.encoders.OuterScopes.addOuterScope(this)

scala> val characters\_BadType\_ds = spark.read.option("header", "true").option("delimiter",";").option("inderSchema","true").csv("/root/StarWars.csv")

characters\_BadType\_ds: org.apache.spark.sql.DataFrame = [name: string, height: string ... 5 more fields]

<https://www.balabit.com/blog/spark-scala-dataset-tutorial/>

<https://grzegorzgajda.gitbooks.io/spark-examples/content/regression/regression-algorithms.html>

<https://jaceklaskowski.gitbooks.io/mastering-apache-spark/exercises/spark-first-app.html>

Maven Pom file:

---------------------

<https://drive.google.com/file/d/0B-ur4R5mlgGLRGNETk5nX1JUNVE/view>

<http://www.devinline.com/2016/01/apache-spark-setup-in-eclipse-scala-ide.html>

<http://scala-ide.org/download/sdk.html>





