# Giới thiệu

Trang chủ: <https://spark.apache.org/>

- spark là engine dùng để xử lí, phân tích dữ liệu lớn

- spark có thể “chạy trên" hadoop-yarn, mesos, ec2, kubernetes

- spark có thể truy cập dữ liệu từ hdfs, alluxio, cassandra, hbase, hive

Tổng quan quá trình hoạt động

- bạn sẽ code một app và package (build) thành job (một file jar bình thường)

- bạn sau đó sẽ upload job (jar) này lên server và dùng lệnh spark-submit để đẩy job

- server sau đó sẽ nhận job của bạn và chạy

# Nội dung

Tham khảo

- <https://spark.apache.org/docs/latest/quick-start.html>

- <https://spark.apache.org/examples.html>

- <https://spark.apache.org/docs/latest/api/scala/org/apache/spark/index.html>

## 1. Khởi tạo spark

### 1.1 SparkContext

- khởi tạo trực tiếp với new

val spark = new SparkContext(“yarn”,”test”)

- khởi tạo thông qua SparkConf

val sparkConf = new SparkConf()

sparkConf.setMaster(“yarn")

sparkConf.setAppName(“test”)

val spark = new SparkContext(sparkConf)

### 1.2 SparkSession

- khởi tạo trực tiếp với builder

val spark = SparkSession.builder().master(“yarn”).appName(“test”).getOrCreate()

- khởi tạo thông qua với SparkConf

val sparkConf = new SparkConf()

sparkConf.setMaster(“yarn")

sparkConf.setAppName(“test”)

val spark = SparkSession.builder().config(sparkConf).getOrCreate()

### 1.3 Chú ý

- ngoài master và appname còn các tham số khác có thể set được

- tuy nhiên trong thực tế thì chỉ cần set appname trong code là đủ

- có thể khởi tạo và dùng spark với SparkContext và SparkSession

tuy nhiên nên dùng SparkSession do có thể truy cập SparkContext bằng cách

val sparkContext = sparkSession.sparkContext

VD: submit.sh

MAIN\_CLASS=vn.vccorp.adtech.bigdata.Main

CONFIG\_PROPERTIES=config.properties

JARS=libs/mysql-connector-java-5.1.34.jar

JAR=attribution-1.0.0.jar

$SPARK\_HOME/bin/spark-submit \

--class $MAIN\_CLASS \

--master yarn \

--executor-cores 8 \

--executor-memory 2G \

--driver-memory 2G \

--num-executors 4 \

--files $CONFIG\_PROPERTIES \

--jars $JARS \

$JAR

- phía trên là câu lệnh (bash script) khi submit thực tế

- có thể thấy các thông số quan trọng (như master) đã được set ngay lúc submit

- do vậy trong code không cần phải set tham số này nữa

- tuy nhiên nếu set master là “local” trong code thì spark sẽ chạy local trên máy bạn

rất thuận tiện cho việc debug, testing

## 2. Đọc - Ghi file

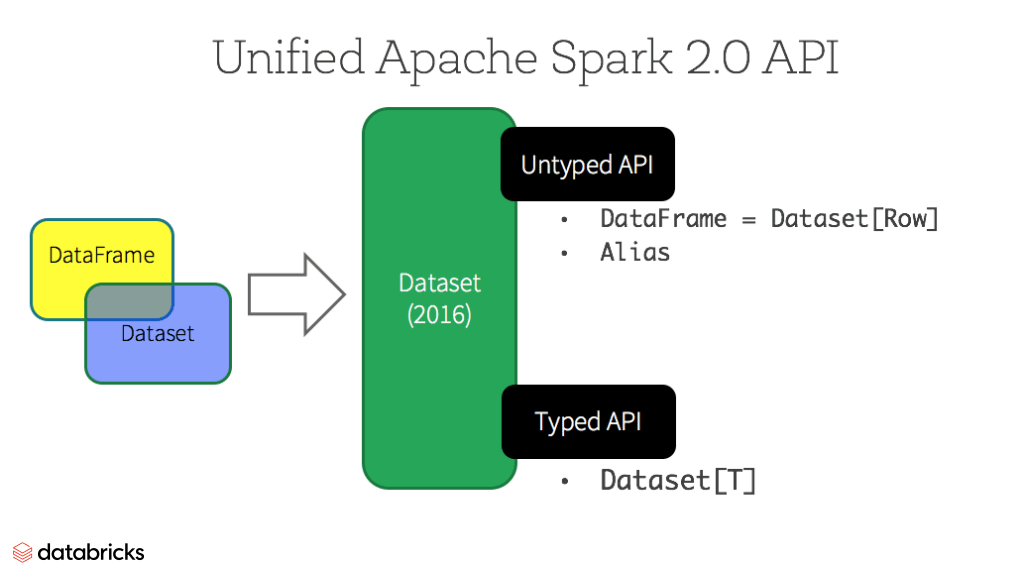
Ghi chú:

- phạm vi của docs này chỉ thực hiện với hadoop dùng hdfs

- spark: SparkSession

- sparkContext: SparkContext

- trong tài liệu đề cập tới dataset, dataframe, ngoài ra còn có RDD (tự tìm hiểu)



### 2.1 chú ý đường dẫn

- đường dẫn tương đối (relative)

- với master là local thì mọi đường dẫn sẽ hoạt động như máy ở local

- còn thực tế khi submit (ở server) thì mặc định bạn sẽ ở thư mục

/user/<username>

với username là username của bạn khi submit

VD:

spark.sparkContext.textFile(“test.txt”)

// spark sẽ tìm ở trong thư mục hiện tại có file test.txt hay không

// với master là local thì sẽ là thư mục bạn chạy spark

// với master là yarn sẽ là thư mục /user/<username>

spark.sparkContext.textFile(“/home/tuana9a/test.txt”)

// spark tìm theo đường dẫn tuy nhiên sẽ từ root chứ không theo thư mục hiện tại

// cả master là local và master là yarn hoạt động giống nhau

- đường dẫn tuyệt đối (full, đầy đủ)

không bị ảnh hưởng bởi việc set master

VD:

spark.read.parquet(“hdfs://adt-dw-spark-hdfs-cloud-37-209:8020/Data/Logging/ad-cpa-bid/2021-06-22”)

//đọc data từ một server khác

spark.sparkContext.textFile(“file:///home/tuana9a/test.txt”)

//với đường dẫn trên thì spark sẽ không đọc data của hadoop mà đọc file từ file system

### 2.2 đọc ghi dạng text

val textFile = spark.sparkContext.textFile(“test.txt”)

val textFile = spark.sparkContext.textFile("file:///ads\_log/ad-kinghub-pv/2021-03-30" + "/\*.dat")

val textFile = spark.read.textFile(“file:///ads\_log/ad-kinghub-pv/2021-03-30/\*.dat”)

// trả về một collection chứa các phần tử là các dòng của file đọc được

// có thể hình dung là một list của các String

textFile.collect().foreach(println) // in tất cả các dòng của một file

// với master là local có thể dùng trực tiếp foreach sẽ print được

// với master là yarn thì cần collect do data dạng phân tán (nếu k collect thì bạn đoán xem 😃)

textFile.saveAsTextFile(“một đường dẫn nào đó”) // lưu text file ra nơi nào đó

### 2.3 đọc ghi dạng parquet

- parquet là một định dạng file giống như excel, csv

- có cột, tên cột là tên thuộc tính của case class tương ứng

- mỗi hàng là một bản ghi

- parquet cho phép kiểu dữ liệu một cột có thể không phải nguyên thủy

tức một class kết tập trong một class (xem [Bonus](#_22qp55vfjcl1))

- chủ yếu dùng dataframe và dataset trong quá trình xử lí

Chú ý:

import spark.implicits.\_

việc import này dùng lúc as để biến dataframe thành dataset

import này chỉ dùng được khi khởi tạo spark: SparkSession (như ở trên đã nêu)

// dataframe

val cpa = spark.read.parquet("hdfs://10.5.37.209:8020/Data/Logging/ad-cpa-bid/2021-03-22")

cpa.printSchema() // xem thông tin các trường, kiểu dữ liệu của trường đó

cpa.first() // lấy thằng đầu tiên của data

cpa.take(n).show() // show n thằng

cpa.take(n).foreach(println) //lấy n thằng và print chúng nó

cpa.show(n, false) // show n thằng, truncate=false, không cắt xâu dài hơn 20 ký tự

cpa.collect.foreach(println) // lấy tất, xong in tất

cpa.count() // đếm xem có bao nhiêu record

// dataset

case class Cpa(guid: String,

dt: String,

clickId: String,

cpa\_value: String,

sourceType: String,

f12: String)

val cpa = spark.read.parquet("hdfs://10.5.37.209:8020/Data/Logging/ad-cpa-bid/2021-03-22")

.as[Cpa]

// với .as[Cpa] thì trả về Dataset[Cpa], thay vì Row giờ ta có thể lấy thuộc tính của data

// như .guid .dt thay vì phải nhớ thứ tự cột trong data gốc

cpa.select($”guid”, $”dt”) // tạo dataframe mới từ các cột đã chọn

cpa.map(x => (utils.StringToLong(x.dt), x.d\_guid)) // trả về Dataset[Tuple2]

cpa.select($”guid”.as(“guid\_new”), $”dt”.as(“dt\_new”)) // trả về Dataframe và tên cột thay đổi

cpa.withColumnRenamed(“guid”,”guid\_new”) // trả về Dataframe với tên cột thay đổi

// lấy các record có guid = 696969 và dt (datetime) sau ngày 2021-06-22

cpa.filter(“guid = '696969' and dt > ‘2021-06-22’”) // lọc data với điều kiện

cpa.filter(x => x.guid == “696969” && x.dt > “2021-06-22”) // tương đương

// hàm count lấy được từ org.apache.spark.sql.functions.\_

cpa.groupBy(“guid”).agg(count(“dt”)) // groupBy guid và đếm

cpa.agg(count(“dt“)) // đếm luôn không cần group

cpa.select(count(“dt”)) // tương đương ở trên

// ngoài hàm count còn rất nhiều hàm có sẵn khác

// sum, countDistinct, max, min, ...

cpa.orderBy(“guid”) // sắp xếp theo cột guid, mặc định là tăng dần

cpa.orderBy(desc("guid")) // sắp xếp theo cột guid, giảm dần, hàm desc lấy giống vs hàm count

cpa.distinct() // loại bỏ các phần tử trùng lặp

cpa.dropDuplicates($"guid",$"dt") // loại bỏ trùng lặp theo cột

cpa.unionAll(cpa\_1) // hợp của 2 tập hợp, có trùng lặp

cpa.union(cpa\_1) // tương đương

cpa.intersect(cpa\_1) // giao của 2 tập hợp

cpa.join(cpa\_1, "guid") // join bằng cột guid mặc định inner join

cpa.join(cpa\_1, Seq("guid", "dt"), "inner") // join bằng nhiều cột loại là inner

// ngoài inner có cross, outer, full, fullouter, full\_outer, left, leftouter, left\_outer, right, rightouter, right\_outer, semi,

// leftsemi, left\_semi, anti, leftanti, left\_anti.

cpa.write.mode(SaveMode.Overwrite).parquet("một đường dẫn nào đó") // lưu data

// nếu k có mode mà tồn tại data sẽ báo lỗi

// ngoài Overwrite còn Append

## 3. Tương tác với datasource

### 

### 3.1 Phoenix

dataframe.write

.format("org.apache.phoenix.spark")

.mode(SaveMode.Overwrite)

.option("table", table)

.option("zkUrl", server)

.save()

### 3.2 Cassandra

dataframe.write

.format("org.apache.spark.sql.cassandra")

.mode(org.apache.spark.sql.SaveMode.Append)

.option("spark.cassandra.connection.host", cassHost)

.option("keyspace", ks)

.option("table", table)

.save()

### 3.3 Elasticsearch

def saveToEs(spark: SparkSession, df: DataFrame, index: String, \_type: String): Unit = {

val resource = index + "/" + \_type

var conf: Map[String, String] = Map[String, String]()

conf += (("es.resource", resource))

conf += (("es.nodes", ConfigInfo.ES\_SEED\_HOST))

conf += (("es.port", ConfigInfo.ES\_PORT))

conf += (("es.mapping.id", "id"))

conf += (("es.write.operation", "upsert"))

conf += (("es.net.http.auth.user", "elastic"))

conf += (("es.net.http.auth.pass", "admin@123"))

conf += (("es.mapping.exclude", "id"))

df.saveToEs(conf)

}

### 3.4 Aerospike

def push(data: List[AeroData], seeds: String, timeout: Int): Unit = {

if (data == null || data.isEmpty) return

val client = AeroFactory.connect(seeds)

for (aeroData <- data) {

try {

val expire = aeroData.expire

val writePolicy = new WritePolicy

writePolicy.setTimeout(timeout)

writePolicy.expiration = expire

writePolicy.sendKey = true

client.put(writePolicy, aeroData.key, aeroData.putOpBins:\_\*)

} catch {

case e: AerospikeException =>

logger.error("error push data into aerospike {}", e)

}

}

client.close()

}

## 4. Nâng cao

### 4.1 UDFs

- <https://spark.apache.org/docs/latest/sql-ref-functions-udf-scalar.html>

- viết tắt của User Defined Functions

- là những hàm mình có thể custom dùng như các hàm count, max, … ở trên

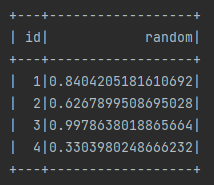
val temp1 = Seq(1, 2, 3, 4).toDF("id")

val temp2 = Seq(1, 3, 5, 6).toDF("id")

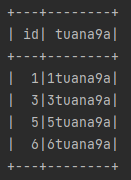
val udf1 = udf(() => Math.random())

val udf2 = udf((i: Int) => i.toString + "tuana9a")

temp1.select($"id", udf1().as("random")).show()



temp2.select($"id", udf2($"id").as("tuana9a")).show()



### 4.2 UDAFs

- <https://spark.apache.org/docs/latest/sql-ref-functions-udf-aggregate.html>

- viết tắt của User Defined Aggregate Functions

- xử lí với tập data, giống map reduce

### 4.4 Reshape data with pivot

- <https://databricks.com/blog/2016/02/09/reshaping-data-with-pivot-in-apache-spark.html>

- đại số, xoay cột, xoay hàng, múa quạt

## 5. Bonus

### 5.1 data dạng kết tập

case class Person(name: String, age: Int)

case class Couple(husband: Person, waifu: Person)

case class Family(couple: Couple, childs: List[Person])

val person1 = Person("tuana9a", 21)

val person2 = Person("gemdino", 21)

val person3 = Person("long", 21)

val person4 = Person("ngan", 19)

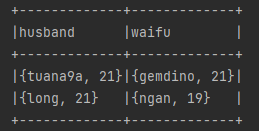
val person5 = Person("tuan anh", 1)

val couple1 = Couple(person1, person2)

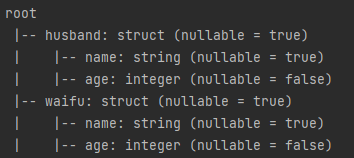
val couple2 = Couple(person3, person4)

val dataset1 = Seq(couple1, couple2).toDS()

dataset1.show()



dataset1.printSchema()

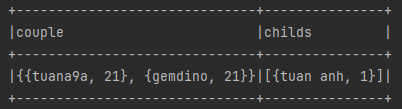


// parquet hơn excel, csv ở khoản này :V

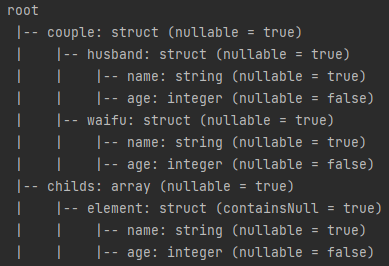
val family1 = Family(couple1, Seq(person5).toList)

val dataset2 = Seq(family1).toDS()

dataset2.show()



dataset2.printSchema()



// chơi cả data dạng list