TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM

**TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

**KHOA CÔNG NGHỆ THÔNG TIN**



**BÀI TẬP XỬ LÝ DỮ LIỆU LỚN**

**TÌM HIỂU VỀ SPARK**

*Người hướng dẫn*: **Thầy LÊ ANH CƯỜNG**

*Người thực hiện*: **DƯƠNG CAO MINH NHẬT – 51704078**

Lớp **: 17050401**

Khoá  **: 21**

**THÀNH PHỐ HỒ CHÍ MINH, NĂM 2020**

TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM

**TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

**KHOA CÔNG NGHỆ THÔNG TIN**



**BÀI TẬP XỬ LÝ DỮ LIỆU LỚN**

**TÌM HIỂU VỀ SPARK**

*Người hướng dẫn*: **Thầy LÊ ANH CƯỜNG**

*Người thực hiện*: **DƯƠNG CAO MINH NHẬT – 51704078**

Lớp **: 17050401**

Khoá  **: 21**

**THÀNH PHỐ HỒ CHÍ MINH, NĂM 2020**

MỤC LỤC

[LỜI CẢM ƠN i](#_Toc57750642)

[PHẦN XÁC NHẬN VÀ ĐÁNH GIÁ CỦA GIẢNG VIÊN iii](#_Toc57750643)

[TÓM TẮT iv](#_Toc57750644)

[MỤC LỤC 1](#_Toc57750645)

[DANH MỤC CÁC BẢNG BIỂU, HÌNH VẼ, ĐỒ THỊ 4](#_Toc57750646)

[CHƯƠNG 1 – SPARK PROPERTIES 5](#_Toc57750647)

[1.1 Khái niệm 5](#_Toc57750648)

[1.2 Dynamically Loading Spark Properties 5](#_Toc57750648)

[CHƯƠNG 2 –SPARK RDD 9](#_Toc57750655)

[2.1 Apache Spark RDD 5](#_Toc57750656)

[2.1.1 Thực thi trên Map Redure 6](#_Toc57750661)

[2.1.2 Thực thi trên Spark RDD 7](#_Toc57750661)

[2.2 Các loại RDD 8](#_Toc57750657)

[2.3 Các Transformation và Action với RDD 9](#_Toc57750658)

[2.4 Một số kỹ thuật với RDD 11](#_Toc57750659)

[CHƯƠNG 3 – SPARK DATAFRAME 14](#_Toc57750663)

[3.1 Khái niệm 15](#_Toc57750664)

[3.2 Các tính năng của DataFrame 16](#_Toc57750665)

[3.3 SQLCONTEXT 16](#_Toc57750666)

[3.3.1 Tài liệu tham khảo và cách trích dẫn 14](#_Toc57750667)

[3.3.2 Qui định của Khoa Công nghệ thông tin 15](#_Toc57750668)

[3.3 SQLCONTEXT 15](#_Toc57750666)

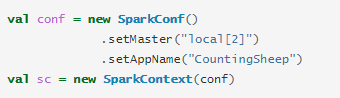
[3.3 Hoạt động của DataFrame 17](#_Toc57750666)

[Tài liệu tham khảo 19](#_Toc57750669)

CHƯƠNG 1 – SPARK PROERTIES

* 1. Khái niệm

Spark Property kiểm soát hầu hết các cài đặt ứng dụng và được cấu hình riêng cho từng ứng dụng. Các thuộc tính này có thể được đặt trực tiếp trên SparkConf được chuyển tới SparkContext của bạn. SparkConf cho phép bạn định cấu hình một số thuộc tính phổ biến (ví dụ: URL chính và tên ứng dụng), cũng như các cặp khóa-giá trị tùy ý thông qua phương thức set (). Ví dụ: chúng ta có thể khởi tạo một ứng dụng với hai luồng như sau:



Lưu ý rằng chúng ta có thể có nhiều hơn 1 luồng ở chế độ cục bộ(local) và trong những trường hợp như Spark Streaming, chúng tôi thực sự có thể yêu cầu nhiều hơn 1 luồng để ngăn chặn bất kỳ loại vấn đề starvation nào.

Các thuộc tính chỉ định một khoảng thời gian nên được cấu hình với một đơn vị thời gian. Định dạng sau được chấp nhận:

* 25ms (milliseconds)
* 5s (seconds)
* 10m or 10min (minutes)
* 3h (hours)
* 5d (days)
* 1y (years)

Thuộc tính chỉ định kích thước byte phải được cấu hình với đơn vị kích thước. Định dạng sau được chấp nhận:

* 1b (bytes)
* 1k or 1kb (kibibytes = 1024 bytes)
* 1m or 1mb (mebibytes = 1024 kibibytes)
* 1g or 1gb (gibibytes = 1024 mebibytes)
* 1t or 1tb (tebibytes = 1024 gibibytes)
* 1p or 1pb (pebibytes = 1024 tebibytes)
  1. Dynamically Loading Spark Properties

Trong một số trường hợp, bạn có thể muốn tránh mã hóa cứng các cấu hình nhất định trong SparkConf. Ví dụ: nếu bạn muốn chạy cùng một ứng dụng với các bản chính khác nhau hoặc số lượng bộ nhớ khác nhau. Spark cho phép bạn chỉ cần tạo một conf trống:

**val** sc **=** **new** **SparkContext**(**new** **SparkConf**())

Sau đó, bạn có thể cung cấp các giá trị cấu hình trong thời gian chạy:

./bin/spark-submit **--name** "My app" **--master** local[4] **--conf** spark.eventLog.enabled=false

**--conf** "spark.executor.extraJavaOptions=-XX:+PrintGCDetails -XX:+PrintGCTimeStamps" myApp.jar

Spark shell và công cụ spark-submit hỗ trợ hai cách để tải cấu hình động. Đầu tiên là các tùy chọn dòng lệnh, chẳng hạn như --master, như hình trên. spark-submit có thể chấp nhận bất kỳ thuộc tính Spark nào sử dụng cờ --conf / -c, nhưng sử dụng cờ đặc biệt cho các thuộc tính đóng một vai trò trong việc khởi chạy ứng dụng Spark. Chạy ./bin/spark-submit --help sẽ hiển thị toàn bộ danh sách các tùy chọn này.

Bin / spark-submit cũng sẽ đọc các tùy chọn cấu hình từ conf / spark-defaults.conf, trong đó mỗi dòng bao gồm một khóa và một giá trị được phân tách bằng khoảng trắng. Ví dụ:

spark.master spark://5.6.7.8:7077

spark.executor.memory 4g

spark.eventLog.enabled true

spark.serializer org.apache.spark.serializer.KryoSerializer

Mọi giá trị được chỉ định dưới dạng cờ hoặc trong tệp thuộc tính sẽ được chuyển đến ứng dụng và được hợp nhất với những giá trị được chỉ định thông qua SparkConf. Các thuộc tính được đặt trực tiếp trên SparkConf được ưu tiên cao nhất, sau đó các cờ được chuyển đến spark-submit hoặc spark-shell, sau đó là các tùy chọn trong tệp spark-defaults.conf.

Các thuộc tính của Spark chủ yếu có thể được chia thành hai loại: một là liên quan đến triển khai, như “spark.driver.memory”, “spark.executor.instances”, loại thuộc tính này có thể không bị ảnh hưởng khi thiết lập chương trình thông qua SparkConf trong thời gian chạy, hoặc hành vi tùy thuộc vào trình quản lý cụm và chế độ triển khai bạn chọn, vì vậy bạn nên đặt thông qua tệp cấu hình hoặc tùy chọn dòng lệnh spark-submit; một loại khác chủ yếu liên quan đến kiểm soát thời gian chạy Spark, như “spark.task.maxFailures”, loại thuộc tính này có thể được đặt theo một trong hai cách nào cũng được.

CHƯƠNG 2 – SPARK RDD

2.1 Apache Spark RDD

RDD(Resilient Distributed Dataset): là một cấu trúc dữ liệu cơ bản của Spark. Nó là một tập hợp bất biến phân tán của một đối tượng. Mỗi dataset trong RDD được chia ra thành nhiều phần vùng logical. Có thể được tính toán trên các node khác nhau của một cụm máy chủ (cluster).

RDDs có thể chứa bất kỳ kiểu dữ liệu nào của Python, Java, hoặc đối tượng Scala, bao gồm các kiểu dữ liệu do người dùng định nghĩa. Thông thường, RDD chỉ cho phép đọc, phân mục tập hợp của các bản ghi. RDDs có thể được tạo ra qua điều khiển xác định trên dữ liệu trong bộ nhớ hoặc RDDs, RDD là một tập hợp có khả năng chịu lỗi mỗi thành phần có thể được tính toán song song.

Có 2 cách để tạo RDD:

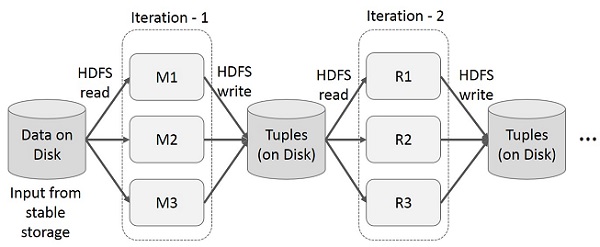
* + Tạo từ một tập hợp dữ liệu có sẵn trong ngôn ngữ sử dụng như Java, Python, Scala.
  + Lấy từ dataset hệ thống lưu trữ bên ngoài như HDFS, Hbase hoặc các cơ sở dữ liệu quan hệ.

2.1.1 Thực thi trên Map Redure

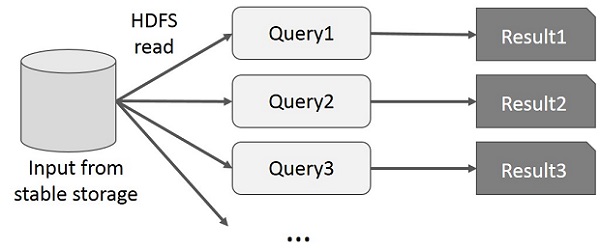
MapReduce được áp dụng rộng rãi để xử lý và tạo các bộ dữ liệu lớn với thuật toán xử lý phân tán song song trên một cụm. Nó cho phép người dùng viết các tính toán song song, sử dụng một tập hợp các toán tử cấp cao, mà không phải lo lắng về xử lý/phân phối công việc và khả năng chịu lỗi.

Tuy nhiên, trong hầu hết các framework hiện tại, cách duy nhất để sử dụng lại dữ liệu giữa các tính toán (Ví dụ: giữa hai công việc MapReduce) là ghi nó vào storage (Ví dụ: HDFS). Mặc dù framework này cung cấp nhiều hàm thư viện để truy cập vào tài nguyên tính toán của cụm Cluster, điều đó vẫn là chưa đủ.

Cả hai ứng dụng Lặp (Iterative) và Tương tác (Interactive) đều yêu cầu chia sẻ truy cập và xử lý dữ liệu nhanh hơn trên các công việc song song. Chia sẻ dữ liệu chậm trong MapReduce do sao chép tuần tự và tốc độ I/O của ổ đĩa. Về hệ thống lưu trữ, hầu hết các ứng dụng Hadoop, cần dành hơn 90% thời gian để thực hiện các thao tác đọc-ghi HDFS.



hình 2.1: Iterative Operation trên MapReduce



hình 2.2: Interactive Operations trên MapReduce

2.1.2 Thực thi trên Spark RDD

Để khắc phục được vấn đề về MapRedure, các nhà nghiên cứu đã phát triển một framework chuyên biệt gọi là Apache Spark. Ý tưởng chính của Spark là Resilient Distributed Datasets (RDD); nó hỗ trợ tính toán xử lý trong bộ nhớ. Điều này có nghĩa, nó lưu trữ trạng thái của bộ nhớ dưới dạng một đối tượng trên các công việc và đối tượng có thể chia sẻ giữa các công việc đó. Việc xử lý dữ liệu trong bộ nhớ nhanh hơn 10 đến 100 lần so với network và disk.

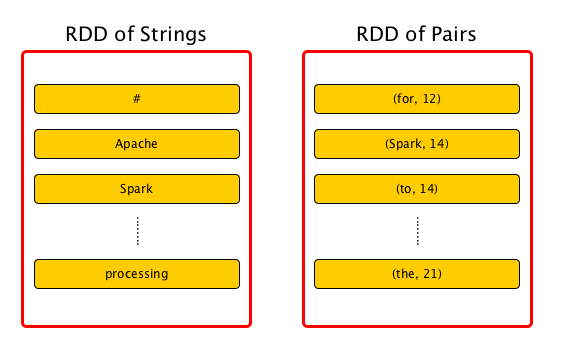


hình 2.3: Iterative Operation trên Spark RDD



hình 2.4: Interactive Operations trên Spark RDD

2.2 CÁC LOẠI RDD



hình 2.2: Các loại RDD

Các RDD biểu diễn một tập hợp cố định, đã được phân vùng các record để có thể xử lý song song.

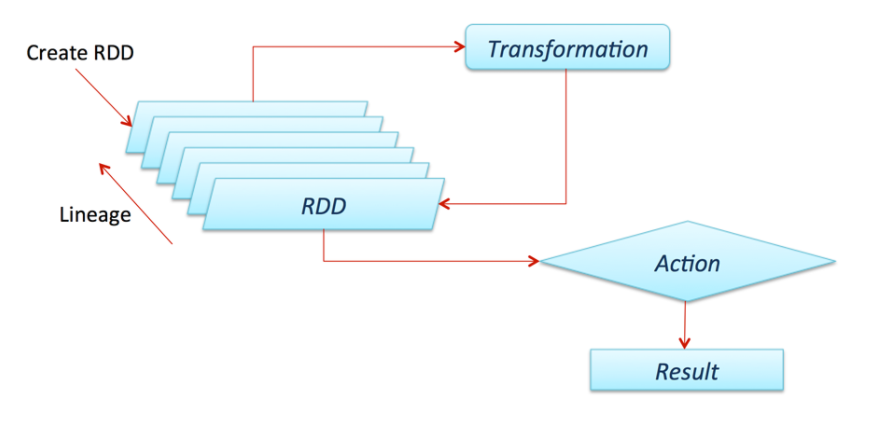
Các record trong RDD có thể là đối tượng Java, Scale hay Python tùy lập trình viên chọn. Không giống như DataFrame, mỗi record của DataFrame phải là một dòng có cấu trúc chứa các field đã được định nghĩa sẵn. RDD đã từng là API chính được sử dụng trong series Spark 1.x và vẫn có thể sử dụng trong version 2.X nhưng không còn được dùng thường xuyên nữa.

RDD API có thể được sử dụng trong Python, Scala hay Java:

* Scala và Java: Perfomance tương đương trên hầu hết mọi phần. (Chi phí lớn nhất là khi xử lý các raw object)
* Python: Mất một lượng performance, chủ yếu là cho việc serialization giữa tiến trình Python và JVM

2.3 CÁC TRANSFORMATION VÀ ACTION VỚI RDD

RDD cung cấp các transformation và action hoạt động giống như DataFrame lẫn DataSets. Transformation xử lý các thao tác lazily và Action xử lý thao tác cần xử lý tức thời.



Một số Transformation:

Nhiều phiên bản transformation của RDD có thể hoạt động trên các Structured API, transformation xử lý lazily, tức là chỉ giúp dựng execution plans, dữ liệu chỉ được truy xuất thực sự khi thực hiện action

* distinct: loại bỏ trùng lắp trong RDD
* filter: tương đương với việc sử dụng where trong SQL – tìm các record trong RDD xem những phần tử nào thỏa điều kiện. Có thể cung cấp một hàm phức tạp sử dụng để filter các record cần thiết – Như trong Python, ta có thể sử dụng hàm lambda để truyền vào filter
* map: thực hiện một công việc nào đó trên toàn bộ RDD. Trong Python sử dụng lambda với từng phần tử để truyền vào map
* flatMap: cung cấp một hàm đơn giản hơn hàm map. Yêu cầu output của map phải là một structure có thể lặp và mở rộng được.
* sortBy: mô tả một hàm để trích xuất dữ liệu từ các object của RDD và thực hiện sort được từ đó.
* randomSplit: nhận một mảng trọng số và tạo một random seed, tách các RDD thành một mảng các RDD có số lượng chia theo trọng số.

Một số Action:

Action thực thi ngay các transformation đã được thiết lập để thu thập dữ liệu về driver để xử lý hoặc ghi dữ liệu xuống các công cụ lưu trữ.

* reduce: thực hiện hàm reduce trên RDD để thu về 1 giá trị duy nhất
* count: đếm số dòng trong RDD
* countApprox: phiên bản đếm xấp xỉ của count, nhưng phải cung cấp timeout vì có thể không nhận được kết quả.
* countByValue: đếm số giá trị của RDD chỉ sử dụng nếu map kết quả nhỏ vì tất cả dữ liệu sẽ được load lên memory của driver để tính toán chỉ nên sử dụng trong tình huống số dòng nhỏ và số lượng item khác nhau cũng nhỏ.
* countApproxDistinct: đếm xấp xỉ các giá trị khác nhau
* countByValueApprox: đếm xấp xỉ các giá trị
* first: lấy giá trị đầu tiên của dataset
* max và min: lần lượt lấy giá trị lớn nhất và nhỏ nhất của dataset
* take và các method tương tự: lấy một lượng giá trị từ trong RDD. take sẽ trước hết scan qua một partition và sử dụng kết quả để dự đoán số lượng partition cần phải lấy thêm để thỏa mãn số lượng lấy.
* top và takeOrdered: top sẽ hiệu quả hơn takeOrdered vì top lấy các giá trị đầu tiên được sắp xếp ngầm trong RDD.
* takeSamples: lấy một lượng giá trị ngẫu nhiên trong RDD

2.4 MỘT SỐ KỸ THUẬT ĐỐI VỚI RDD

Lưu trữ file:

* Thực hiện ghi vào các file plain-text
* Có thể sử dụng các codec nén từ thư viện của Hadoop
* Lưu trữ vào các database bên ngoài yêu cầu ta phải lặp qua tất cả partition của RDD – Công việc được thực hiện ngầm trong các high-level API
* sequenceFile là một flat file chứa các cặp key-value, thường được sử dụng làm định dạng input/output của MapReduce. Spark có thể ghi các sequenceFile bằng các ghi lại các cặp key-value
* Đồng thời, Spark cũng hỗ trợ ghi nhiều định dạng file khác nhau, cho phép define các class, định dạng output, config và compression scheme của Hadoop.

Caching: Tăng tốc xử lý bằng cache

* Caching với RDD, Dataset hay DataFrame có nguyên lý như nhau.
* Chúng ta có thể lựa chọn cache hay persist một RDD, và mặc định, chỉ xử lý dữ liệu trong bộ nhớ

- Checkpointing: Lưu trữ lại các bước xử lý để phục hồi

* Checkpointing lưu RDD vào đĩa cứng để các tiến trình khác để thể sử dụng lại RDD point này làm partition trung gian thay vì tính toán lại RDD từ các nguồn dữ liệu gốc
* Checkpointing cũng tương tự như cache, chỉ khác nhau là lưu trữ vào đĩa cứng và không dùng được trong API của DataFrame
* Cần sử dụng nhiều để tối ưu tính toán.

Để áp dụng bất kỳ thao tác toán tử nào trong PySpark, trước tiên chúng ta cần tạo một PySpark RDD. Khối mã sau có chi tiết về Lớp RDD của PySpark:

class pyspark.RDD (

jrdd,

ctx,

jrdd\_deserializer = AutoBatchedSerializer(PickleSerializer())

)

Hãy để chúng tôi xem cách chạy một vài toán tử cơ bản bằng PySpark. Đoạn mã sau trong tệp Python tạo ra các từ RDD, lưu trữ một tập hợp các từ được đề cập.

words = sc.parallelize (

["scala",

"java",

"hadoop",

"spark",

"akka",

"spark vs hadoop",

"pyspark",

"pyspark and spark"]

)

Bây giờ chúng ta sẽ chạy một vài toán tử trên các từ:

Count(): Số phần tử trong RDD được trả về.

----------------------------------------count.py---------------------------------------

from pyspark import SparkContext

sc = SparkContext("local", "count app")

words = sc.parallelize (

["scala",

"java",

"hadoop",

"spark",

"akka",

"spark vs hadoop",

"pyspark",

"pyspark and spark"]

)

counts = words.count()

print "Number of elements in RDD -> %i" % (counts)

----------------------------------------count.py---------------------------------------

Command - Lệnh cho count () là –

$SPARK\_HOME/bin/spark-submit count.py

Ouput: Đầu ra cho lệnh trên là –

Number of elements in RDD → 8

foreach(f): Chỉ trả về những phần tử đáp ứng điều kiện của hàm bên trong foreach. Trong ví dụ sau, chúng tôi gọi một hàm in trong foreach, hàm này in tất cả các phần tử trong RDD.

----------------------------------------foreach.py---------------------------------------

from pyspark import SparkContext

sc = SparkContext("local", "ForEach app")

words = sc.parallelize (

["scala",

"java",

"hadoop",

"spark",

"akka",

"spark vs hadoop",

"pyspark",

"pyspark and spark"]

)

def f(x): print(x)

fore = words.foreach(f)

----------------------------------------foreach.py--------------------------------------

Command - Lệnh cho count () là –

$SPARK\_HOME/bin/spark-submit foreach.py

Ouput: Đầu ra cho lệnh trên là –

scala

java

hadoop

spark

akka

spark vs hadoop

pyspark

pyspark and spark

Reduce(f): Sau khi thực hiện thao tác nhị phân giao hoán và kết hợp được chỉ định, phần tử trong RDD được trả về. Trong ví dụ sau, chúng tôi đang nhập gói thêm từ toán tử và áp dụng nó trên ‘num’ để thực hiện một thao tác thêm đơn giản.

--------------------------------------reduce.py---------------------------------------

from pyspark import SparkContext

from operator import add

sc = SparkContext("local", "Reduce app")

nums = sc.parallelize([1, 2, 3, 4, 5])

adding = nums.reduce(add)

print "Adding all the elements -> %i" % (adding)

----------------------------------------reduce.py---------------------------------------

Command - Lệnh cho count () là –

$SPARK\_HOME/bin/spark-submit reduce.py

Ouput: Đầu ra cho lệnh trên là –

Adding all the elements -> 15

CHƯƠNG 3 – SPARK DataFrame

3.1 Khái niệm

DataFrame là một tập hợp dữ liệu phân tán, được tổ chức thành các cột được đặt tên. Về mặt khái niệm, nó tương đương với các bảng quan hệ có kỹ thuật tối ưu hóa tốt.

Một DataFrame có thể được xây dựng từ một loạt các nguồn khác nhau như bảng Hive, tệp Dữ liệu có cấu trúc, cơ sở dữ liệu bên ngoài hoặc RDD hiện có. API này được thiết kế cho các ứng dụng Khoa học dữ liệu và Dữ liệu lớn hiện đại lấy cảm hứng từ DataFrame trong Lập trình R và Pandas trong Python.

3.2 Các tính năng của DataFrame

Dưới đây là một số tính năng đặc trưng của DataFrame:

* Khả năng xử lý dữ liệu có kích thước từ Kilobyte đến Petabyte trên một cụm nút đơn đến cụm lớn.
* Hỗ trợ các định dạng dữ liệu khác nhau (Avro, csv, tìm kiếm đàn hồi và Cassandra) và hệ thống lưu trữ (HDFS, bảng HIVE, mysql, v.v.).
* Tối ưu hóa hiện đại và tạo mã thông qua trình tối ưu hóa Spark SQL Catalyst (khung chuyển đổi cây).
* Có thể dễ dàng tích hợp với tất cả các công cụ và khuôn khổ Dữ liệu lớn thông qua Spark-Core.
* Cung cấp API cho Lập trình Python, Java, Scala và R.

3.3 SQLCONTEXT

SQLContext là một lớp và được sử dụng để khởi tạo các chức năng của Spark SQL. Đối tượng lớp SparkContext (sc) là bắt buộc để khởi tạo đối tượng lớp SQLContext. Lệnh sau được sử dụng để khởi tạo SparkContext thông qua spark-shell:

$ spark-shell

Theo mặc định, đối tượng SparkContext được khởi tạo với tên sc khi bắt đầu spark-shell. Sử dụng lệnh sau để tạo SQLContext:

scala> val sqlcontext = new org.apache.spark.sql.SQLContext(sc)

Chúng ta hãy xem xét một ví dụ về bản ghi nhân viên trong tệp JSON có tên là employee.json. Sử dụng các lệnh sau để tạo DataFrame (df) và đọc tài liệu JSON có tên là employee.json với nội dung sau.

employee.json - Đặt tệp này vào thư mục chứa con trỏ scala> hiện tại.

{

{"id" : "1201", "name" : "satish", "age" : "25"}

{"id" : "1202", "name" : "krishna", "age" : "28"}

{"id" : "1203", "name" : "amith", "age" : "39"}

{"id" : "1204", "name" : "javed", "age" : "23"}

{"id" : "1205", "name" : "prudvi", "age" : "23"}

}

3.4 Hoạt động DataFrame

DataFrame cung cấp một ngôn ngữ dành riêng cho miền để thao tác dữ liệu có cấu trúc. Ở đây, chúng tôi bao gồm một số ví dụ cơ bản về xử lý dữ liệu có cấu trúc bằng DataFrames. Hãy làm theo các bước được cung cấp bên dưới để thực hiện các hoạt động DataFrame

Đọc tài liệu JSON

Đầu tiên, chúng ta phải đọc tài liệu JSON. Dựa trên điều này, tạo một DataFrame có tên (dfs). Sử dụng lệnh sau để đọc tài liệu JSON có tên là employee.json. Dữ liệu được hiển thị dưới dạng bảng với các trường - id, tên và tuổi.

scala> val dfs = sqlContext.read.json("employee.json")

Output: Tên trường được lấy tự động từ employee.json.

dfs: org.apache.spark.sql.DataFrame = [age: string, id: string, name: string]

Hiển thị dữ liệu:

scala> dfs.show()

Output - Bạn có thể xem dữ liệu nhân viên trong một định dạng bảng.

<console>:22, took 0.052610 s

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

|age | id | name |

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

| 25 | 1201 | satish |

| 28 | 1202 | krishna|

| 39 | 1203 | amith |

| 23 | 1204 | javed |

| 23 | 1205 | prudvi |

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

Lọc độ tuổi:

Sử dụng lệnh sau để tìm nhân viên có tuổi lớn hơn 23 (tuổi> 23).

scala> dfs.filter(dfs("age") > 23).show()

Output:

<console>:22, took 0.078670 s

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

|age | id | name |

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

| 25 | 1201 | satish |

| 28 | 1202 | krishna|

| 39 | 1203 | amith |

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

**TÀI LIỆU THAM KHẢO**

**Tiếng Việt**

1. <https://laptrinh.vn/books/apache-spark/page/apache-spark-rdd>

**Tiếng Anh – Truy cập ngày 25/11/2020**

1. <https://spark.apache.org/docs/latest/configuration.html>
2. <https://www.tutorialspoint.com/apache_spark/apache_spark_core_programming.htm>
3. <https://www.tutorialspoint.com/spark_sql/spark_sql_dataframes.htm>