**Học viện Đào tạo Lập trình MCI** 

5/23/165 Thái Hà, Đống Đa, Hà Nội/ 13 Cao Thắng, Quận 3

## ĐỀ TEST CUỐI MÔN DATA LAKE DATA WAREHOUSE

### LÝ THUYẾT

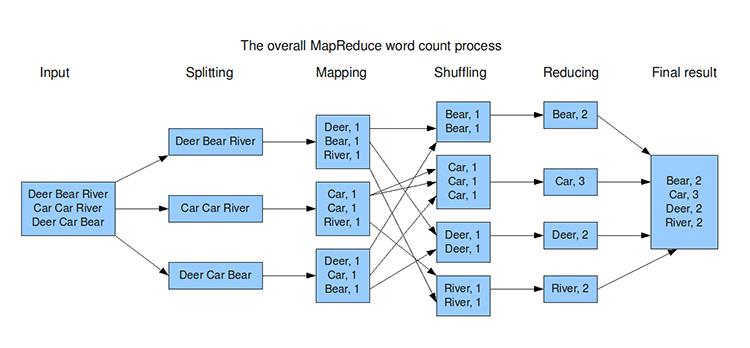
1.Hãy giải thích cơ chế MapReduce trong Hadoop?

Nguyên tắc hoạt động của Map Reduce

1. Data Input: Tiến hành chuẩn bị các dữ liệu đầu vào để cho Map() có thể xử lý
2. Splitting: Phân chia các dữ liệu cần xử lý thành nhiều phần nhỏ trước khi thực hiện.
3. Mapping: tiến hành nhận dữ liệu đầu vào và cấu trúc dữ liệu theo mô hình key-value
4. Shuffling: thu thập, tổng hợp và sắp xếp các cặp key-value trước khi chuyển qua bước reduce
5. Reducing: ghép các cặp key-value lại để quản lý dữ liệu dễ dàng hơn cũng như giảm dung lượng bộ nhớ cần cho storage

Các bước từ splitting đến reducing được chạy song song đồng thời trên các máy khác nhau. Nên MapReduce có thể xử lý khối lượng lớn dữ liệu một cách nhanh chóng và cho ra kết quả trong thời gian ngắn

Hình dưới minh họa các bước của MapReduce khi thực hiện phép phân tích “đếm tần suất xuất hiện các chữ trong 1 chuỗi”.



2.Hãy cho biết cách tạo RDD trong PySpark?

Các bước tạo RDD bằng PySpark

**1/ Tạo SparkSession:** Spark Session là 1 entry point của mọi chương trình Spark. Tạo Spark Session là bước đầu tiên để thực thi tính toán phân tán.

1.1. Trước tiên cần tải về thư viên pyspark bằng câu lệnh:

*Pip install pypark*

1.2. Sau đó sử dụng pyspark để gọi hàm tạo SparkSession trên notebook bằng câu lệnh:

*from pyspark.sql import SparkSession*

1.3. Tạo Spark Session

**spark** = (SparkSession

.master

.builder

.appName("Example-3\_6")

.getOrCreate())

master: tên của master node

appName: tên của chương trình chạy RDD

getOrCreate(): gọi SparkSession vừa tạo (hoặc SparkSession hiện có) để chạy tính toán phân tán

**2/ Tạo RDD** bằng cách gán data vào Spark Session (**spark**) vừa kích hoạt ở trên. Có các cách cơ bản sau:

2.1. Sử dụng hàm ***sparkContext.parallelize*** để tạo RDD từ data có sẵn

#Create RDD from parallelize

data = [1,2,3,4,5,6,7,8,9,10,11,12]

rdd=**spark**.sparkContext.parallelize(data)

### Có thể tạo partition cho data bằng cách thêm số lượng partition trong parallelize. Ví dụ:

rdd=**spark**.sparkContext.parallelize(data,10)

Nếu không khai báo, số lượng paritition sẽ bằng với số core trong máy

### 2.2. Sử dụng hàm *sparkContext.textFile()* để tạo RDD cho các text file có sẵn

#Create RDD from external Data source

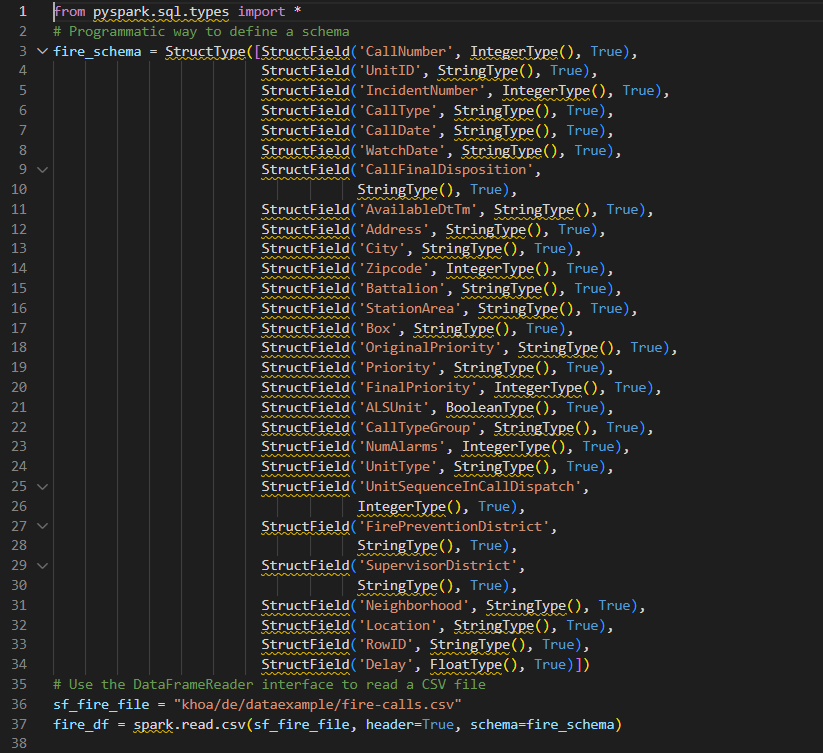
rdd2 = **spark**.sparkContext.textFile("/path/textFile.txt")

*/path/textFile.txt* là directory dẫn đến text file trong máy

Tương tự như 2.1, số lượng partition mặc định sẽ bằng với số core trong máy khi sử dụng hàm này

2.3 Sử dụng hàm ***read.csv*** để tạo RDD từ csv file

Trước tiên ta cần định nghĩa đường dẫn đến csv file (sf\_fire\_file) và schema (fire\_schema). Sau đó tạo RDD (fire\_df) theo ví dụ sau:



2.4. Tạo RDD từ Data Frame

Trước tiên ta cần định nghĩa schema và data. Ví dụ:

schema = StructType(list(StructField("Id", IntegerType(), True),

StructField("First", StringType(), True),

StructField("Last", StringType(), True),

StructField("Url", StringType(), True),

StructField("Published", StringType(), True),

StructField("Hits", IntegerType(), True),

StructField("Campaigns", ArrayType(StringType(), True), True)))

### data = [[1, "Khoa", "Dang", "https://tinyurl.1", "1/4/2016", 4535, ["twitter", "LinkedIn"]],

### [2, "Brooke", "Wenig", "https://tinyurl.2",

### "5/5/2018", 8908, ["twitter", "LinkedIn"]],

### [3, "Denny", "Lee", "https://tinyurl.3", "6/7/2019",

### 7659, ["web", "twitter", "FB", "LinkedIn"]],

### [4, "Tathagata", "Das", "https://tinyurl.4",

### "5/12/2018", 10568, ["twitter", "FB"]],

### [5, "Matei", "Zaharia", "https://tinyurl.5", "5/14/2014",

### 40578, ["web", "twitter", "FB", "LinkedIn"]],

### [6, "Reynold", "Xin", "https://tinyurl.6",

### "3/2/2015", 25568, ["twitter", "LinkedIn"]]

### ]

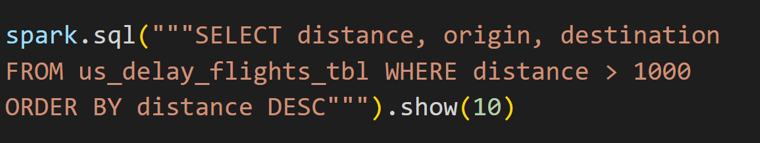
### Sau đó, sử dụng hàm *createDataFrame* để chuyển đổi data frame thành RDD

**spark**.createDataFrame(data, schema)

### 2.5. Tạo RDD từ câu SQL

Sử dụng hàm ***.sql*** để tạo RDD từ câu SQL

Ví dụ:



### II. THỰC HÀNH

1. Hãy viết một chương trình Python sử dụng thư viện Hadoop Streaming để đếm số lần xuất hiện của mỗi từ trong một chuỗi trên Hadoop HDFS.

|  |  |  |
| --- | --- | --- |
| Bước | Lệnh Python | Kết quả |
| 1.Tạo 1 file text, đặt tên file là *de\_test\_hadoop\_word\_count.txt* và bỏ vào trong thư mục  Lưu ý: bước 2,3,4,5 tạo file py thực thi lệnh mapper và reducer riêng rẽ có thể dc thay bằng bước 13 và 14 (dùng mrjob để chạy cả mapper và reducer) |  | File text có nội dung như sau:  *word count from Wikipedia the free encyclopedia*  *the word count is the number of words in a document or passage of text Word counting may be needed when a text*  *is required to stay within certain numbers of words This may particularly be the case in academia legal*  *proceedings journalism and advertising Word count is commonly used by translators to determine the price for*  *the translation job Word counts may also be used to calculate measures of readability and to measure typing*  *and reading speeds usually in words per minute When converting character counts to words a measure of five or*  *six characters to a word is generally used Contents Details and variations of definition Software* |
| 2.Tạo file py chạy bước mapping và đặt tên là mapper.py | #!/usr/bin/env python3  import sys  for line in sys.stdin:  words = line.split()  for word in words:  print('%s\t%s' % (word, 1)) |  |
| 3.Test file mapper.py xem kết quả có như mong muốn bằng lệnh cat | cat de\_test\_hadoop\_word\_count.txt | python3 mapper.py |  |
| 4.Tạo file py chạy bước reducer và đặt tên là reducer.py | #!/usr/bin/env python3  from operator import itemgetter  import sys  current\_word = None  current\_count = 0  word = None  for line in sys.stdin:  line = line.strip()  word, count = line.split('\t',1)  try:  count = int(count)  except ValueError:  continue  if current\_word == word:  current\_count += count  else:  if current\_word:  print('%s\t%s' % (current\_word,current\_count))  current\_count = count  current\_word = word  if current\_word == word:  print('%s\t%s' % (current\_word,current\_count)) |  |
| 5.Test file reducer xem kết quả có như mong muốn bằng lệnh cat.  Lưu ý phải map trước khi reduce nên chạy cả 2 file py mapper và reducer cùng lúc  Bước shuffling được thực hiện trước bước reduce và bằng câu lệnh sort-k1,1 (sắp xếp theo alphabet và nếu như trùng alphabet, word nào có số count nhiều nhất sẽ đứng trước) | cat de\_test\_hadoop\_word\_count.txt | python3 mapper.py | sort -k1,1 | python3 reducer.py |  |
| 6.Khởi động hadoop daemons | Start-dfs.sh  Start-yarn.sh  Hoặc  Start-all.sh |  |
| 7.Tạo directory để chứa file trên hadoop và đặt tên là de\_test | dfs dfs -mkdir /de\_test |  |
| 8.Kiểm tra xem thư mục đã có trên hadoop chưa | hdfs dfs -ls / |  |
| 9.Copy file từ local lên thư mục de\_test | hdfs dfs -copyFromLocal //home/quanho/hadoop/test\_hadoop/de\_test\_hadoop\_word\_count.txt |  |
| 10. Kiểm tra xem file để được copy lên de\_test chưa | hdfs dfs -ls /de\_test |  |
| 11.Đưa quyền execute cho 2 file mapper.py và reducer.py | chmod 777 mapper.py reducer.py |  |
| 12.Download hadoop streaming jar file và set up chạy hadoop streaming  Kết quả sẽ được lưu ở thư mục output trên hadoop | hadoop jar /home/quanho/hadoop/test\_hadoop/hadoop-streaming-2.7.3.jar \  > -input /de\_test/de\_test\_hadoop\_word\_count.txt \  > -output /de\_test/output \  > -mapper /home/quanho/hadoop/test\_hadoop/mapper.py \  > -reducer /home/quanho/hadoop/test\_hadoop/reducer.py |  |
| 13.Có thể gom 2 file mapper và reducer lại thành 1 file py để chạy cho gọn hơn bằng cách dùng mrjob  Import mrjob từ thư viện python và tạo file py với tên là word\_count.py | from mrjob.job import MRJob  class MRWordCount(MRJob):  def mapper(self, \_, line):  for word in line.split():  yield(word, 1)  def reducer(self,word, counts):  yield(word, sum(counts))  if \_\_name\_\_ == '\_\_main\_\_':  MRWordCount.run() |  |
| 14.Test thử chương trình mrjob | python3 word\_count.py de\_test\_hadoop\_word\_count.txt |  |

1. Hãy viết một ứng dụng Spark Streaming để đếm số lượng từ xuất hiện trong một chuỗi ký tự đầu vào real-time.

|  |  |  |
| --- | --- | --- |
| Bước | Lệnh Python | Kết quả |
| 1.Tạo chương trình đếm số lần xuất hiện của từ dựa theo dữ liệu streaming bằng pyspark  Đặt tên chương trình là *de\_test\_streaming.py* | from pyspark.sql import SparkSession  from pyspark.sql.functions import explode  from pyspark.sql.functions import split  spark = SparkSession \  .builder \  .appName("StructuredNetworkWordCount") \  .getOrCreate()  # Create DataFrame representing the stream of input lines from connection to localhost:9999  lines = spark \  .readStream \  .format("socket") \  .option("host", "localhost") \  .option("port", 9999) \  .load()  # Split the lines into words  words = lines.select(  explode(  split(lines.value, " ")  ).alias("word")  )  # Generate running word count  wordCounts = words.groupBy("word").count()  # Start running the query that prints the running counts to the console  query = wordCounts \  .writeStream \  .outputMode("complete") \  .format("console") \  .start()  query.awaitTermination() |  |
| 2. Mở spark | start-master.sh  start-worker.sh spark://quanho-virtual-machine:7077 |  |
| 3.Mở port 9999 và bắt đầu chạy streaming | ncat -lk 9999 |  |
| 4.Chạy chương trình  Do chưa có dữ liệu streaming đầu vào nên ban đầu sẽ trả về 0 giá trị | python3 de\_test\_streaming.py |  |
| 5.Nhập dữ liệu real time. Chương trình sẽ đếm số lần xuất hiện của mỗi từ tương ứng |  |  |

1. Hãy viết một ứng dụng PySpark để đọc dữ liệu từ tập tin CSV, xử lý và lưu trữ kết quả vào tập tin Parquet.

Yêu cầu:

-Đọc dữ liệu từ tập tin CSV vào DataFrame.

-Xóa các cột không cần thiết

-Chuyển đổi kiểu dữ liệu của cột

-Thêm dữ liệu vào Dataframe

-Lưu trữ kết quả vào tập tin Parquet.

|  |  |  |
| --- | --- | --- |
| Bước | Lệnh Python | Kết quả |
| 1.Chuyển file csv MCI cung cấp (*sendy\_logistics.csv*) vào thư mục dùng chạy Spark |  |  |
| 2.Tạo file py tên *de\_test\_read\_csv* , import PySpark và dùng PySpark để đọc file csv vào DataFrame  Sử dụng phương pháp InferSchema nên không cần khai báo Schema cho file csv | from pyspark.sql import SparkSession  spark = (SparkSession  .builder  .appName("spark\_read\_csv")  .getOrCreate()  )  csv\_file = "/home/quanho/SparkTest/sendy\_logistics.csv"  df = (spark.read.format("csv")  .option("inferSchema","true")  .option("header","true")  .load(csv\_file)  ) |  |
| 3.Review Schema của DataFrame vừa tạo bằng lệnh printSchema() | from pyspark.sql import SparkSession  spark = (SparkSession  .builder  .appName("spark\_read\_csv")  .getOrCreate()  )  csv\_file = "/home/quanho/SparkTest/sendy\_logistics.csv"  df = (spark.read.format("csv")  .option("inferSchema","true")  .option("header","true")  .load(csv\_file)  )  **df.printSchema()** |  |
| 4.Bỏ 1 số cột không cần thiết cho việc phân tích  Sử dụng nguyên tắc Immutable. 1 DataFrame mới sẽ dc tạo ra (df\_drop)  Review Schema sau khi bỏ các cột | from pyspark.sql import SparkSession  spark = (SparkSession  .builder  .appName("spark\_read\_csv")  .getOrCreate()  )  csv\_file = "/home/quanho/SparkTest/sendy\_logistics.csv"  df = (spark.read.format("csv")  .option("inferSchema","true")  .option("header","true")  .load(csv\_file)  )  **df\_drop = df.drop("Destination Lat","Destination Long","Pickup Lat","Pickup Long","Rider Id")**  **df\_drop.printSchema()** |  |
| 5.Data Type của cột Placement – Time *đ*ang là string (hình trên) chuyển sang data type là date  Sử dụng nguyên tắc Immutable. 1 DataFrame mới sẽ dc tạo ra (df\_changeType)    Review Schema sau khi đổi data type | from pyspark.sql import SparkSession  **from pyspark.sql.types import \***  spark = (SparkSession  .builder  .appName("spark\_read\_csv")  .getOrCreate()  )  csv\_file = "/home/quanho/SparkTest/sendy\_logistics.csv"  df = (spark.read.format("csv")  .option("inferSchema","true")  .option("header","true")  .load(csv\_file)  )  df\_drop = df.drop("Destination Lat","Destination Long","Pickup Lat","Pickup Long","Rider Id")  **df\_changeType = df\_drop.withColumn("Placement - Time",**  **df\_drop["Placement - Time"]**  **.cast(DateType()))**  **df\_changeType.printSchema()** |  |
| 6.Thêm một cột Is Working Day dùng để xác định ngày làm việc (working day) và ngày không làm việc (non-working day)  Trả về 1 nếu Confirmation - Weekday (Mo = 1) = 1,2,3,4,5 (working day)  Trả về 0 nếu Confirmation - Weekday (Mo = 1) = 6,7 (non-working day)  Sử dụng nguyên tắc Immutable. 1 DataFrame mới sẽ dc tạo ra (df\_addColumn)  Group lại theo cột Confirmation - Weekday (Mo = 1), group by và sort cột Confirmation - Weekday (Mo = 1) từ nhỏ đến lớn để review data cột Is Working Day mới thêm vào | from pyspark.sql import SparkSession  from pyspark.sql.types import \*  **from pyspark.sql.functions import when**  spark = (SparkSession  .builder  .appName("spark\_read\_csv")  .getOrCreate()  )  csv\_file = "/home/quanho/SparkTest/sendy\_logistics.csv"  df = (spark.read.format("csv")  .option("inferSchema","true")  .option("header","true")  .load(csv\_file)  )  df\_drop = df.drop("Destination Lat","Destination Long","Pickup Lat","Pickup Long","Rider Id")  df\_changeType = df\_drop.withColumn("Placement - Time",  df\_drop["Placement - Time"]  .cast(DateType()))  **df\_addColumn = df\_changeType.withColumn("Is Working Day", \**  **when((df\_changeType["Confirmation - Weekday (Mo = 1)"] < 6), 1) \**  **.when((df\_changeType["Confirmation - Weekday (Mo = 1)"] >= 6), 0) \**  **)**  **(df\_addColumn.select("Confirmation - Weekday (Mo = 1)", "Is Working Day")**  **.groupBy("Confirmation - Weekday (Mo = 1)").max("Is Working Day")**  **.orderBy("Confirmation - Weekday (Mo = 1)",ascending=True)**  **).show()** |  |
| 7.Lưu trữ kết quả vào tập tin parquet de\_test.parquet | from pyspark.sql import SparkSession  from pyspark.sql.types import \*  from pyspark.sql.functions import when  spark = (SparkSession  .builder  .appName("spark\_read\_csv")  .getOrCreate()  )  csv\_file = "/home/quanho/SparkTest/sendy\_logistics.csv"  df = (spark.read.format("csv")  .option("inferSchema","true")  .option("header","true")  .load(csv\_file)  )  df\_drop = df.drop("Destination Lat","Destination Long","Pickup Lat","Pickup Long","Rider Id")  df\_changeType = df\_drop.withColumn("Placement - Time",  df\_drop["Placement - Time"]  .cast(DateType()))  df\_addColumn = df\_changeType.withColumn("Is Working Day", \  when((df\_changeType["Confirmation - Weekday (Mo = 1)"] < 6), 1) \  .when((df\_changeType["Confirmation - Weekday (Mo = 1)"] >= 6), 0) \  )  **df\_addColumn.write.parquet("/home/quanho/SparkTest/de\_test.parquet")** |  |
| 8.Đọc thử file parquet | from pyspark.sql import SparkSession  from pyspark.sql.types import \*  from pyspark.sql.functions import when  spark = (SparkSession  .builder  .appName("spark\_read\_csv")  .getOrCreate()  )  csv\_file = "/home/quanho/SparkTest/sendy\_logistics.csv"  df = (spark.read.format("csv")  .option("inferSchema","true")  .option("header","true")  .load(csv\_file)  )  df\_drop = df.drop("Destination Lat","Destination Long","Pickup Lat","Pickup Long","Rider Id")  df\_changeType = df\_drop.withColumn("Placement - Time",  df\_drop["Placement - Time"]  .cast(DateType()))  df\_addColumn = df\_changeType.withColumn("Is Working Day", \  when((df\_changeType["Confirmation - Weekday (Mo = 1)"] < 6), 1) \  .when((df\_changeType["Confirmation - Weekday (Mo = 1)"] >= 6), 0) \  )  df\_addColumn.write.parquet("/home/quanho/SparkTest/de\_test.parquet")  **spark.read.parquet("/home/quanho/SparkTest/de\_test.parquet").show(5)** |  |