Tao gặp tình huống như này :   
172.30.2.98 : làm master node và worker 1   
172.30.2.58 : làm slave node và là worker 2   
172.30.2.90 : làm slave node và là worker 3   
  
Khi thực hiện lệnh : spark-submit --master spark://172.30.2.98:7077 /root/data\_sample/test\_pyspark.py   
  
để đọc file có đường dẫn nằm trên master node /root/data\_sample/citizens\_data\_small.csv   
Thì worker tại 172.30.2.58 báo lỗi như này :   
  
24/11/25 14:33:19 ERROR Executor: Exception in task 0.0 in stage 1.0 (TID 1) org.apache.spark.SparkFileNotFoundException: File file:/root/data\_sample/citizens\_data\_small.csv does not exist It is possible the underlying files have been updated. You can explicitly invalidate the cache in Spark by running 'REFRESH TABLE tableName' command in SQL or by recreating the Dataset/DataFrame involved. at org.apache.spark.sql.errors.QueryExecutionErrors$.readCurrentFileNotFoundError(QueryExecutionErrors.scala:781) at org.apache.spark.sql.execution.datasources.FileScanRDD$$anon$1.org$apache$spark$sql$execution$datasources$FileScanRDD$$anon$$readCurrentFile(FileScanRDD.scala:220) at org.apache.spark.sql.execution.datasources.FileScanRDD$$anon$1.nextIterator(FileScanRDD.scala:279) at org.apache.spark.sql.execution.datasources.FileScanRDD$$anon$1.hasNext(FileScanRDD.scala:129) at scala.collection.Iterator$$anon$10.hasNext(Iterator.scala:460) at scala.collection.Iterator$$anon$10.hasNext(Iterator.scala:460) at scala.collection.Iterator$$anon$10.hasNext(Iterator.scala:460) at scala.collection.Iterator$$anon$12.hasNext(Iterator.scala:513) at scala.collection.Iterator$$anon$12.hasNext(Iterator.scala:513) at scala.collection.Iterator$$anon$10.hasNext(Iterator.scala:460) at scala.collection.Iterator.foreach(Iterator.scala:943) at scala.collection.Iterator.foreach$(Iterator.scala:943) at scala.collection.AbstractIterator.foreach(Iterator.scala:1431) at scala.collection.TraversableOnce.foldLeft(TraversableOnce.scala:199) at scala.collection.TraversableOnce.foldLeft$(TraversableOnce.scala:192) at scala.collection.AbstractIterator.foldLeft(Iterator.scala:1431) at scala.collection.TraversableOnce.aggregate(TraversableOnce.scala:260) at scala.collection.TraversableOnce.aggregate$(TraversableOnce.scala:260) at scala.collection.AbstractIterator.aggregate(Iterator.scala:1431) at org.apache.spark.rdd.RDD.$anonfun$aggregate$2(RDD.scala:1226) at org.apache.spark.SparkContext.$anonfun$runJob$6(SparkContext.scala:2487) at org.apache.spark.scheduler.ResultTask.runTask(ResultTask.scala:93) at org.apache.spark.TaskContext.runTaskWithListeners(TaskContext.scala:166) at org.apache.spark.scheduler.Task.run(Task.scala:141) at org.apache.spark.executor.Executor$TaskRunner.$anonfun$run$4(Executor.scala:620) at org.apache.spark.util.SparkErrorUtils.tryWithSafeFinally(SparkErrorUtils.scala:64) at org.apache.spark.util.SparkErrorUtils.tryWithSafeFinally$(SparkErrorUtils.scala:61) at org.apache.spark.util.Utils$.tryWithSafeFinally(Utils.scala:94) at org.apache.spark.executor.Executor$TaskRunner.run(Executor.scala:623) at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1149) at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:624) at java.lang.Thread.run(Thread.java:750)

**ChatGPT said:**

ChatGPT

Lỗi bạn gặp phải xảy ra vì các worker (trên các node 172.30.2.58 và 172.30.2.90) không thể truy cập file citizens\_data\_small.csv, do file này chỉ tồn tại cục bộ trên **master node** (172.30.2.98). Trong Spark Standalone Cluster, khi một file được tham chiếu theo đường dẫn cục bộ (file://), các worker sẽ cố gắng truy cập nó từ chính hệ thống tệp cục bộ của chúng, dẫn đến lỗi.

**Cách khắc phục**

Để giải quyết vấn đề này, bạn có thể chọn một trong các giải pháp dưới đây:

**1. Sử dụng HDFS hoặc một hệ thống lưu trữ phân tán**

Đây là cách tốt nhất để làm việc với cụm Spark:

* **Cài đặt HDFS** trên cụm của bạn và lưu file citizens\_data\_small.csv vào HDFS.
* Truy cập file bằng cách sử dụng đường dẫn HDFS, ví dụ:

python

Copy code

data\_path = "hdfs://172.30.2.98:9000/data/citizens\_data\_small.csv"

df = spark.read.csv(data\_path, header=True)

Hoặc khi chạy spark-submit:

bash

Copy code

spark-submit --master spark://172.30.2.98:7077 /root/data\_sample/test\_pyspark.py

**2. Phân phối file bằng Spark's --files**

Sử dụng tùy chọn --files khi chạy spark-submit. Điều này sẽ phân phối file đến tất cả các node trong cụm:

bash

Copy code

spark-submit --master spark://172.30.2.98:7077 \

--files /root/data\_sample/citizens\_data\_small.csv \

/root/data\_sample/test\_pyspark.py

Trong code, bạn có thể tham chiếu file với đường dẫn Spark tự động:

python

Copy code

from pyspark.sql import SparkSession

# Lấy đường dẫn file từ Spark

file\_path = SparkFiles.get("citizens\_data\_small.csv")

# Đọc file

spark = SparkSession.builder.appName("Example").getOrCreate()

df = spark.read.csv(file\_path, header=True)

df.show()

**3. Đồng bộ file giữa các node**

Nếu file nhỏ và không muốn cài đặt HDFS, bạn có thể đồng bộ file thủ công lên tất cả các node:

**a. Đồng bộ file thủ công**

Sao chép file citizens\_data\_small.csv vào cùng đường dẫn trên tất cả các node:

bash

Copy code

scp /root/data\_sample/citizens\_data\_small.csv user@172.30.2.58:/root/data\_sample/

scp /root/data\_sample/citizens\_data\_small.csv user@172.30.2.90:/root/data\_sample/

**b. Đảm bảo file tồn tại ở tất cả các node**

Khi chạy Spark job, file sẽ được tìm thấy trên các worker vì chúng đã được đồng bộ.

**4. Chia sẻ file qua NFS (Network File System)**

Nếu không muốn đồng bộ thủ công, bạn có thể thiết lập một NFS mount điểm chung giữa các node:

1. **Cài đặt NFS Server trên Master**:
   * Chia sẻ thư mục /root/data\_sample/ qua NFS.
2. **Mount NFS trên các Slave**:
   * Mount thư mục /root/data\_sample/ trên 172.30.2.58 và 172.30.2.90.

Sau khi thiết lập, mọi node sẽ có thể truy cập file qua đường dẫn thống nhất /root/data\_sample/citizens\_data\_small.csv.

**5. Thay thế file:// bằng file:/// trên Master**

Nếu không muốn sử dụng các phương pháp trên và file chỉ cần nằm trên master node:

python

Copy code

file\_path = "file:///root/data\_sample/citizens\_data\_small.csv"

df = spark.read.csv(file\_path, header=True)

Nhưng điều này sẽ vẫn không hoạt động nếu Spark phân tán task đến các worker vì họ không thể truy cập file trên master.

**Khuyến nghị**

Sử dụng **HDFS hoặc Spark's --files** là cách tốt nhất để giải quyết vấn đề này. Nó không chỉ đảm bảo khả năng mở rộng mà còn phù hợp với các yêu cầu của Spark trong môi trường cụm.

Khi file CSV được mount trên tất cả các node trong cụm Spark, và bạn chạy đoạn mã của mình, các worker sẽ phối hợp để xử lý file theo cơ chế sau:

**1. Giai đoạn khởi động SparkSession**

* SparkSession được khởi tạo trên Master node (172.30.2.98).
* Master node chịu trách nhiệm lập kế hoạch (scheduling) công việc, bao gồm:
  + Phân chia các tác vụ (tasks).
  + Giao nhiệm vụ tới các worker node (172.30.2.58, 172.30.2.90).

**2. Đọc file CSV**

Lệnh spark.read.csv(file\_path, header=True, inferSchema=True):

* **Master node**:
  + Xác minh rằng đường dẫn file file\_path tồn tại.
  + Thông báo cho các worker node rằng file nằm trên hệ thống tệp chung và tất cả worker có thể đọc file này từ đường dẫn được cung cấp.
* **Worker nodes**:
  + Khi bắt đầu xử lý file CSV, Spark chia file thành các **partition** dựa trên kích thước file và logic phân vùng mặc định.
  + Mỗi worker đọc một phần (partition) của file CSV từ hệ thống tệp được mount.

**Điểm quan trọng:**

Nếu thư mục đã được mount chung, các worker sẽ truy cập file trực tiếp thông qua đường dẫn giống nhau (/root/data\_sample/citizens\_data\_small.csv) mà không cần phụ thuộc vào master node.

**3. Áp dụng bộ lọc (filter)**

Lệnh df.filter(col(column\_name).contains(search\_word)):

* Sau khi file được phân vùng và mỗi partition được xử lý độc lập:
  + Worker nodes thực hiện **bộ lọc** cục bộ trên partition mà chúng xử lý.
  + Ví dụ:
    - Worker 1 (172.30.2.98) xử lý partition 1.
    - Worker 2 (172.30.2.58) xử lý partition 2.
    - Worker 3 (172.30.2.90) xử lý partition 3.
* Worker chỉ xử lý dữ liệu nằm trong partition của chúng, sử dụng cột và từ khóa tìm kiếm (column\_name, search\_word).

**4. Đếm số dòng (count())**

Lệnh filtered\_df.count():

* Các worker:
  + Đếm số dòng trong partition của chúng khớp với điều kiện.
  + Kết quả từ mỗi worker được gửi về Master node.
* **Master node**:
  + Tổng hợp kết quả từ tất cả các worker để tính tổng số dòng khớp.

**5. Ghi log**

Các log liên quan đến quá trình thực thi được ghi vào log\_pyspark.log trên master node (vì logging được thực hiện từ ứng dụng driver chạy trên master).

**Tóm tắt cơ chế xử lý file**

1. **Phân vùng file**:
   * Spark chia file thành các partition, mỗi partition tương ứng với một phần dữ liệu.
   * Các partition được phân phối đều cho các worker.
2. **Đọc file từ hệ thống tệp chung**:
   * Worker nodes truy cập trực tiếp vào file thông qua đường dẫn mount /root/data\_sample/citizens\_data\_small.csv.
3. **Xử lý cục bộ tại worker**:
   * Worker thực hiện các phép tính (lọc, đếm) trên partition được giao.
4. **Tổng hợp kết quả**:
   * Master node thu thập và hợp nhất kết quả từ các worker.

**Ưu điểm khi sử dụng hệ thống tệp chung (mount)**

* Không cần đồng bộ file giữa các node.
* Hiệu suất cao vì các worker xử lý file trực tiếp, không qua trung gian.

**Hạn chế**

* Nếu file lớn, cần đảm bảo hệ thống tệp chung có hiệu suất đọc đủ cao.
* Tất cả các node phải có cùng cấu trúc đường dẫn tới file.