How can I use the map() transformation in PySpark effectively?

Authored by

stats writer

June 24, 2024

The map() transformation in PySpark is a powerful tool that allows for efficient manipulation and transformation of data in distributed systems. By applying a function to each element in a given dataset, map() enables users to easily and effectively perform operations such as filtering, aggregation, and data cleaning. With its ability to process data in parallel, map() greatly enhances the speed and scalability of data processing workflows. By understanding the syntax and proper usage of map() in PySpark, users can effectively harness its capabilities to improve the efficiency and accuracy of their data analysis tasks.

The `map()`in PySpark is a transformation function that is used to apply a function/lambda to each element of an RDD (Resilient Distributed Dataset) and return a new RDD consisting of the result.

When you have complex operations to apply on an RDD, the `map()` transformation is defacto function. You can use this for simple to complex operations like deriving a new element from exising data, or transforming the data, etc;

key Points:

Spark map() vs mapPartitions() Explained with Examples

First, let's create an RDD from the list.

```
# Imports
from pyspark.sql import SparkSession
spark = SparkSession.builder.master("local")
.appName(arabpsychology.com).getOrCreate()
```

data =

rdd=spark.sparkContext.parallelize(data)

map() Syntax

Syntax

```
# Syntax
map(f, preservesPartitioning=False)
```

PySpark RDD map() Example

Here's how the `map()` transformation works:

Function Application: You define a function that you want to apply to each element of the RDD.**Function Application to RDD**: You call the `map()` transformation on the RDD and pass the function as an argument to it.**Transformation Execution**: Spark applies the provided function to each element of the RDD in a distributed manner across the cluster.**New RDD Creation**: The `map()` transformation returns a new RDD containing the results of applying the function to each element of the original RDD.

```
# map() with rdd
rdd2=rdd.map(lambda x: (x,1))
for element in rdd2.collect():
print(element)
```

Here, We apply the `map()` transformation to each element `x` in the RDD `rdd`. The lambda function `(lambda x: (x, 1))` takes each element `x` of the RDD `rdd` and returns a tuple `(x, 1)`. So, for each element in `rdd`, the resulting RDD `rdd2` contains a tuple where the original element `x` is paired with the integer `1`.

```
('Project', 1)
('Gutenberg's', 1)
('Alice's', 1)
('Adventures', 1)
('in', 1)
('Wonderland', 1)
('Project', 1)
('Gutenberg's', 1)
('Adventures', 1)
('in', 1)
('Wonderland', 1)
('Project', 1)
('Gutenberg's', 1)
```

PySpark map() Example with DataFrame

PySpark DataFrame doesn't have `map()` transformation to apply the lambda function, when you

wanted to apply the custom transformation, you need to convert the DataFrame to RDD and apply the map() transformation. Let's use another dataset to explain this.

```
data =
```

columns =
df = spark.createDataFrame(data=data, schema = columns)
df.show()

Output:
#+---------+--------+------+------+
#|firstname|lastname|gender|salary|
#+---------+--------+------+------+
#| James| Smith| M| 30|
#| Anna| Rose| F| 41|
#| Robert|Williams| M| 62|
#+---------+--------+------+------+

Use map() transfromation on DataFrame.

```
# Refering columns by index.
rdd2=df.rdd.map(lambda x:
(x+","+x,x,x*2)
)
df2=rdd2.toDF( )
df2.show()
```

Output:
#+--------------+-----+----------+
#| name|gender|new_salary|
#+--------------+------+----------+
#| James,Smith| M| 60|
#| Anna,Rose| F| 82|
#|Robert,Williams| M| 124|
#+--------------+------+----------+

The above example refers to the columns by index. The below example uses column names.

```
# Referring Column Names
rdd2=df.rdd.map(lambda x:
(x+","+x,x,x*2)
)
```

Another alternative

```
# Referring Column Names
rdd2=df.rdd.map(lambda x:
(x.firstname+","+x.lastname,x.gender,x.salary*2)
)
```

Using custom function on map() transformation

You can also create a function and use this function on map() transformation

```
# By Calling function
def func1(x):
firstName=x.firstname
lastName=x.lastname
name=firstName+","+lastName
gender=x.gender.lower()
salary=x.salary*2
return (name,gender,salary)
```

# Apply the func1 function using lambda
rdd2 = df.rdd.map(lambda x: func1(x))

#or
# Apply the func1 function to each element of the RDD using map()
rdd2 = df.rdd.map(func1)

Complete PySpark map() example

Below is a complete example of PySpark map() transformation.

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()
```

```
data =

rdd=spark.sparkContext.parallelize(data)

rdd2=rdd.map(lambda x: (x,1))
for element in rdd2.collect():
print(element)

data =

columns =
df = spark.createDataFrame(data=data, schema = columns)
df.show()

rdd2=df.rdd.map(lambda x:
(x+","+x,x,x*2)
)
df2=rdd2.toDF( )
df2.show()

#Referring Column Names
rdd2=df.rdd.map(lambda x:
(x+","+x,x,x*2)
)

#Referring Column Names
rdd2=df.rdd.map(lambda x:
(x.firstname+","+x.lastname,x.gender,x.salary*2)
)

def func1(x):
firstName=x.firstname
lastName=x.lastname
name=firstName+","+lastName
gender=x.gender.lower()
salary=x.salary*2
return (name,gender,salary)

rdd2=df.rdd.map(lambda x: func1(x))
```

Frequently Asked Questions on map()

How does the `map()` transformation differ from other transformations, like `flatMap()` in PySpark?

The `map()` transformation applies a function on each element of the RDD independently, resulting in a new RDD with the same number of elements. Meanwhile, `flatMap`() can produce a variable number of output elements for each input element.

Can we apply Python lambda functions with the `map()` transformation in PySpark?

We can use Python lambda functions or regular functions with the `map()` transformation.
For example:
rdd = sc.parallelize()
even_square = rdd.map(lambda x: x**2)

How does the `map()` transformation handle null or missing values?

The `map()` transformation in PySpark processes each element independently, and by default, it does not handle the null or missing values. We need to handle these cases within the mapping function explicitly.

How to use the `map()` transformation with key-value pairs in PySpark?

For key-value pairs, we need to use the `map()` transformation with a function that operates on the values while preserving the keys.

Conclusion

In summary, you've learned how to use a `map()` transformation on every element within a PySpark RDD and have observed that it returns the same number of rows as the input RDD. This distinction is one of the differences between `flatMap()` transformation. Additionally, you've gained insight into leveraging `map()` on DataFrames by first converting them to RDDs.

Happy Learning !!

Related Articles