“What is the purpose of PySpark Accumulator and can you provide an example of how it is used?

Authored by

stats writer

June 24, 2024

PySpark Accumulator is a built-in feature in PySpark that allows for efficient and distributed shared variables to be used in parallel operations. Its purpose is to provide a way to aggregate values from multiple tasks and use them in a single driver program without having to manually handle the synchronization and communication between tasks. This avoids potential race conditions and improves the performance of data processing tasks.

An example of how PySpark Accumulator is used is in a word count program. The accumulator variable can be initialized to 0 and then incremented for each word encountered in a text file by parallel tasks. The final value of the accumulator variable will give the total count of words in the file, which can then be used in the driver program for further analysis or output. This eliminates the need for explicit communication between tasks and simplifies the code.

The PySpark Accumulator is a shared variable that is used with RDD and DataFrame to perform sum and counter operations similar to Map-reduce counters. These variables are shared by all executors to update and add information through aggregation or computative operations.

In this article, I've explained what is PySpark Accumulator, how to create, and using it on RDD and DataFrame with an example.

What is PySpark Accumulator?

Accumulators are write-only and initialize once variables where only tasks that are running on workers are allowed to update and updates from the workers get propagated automatically to the driver program. But, only the driver program is allowed to access the Accumulator variable using the **value** property.

How to create Accumulator variable in PySpark?

Using **accumulator**() from SparkContext class we can create an Accumulator in PySpark programming. Users can also create Accumulators for custom types using `AccumulatorParam` class of PySpark.

Some points to note..

We can create Accumulators in PySpark for primitive types **int** and **float**. Users can also create Accumulators for custom types using `AccumulatorParam` class of PySpark.

Creating Accumulator Variable

Below is an example of how to create an accumulator variable "**accum**" of type int and using it to sum all values in an RDD.

```
from pyspark.sql import SparkSession
spark=SparkSession.builder.appName("accumulator").getOrCreate()
```

accum=spark.sparkContext.accumulator(0)
rdd=spark.sparkContext.parallelize()
rdd.foreach(lambda x:accum.add(x))
print(accum.value) #Accessed by driver

Here, we have created an accumulator variable **accum** using **spark.sparkContext.accumulator(0)** with initial value 0. Later, we are iterating each element in an rdd using foreach() action and adding each element of rdd to accum variable. Finally, we are getting accumulator value using **accum.value** property.

Note that, In this example, `rdd.foreach()` is executed on workers and `accum.value` is called from PySpark driver program.

Let's see another example of an accumulator, this time will do with a function.

```
accuSum=spark.sparkContext.accumulator(0)
def countFun(x):
global accuSum
accuSum+=x
rdd.foreach(countFun)
print(accuSum.value)
```

We can also use accumulators to do a counters.

```
accumCount=spark.sparkContext.accumulator(0)
rdd2=spark.sparkContext.parallelize()
rdd2.foreach(lambda x:accumCount.add(1))
print(accumCount.value)
```

PySpark Accumulator Example

Below is a complete RDD example of using different accumulators that I was able to run on my environment.

```
import pyspark
```

```
from pyspark.sql import SparkSession
spark=SparkSession.builder.appName("accumulator").getOrCreate()
```

accum=spark.sparkContext.accumulator(0)
rdd=spark.sparkContext.parallelize()
rdd.foreach(lambda x:accum.add(x))
print(accum.value)

accuSum=spark.sparkContext.accumulator(0)
def countFun(x):
global accuSum
accuSum+=x
rdd.foreach(countFun)
print(accuSum.value)

accumCount=spark.sparkContext.accumulator(0)
rdd2=spark.sparkContext.parallelize()
rdd2.foreach(lambda x:accumCount.add(1))
print(accumCount.value)

Conclusion

In summary, PySpark Accumulators are shared variables that can be updated by executors and propagate back to driver program. These variables are used to add sum or counts and final results can be accessed only by driver program.

Related Articles

Reference