Intro To Spark Assignment 5

By Himani Anil Deshpande

Describe what are Accumulators and Broadcast Variables in Spark and when to use these shared variables?

Shared Variables:

When a function passed to a Spark operation (such as a map or reduction) to run on a remote cluster node, it doesnt work on the original copy of the variables but works with a separate copy of all the variables used in the function.
These copies of variables are available in all the remote nodes and while the operation runs on these nodes the updates are done on these copies of variables and not the original copy, but the results from these nodes doesnt return the updates done on these copies of variables to the driver program. Supporting common read / write shared variables between tasks is inefficient. But, Spark provides two restricted types of shared variables for two common usage patterns: broadcast variables and accumulators.

Broadcast Variables:

Spark uses efficient broadcast algorithms to distribute broadcast variables and keep the communication cost to the minimum. These variables are for keeping a read-only copy of the variables which are cached on each node rather than transferring the copy with the job/task. As they are read-only no updates need to be propagated. These are useful when we need to keep a copy of a large dataset on every node.
The easiest way to meet read-only requirements is to pass a reference to a primitive value or immutable object. In such cases, you can only change the value of the broadcast variable in the driver code. Spark actions are performed through a series of stages separated by a distributed "shuffle" operation.
Automatically, Spark sends the common data needed for each stage of the task. The data transferred in this way is cached in a serialized format and deserialized before each task is performed. In other words, explicitly creating broadcast variables only makes sense if the task requires the same data across multiple stages, or if it is important to cache the data in a deserialized format.

Broadcast variables are created from a variable v by calling SparkContext.broadcast(v). The broadcast variable is a wrapper around v, and its value can be accessed by calling the value method.

```
from pyspark.sql import SQLContext

sqlContext = SQLContext(sc)


country = {"IN":"India", "KR":"South Korea", "DK":"Denmark", "AR":"Argentina"}
broadcast_country = sc.broadcast(country)

data = [("Shaun","Verghese","IN"),
    ("Chandra","Shah","IN"),
    ("Paula","Williams","DK"),
    ("Maria","D'costa","AR"),
     ("Kim","Unn Park","KR")
  ]

columns = ["firstname","lastname","country"]
df = sqlContext.createDataFrame(data = data, schema = columns)
df.printSchema()
df.show(truncate=False)

def state_convert(code):
    return broadcast_country.value[code]

result = df.rdd.map(lambda x: (x[0],x[1],state_convert(x[2]))).toDF(columns)
result.show(truncate=False)
```

```
/usr/local/lib/python3.7/dist-packages/pyspark/sql/context.py:79: FutureWarning: Deprecated in 3.0.0. Use SparkSession.builder.getOrCreate() instead.
  FutureWarning
root
 |-- firstname: string (nullable = true)
 |-- lastname: string (nullable = true)
 |-- country: string (nullable = true)

+---------+--------+-------+
|firstname|lastname|country|
+---------+--------+-------+
|Shaun    |Verghese|IN     |
|Chandra  |Shah    |IN     |
|Paula    |Williams|DK     |
|Maria    |D'costa |AR     |
|Kim      |Unn Park|KR     |
+---------+--------+-------+

+---------+--------+-----------+
|firstname|lastname|country    |
+---------+--------+-----------+
|Shaun    |Verghese|India      |
|Chandra  |Shah    |India      |
|Paula    |Williams|Denmark    |
|Maria    |D'costa |Argentina  |
|Kim      |Unn Park|South Korea|
+---------+--------+-----------+
```

I the above example, I am creating a dataframe containing Person data and a braodcast variables containing the lookup for a few counties codes. Then I use the bradcast variable to convert these country codes(Alphabetical codes) to the Full country name.

Accumulators:

Accumulators provide a simple syntax for aggregating values from worker nodes back to the driver program. Accumulators are variables that are "added" only by associative law and commutative operations, so they can be efficiently supported in parallel.

These can be used to implement counters, like MapReduce counters or sums. Spark by default supports numeric accumulators with added support for new types.
we can create named or unnamed accumulators. A named accumulator appears in the web UI of the stage.

```
[8]
    num = sc.accumulator(0)
    def func_add(x):
       global num
       num+=x
      #  print(['*']*num)
    rdd = sc.parallelize([1,2,3,4,5])
    rdd.foreach(func_add)
    final = num.value
    print ("Accumulated value is ->", final)
```

```
Accumulated value is -> 15
```

In the above example the values of RDD are summed and stored in Accumulator

```
# from pyspark.sql import SparkSession
# import pyspark
from functools import partial

def remove_odd(item, accumulator):
    if item % 2 == 0:
        accumulator += 1
    return '0' in str(item)


accumulator = sc.accumulator(0)
count_filter = partial(remove_odd, accumulator=accumulator)

print(sc.range(0, 100).filter(count_filter).sum())

print('accum', accumulator)
```

```
450
accum 50
```

References:

https://spark.apache.org/docs/2.2.0/rdd-programming-guide.html#shared-variables

Canvas Modules