Setting UP PySpark enviroment in Colab

In [ ]:
```
!pip install pyspark py4j
```

Question 1 and Question 2

Spark uses shared variables for parallel processing. A copy of shared variable goes on each node of the cluster when the driver sends a task to the executor on the cluster, so that it can be used for performing tasks.

Two types of shared variables supported by Spark:

- Broadcast
- Accumulator

In [26]:
```
from pyspark.sql import SparkSession
from pyspark import SparkContext
```

Broadcast

Broadcast variables are read-only shared variables that are cached and available on all nodes in a cluster in-order to access or use by the tasks. Instead of sending this data along with every task, PySpark distributes broadcast variables to the workers using efficient broadcast algorithms to reduce communication costs. Broadcast variables are used in the same way for RDD and DataFrames and when we run these applications that have the broadcast variables defined and used, PySpark does the following:

- PySpark breaks the job into stages that have distributed shuffling and actions are executed with in the stage.
- Later Stages are also broken into tasks.
- Spark broadcasts the common data (reusable) needed by tasks within each stage.
- The broadcasted data is cache in serialized format and deserialized before executing each task.

The following example comes from the Broadcast Variable source from the Reference tab.

In [27]:
```
spark = SparkSession.builder.appName('Broadcast App').getOrCreate()
```

In [29]:
```
# This function takes the broadcast variable to look for its corresponding state
def state_convert(x):
    return broadcast_states.value[x] # x value used to access values in broadcast_states
```

In [33]:
```
# Broadcast variables are used to save the copy of data across all nodes
# Note that broadcast variables are not sent to executors with sc.broadcast(variable) call instead,
# they will be sent to executors when they are first used.

states = {'PA':'Panama', 'TLH':'Tallahassee', 'MIA':'Miami'} # dictionary

lst1 = [('Carolina','Licona','USA','TLH'),
        ('Vicente','De Leon','USA','MIA'),
        ('Max','Licona','PTY','PA'),
        ('Maria','Jones','USA','TLH')]

broadcast_states = spark.sparkContext.broadcast(states)
rdd = spark.sparkContext.parallelize(lst1) # using parallelize() function to create rdd from a list collection

result = rdd.map(lambda x: (x[0],x[1],x[2],state_convert(x[3]))).collect() # extracts elements from tuples and applies state_convert function
print('Broadcast Variable results: ', result)
```

```
Broadcast Variable results:  [('Carolina', 'Licona', 'USA', 'Tallahassee'), ('Vicente', 'De Leon', 'USA', 'Miami'), ('Max', 'Licona', 'PTY', 'Panama'), ('Maria', 'Jones', 'USA', 'Tallahassee')]
```

- PA -> Panama
- TLH -> Tallahassee
- MIA -> Miami

Accumulator

It's a shared variable concetp to aggregate information.

It’s a shared variable that is used with RDD and DataFrame (distributed data) to perform sum and counter operations like Map_reduce counters. These variables are shared by all executors to update and add information through aggregation or computative operations. They’re basically shared variables that can be updated by executors and propagate back to driver program for result collections. The following example comes from the “Accumulators” reference:

In [7]:
```
spark = SparkSession.builder.appName('Accumulator App').getOrCreate()
```

Worker tasks on a Spark cluster can add values to an Accumulator with the += operator, but only the driver program is allowed to access its value, using value. Updates from the workers get propagated automatically to the driver program.

In [19]:
```
def accum_example(x): # accum_example function takes x(input) and updates the acc_sum by adding the x value to it.
  global acc_sum
  acc_sum += x
```

In [28]:
```
# Accumulator variables are used for aggregating the information through associative and commutative operations
# See pyspark.Accumulator documentation on how to use the spark.sparkContext.accumulator(0)
# It also supports data types like int and float

lst2 = [1, 2, 3, 4, 5]
acc_sum = spark.sparkContext.accumulator(0) # the accumulator. It uses the accumulator method, initializing with 0.
rdd = spark.sparkContext.parallelize(lst2) # using parallelize() function to create rdd from a list collection

rdd.foreach(accum_example) # applies accum_example function to each element in the rdd using foreach
accumulator_value = acc_sum.value
print('Accumulator value:', accumulator_value) # returns acc sum
```

```
Accumulator value: 15
```

Other example that will also return 15:

In [18]:
```
accum=spark.sparkContext.accumulator(0)
rdd=spark.sparkContext.parallelize([1,2,3,4,5])
rdd.foreach(lambda x:accum.add(x))
print(accum.value)
```

```
15
```

When to use?

`Broadcast variables` are used to save the copy of data across all nodes. This variable is used, for example, to allow the programmer to keep a read-only variable cached on each machine rather than shipping a copy of it with tasks. It is suitable for resubale data that needs to be accessed by multiple tasks. An `accumulator` is used for aggregating the information through associative and commutative operations. Used for tasks like counting operations, sum operations, collecting computations.

Question 3

Why do we need UDF (User Defined Functions)?

UDFs are used to extend the functions of the framework and re-use these functions on multiple DataFrames. This means we can create our functions to perform operations on dataframe columns and reuse these functions across multiple dataframes and SQL Expression.

In [42]:
```
from pyspark.sql.functions import udf, col
```

In [38]:
```
# Function to return upper case letters. Applied to UDF new column
def caps_lock(str):
    return str.upper()
```

In [34]:
```
spark = SparkSession.builder.appName('UDF App').getOrCreate()
```

Creating dataframe containing 2 columns: CustomerID and Customer Name.

In [54]:
```
columns = ['CustomerID', 'Customer Name']

data = [('123', 'Vicente De Leon'),
        ('456', 'Carolina Licona'),
        ('789', 'Max Licona')]

df3 = spark.createDataFrame(data = data, schema = columns)
df3.show(truncate = False)
```

```
+----------+---------------+
|CustomerID|Customer Name  |
+----------+---------------+
|123       |Vicente De Leon|
|456       |Carolina Licona|
|789       |Max Licona     |
+----------+---------------+
```

Converting caps_lock() function to UDF and then use it with dataframe withColumn(). Example, creating the Caps Lock Name column to get the upper case lleter Customer Names.

In [61]:
```
caps_lock_UDF = udf(lambda x: caps_lock(x)) # converting function to UDF, by default -> UDF StringType()
# caps_lock_UDF = udf(lambda z: caps_lock(z), StringType())

# Apply the caps_lock_UDF to create new column -> Upper Case Customer Name
df4 = df3.withColumn('Upper Case Customer Name', caps_lock_UDF(col('Customer Name')))

df4.show(truncate = False)
```

```
+----------+---------------+-----------------------+
|CustomerID|Customer Name  |Upper Case Customer Name|
+----------+---------------+-----------------------+
|123       |Vicente De Leon|VICENTE DE LEON        |
|456       |Carolina Licona|CAROLINA LICONA        |
|789       |Max Licona     |MAX LICONA             |
+----------+---------------+-----------------------+
```

References:

- Broadcast Variables: https://sparkbyexamples.com/pyspark/pyspark-broadcast-variables/
- Accumulators: https://sparkbyexamples.com/pyspark/pyspark-accumulator-with-example/
- Broadcast and Accumulator: https://data-flair.training/blogs/pyspark-broadcast-and-accumulator/
- pyspark.Accumulator: https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.Accumulator.html#:~:text=Accumulator,-class%20pyspark.&text=A%20shared%20variable%20that%20can,access%20its%20value%2C%20using%20value%20.
- parallelize(): https://sparkbyexamples.com/pyspark/pyspark-parallelize-create-rdd/
- tutorial: https://www.tutorialspoint.com/pyspark/pyspark_broadcast_and_accumulator.htm
- tutorial: https://medium.com/@sangee01sankar17/broadcast-and-accumulator-variable-in-pyspark-5506dd32cae7
- UDF: https://sparkbyexamples.com/pyspark/pyspark-udf-user-defined-function/
- UDF: https://medium.com/@vaishalisubbaraj/user-defined-function-in-pyspark-e9740dc2d3bd