In [1]:
```
from pyspark.sql import SparkSession
import pyspark.sql.functions as F
from pyspark.sql.types import *

spark = SparkSession\
    .builder\
    .appName("chapter-14-broadcast-vars")\
    .getOrCreate()

import os
SPARK_BOOK_DATA_PATH = os.environ['SPARK_BOOK_DATA_PATH']

sc = spark.sparkContext
```

In [2]:
```
sc
```

Out[2]:

SparkContext

Spark UI (http://192.168.0.114:4042)

Version
v3.0.1

Master
local[*]

AppName
chapter-14-broadcast-vars

In [3]:
```
my_collection = "Spark The Definitive Guide : Big Data Processing Ma
  .split(" ")
words = sc.parallelize(my_collection, 2)  # numSlices = 2
```

In [4]:
```
type(words)
```

Out[4]:
```
pyspark.rdd.RDD
```

In [5]:
```
1 words.collect()
```

Out[5]:
```
['Spark',
 'The',
 'Definitive',
 'Guide',
 ':',
 'Big',
 'Data',
 'Processing',
 'Made',
 'Simple,',
 'Spark',
 'in',
 'the',
 'Park,',
 'very',
 'powerful']
```

In [6]:
```
1 words.glom().collect()
```

Out[6]:
```
[['Spark', 'The', 'Definitive', 'Guide', ':', 'Big', 'Data', 'Processi
ng'],
 ['Made', 'Simple,', 'Spark', 'in', 'the', 'Park,', 'very', 'powerfu
l']]
```

Broadcast

push a small shared dataset to worker nodes to avoid shuffle

In [7]:
```
1 # map selected word to number
2 supplementalData = {"Spark":1000, "Definitive":200,
3                     "Big":-300, "Simple":100, "Data": 99}
```

In [8]:
```
1 suppBroadcast = sc.broadcast(supplementalData)
```

In [9]:
```
1 type(suppBroadcast)
```

Out[9]:
```
pyspark.broadcast.Broadcast
```

In [10]:
```
1 # access broadcast var
2 suppBroadcast.value
```

Out[10]:
```
{'Spark': 1000, 'Definitive': 200, 'Big': -300, 'Simple': 100, 'Data':
99}
```

In [11]:
```
words.map(lambda word: (word, suppBroadcast.value.get(word, -9999999
  .sortBy(lambda wordPair: wordPair[1])\
  .collect()
```

Out[11]:
```
[('The', -999999999),
 ('Guide', -999999999),
 (':', -999999999),
 ('Processing', -999999999),
 ('Made', -999999999),
 ('Simple,', -999999999),
 ('in', -999999999),
 ('the', -999999999),
 ('Park,', -999999999),
 ('very', -999999999),
 ('powerful', -999999999),
 ('Big', -300),
 ('Data', 99),
 ('Definitive', 200),
 ('Spark', 1000),
 ('Spark', 1000)]
```

Accumulator

global count

In [12]:
```
file_path = SPARK_BOOK_DATA_PATH + "/data/flight-data/parquet/2010-s

flights = spark.read.parquet(file_path)
```

In [13]:
```
accChina = sc.accumulator(0)
```

In [14]:
```
type(accChina)
```

Out[14]:
```
pyspark.accumulators.Accumulator
```

In [15]:
```
def accChinaFunc(flight_row):
    if flight_row["DEST_COUNTRY_NAME"] == "China" or flight_row["ORI
        accChina.add(flight_row["count"])
```

`foreach()` to process each row

In [16]:
```
flights.foreach(lambda flight_row: accChinaFunc(flight_row))
```

In [17]:
```
accChina.value # 953
```

Out[17]:
```
953
```

verify accumulator via DataFrame

In [18]:
```
1 flights.filter("DEST_COUNTRY_NAME == 'China' OR ORIGIN_COUNTRY_NAME
```

```
+-----------------+-------------------+-----+
|DEST_COUNTRY_NAME|ORIGIN_COUNTRY_NAME|count|
+-----------------+-------------------+-----+
|    United States|              China|  505|
|            China|      United States|  448|
+-----------------+-------------------+-----+
```

In [19]:
```
1 flights.where("DEST_COUNTRY_NAME='China' or ORIGIN_COUNTRY_NAME='Chi
```

```
+----------+
|sum(count)|
+----------+
|       953|
+----------+
```

verify accumulator via SQL

In [20]:
```
1 flights.createOrReplaceTempView("flights")
```

In [21]:
```
1 sql_stmt = """
2 select sum(count) as accChina
3 from flights
4 where DEST_COUNTRY_NAME='China' or ORIGIN_COUNTRY_NAME='China'
5 """
6 spark.sql(sql_stmt).show()
```

```
+--------+
|accChina|
+--------+
|     953|
+--------+
```

In [ ]:
```
1
```

RDD.glom()

Return an RDD created by coalescing all elements within each partition into a list.

Examine how data is partitioned

In [22]:
```
1 rdd = sc.parallelize(range(15), 4)
```

In [23]:
```
1 type(rdd)
```

Out[23]:
```
pyspark.rdd.PipelinedRDD
```

In [24]:
```
1 rdd.getNumPartitions()
```

Out[24]: 4

In [25]:
```
1 rdd.collect()
```

Out[25]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]

In [26]:
```
1 rdd.glom().collect()
```

Out[26]: [[0, 1, 2], [3, 4, 5, 6], [7, 8, 9, 10], [11, 12, 13, 14]]

In [27]:
```
1 sc.parallelize([0, 2, 3, 4, 6, 7], 5).glom().collect()
2 # [[0], [2], [3], [4], [6]]
```

Out[27]: [[0], [2], [3], [4], [6, 7]]

In [ ]:
```
1
```