# Tema 3: Procesamiento con Apache Spark

Máster en Ciencia de Datos (Universidad de Córdoba)

UCO ONLINE

# Transformaciones y Acciones con RDDs

Tema 3: Procesamiento con Apache Spark

UCO ONLINE

# Índice de la sección

- Transformaciones Vs Acciones
- Transformaciones
- Acciones

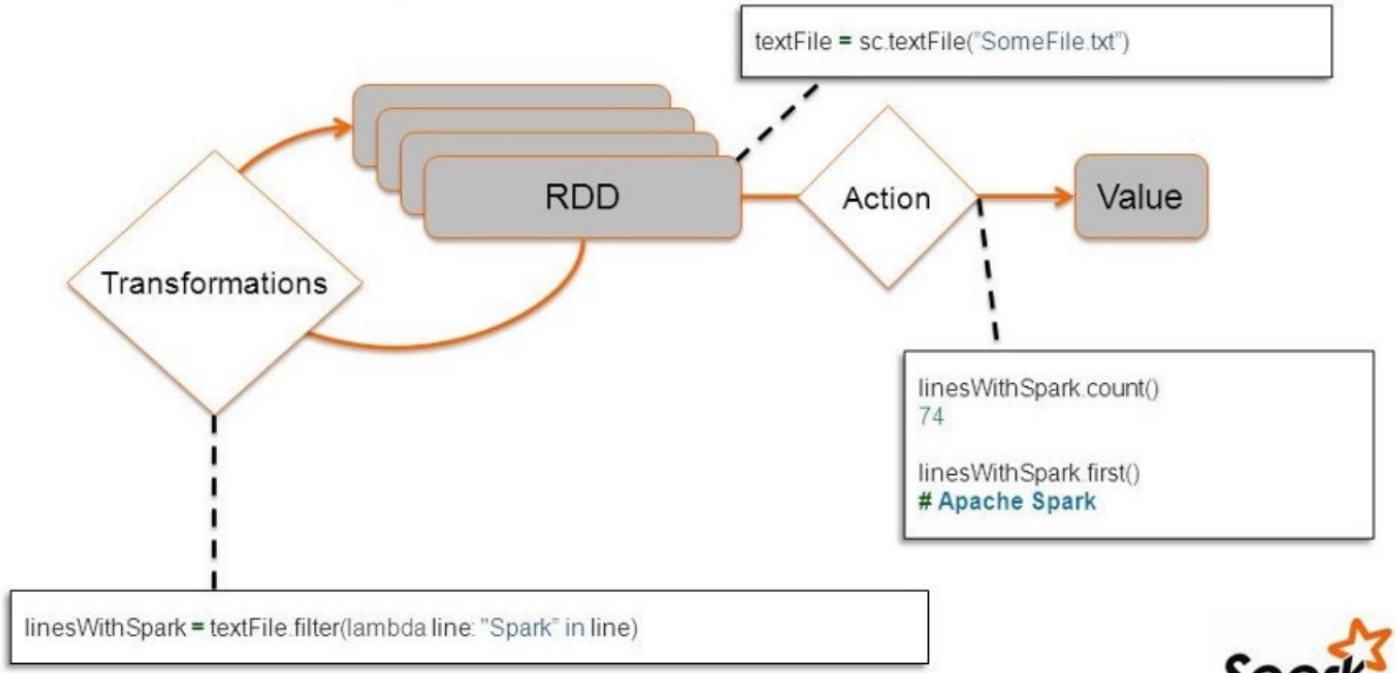
# Transformaciones Vs Acciones (I)

- Dentro de Apache Spark se tienen una serie de operaciones para realizar el procesamiento de los datos, estos datos pueden ser ingestados directamente desde alguna fuente externa o cargados desde un fichero.
- La diferencia principal entre operaciones de transformación sobre las acciones es que las transformaciones van a generar un nuevo RDD, sin embargo las acciones generan un nuevo valor a partir de esos datos.

# Transformaciones Vs Acciones (II)

- La transformación genera un nuevo RDD obteniendo un nuevo mapa de los datos.
- Las acciones obtienen un valor que es mandado al Driver, obteniendo una reducción de los mismos.
- Si unimos las transformaciones y acciones, estamos aplicando un sistema Map-Reduce.

# Transformaciones Vs Acciones (III)



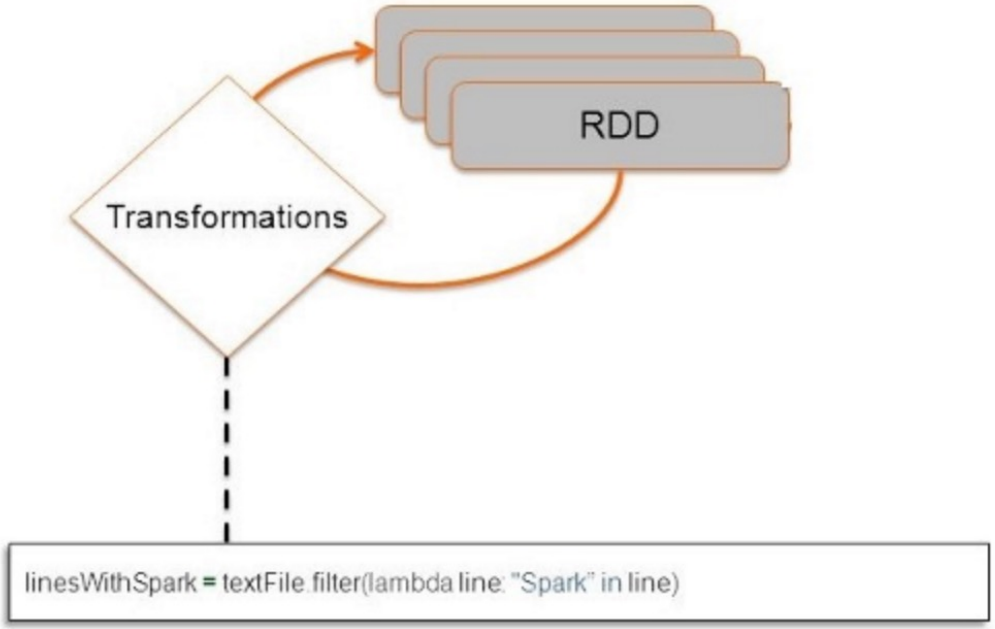
# Transformaciones (I)

- Este primer grupo de funciones son las encargadas de que los datos puedan ser manipulados de muchas formas. De este modo, se pueden obtener resultados relevantes del análisis que se esta desarrollando.
- Es importante resaltar que, las transformaciones no son aplicadas sobre el RDD original si no que, se crea un nuevo RDD que contiene los cambios realizados a los datos del origen.

# Transformaciones (II)

- Este tipo de comportamiento es llamado *Lazy (Evaluación Perezosa)*, todas las funciones de este grupo se comportan de esta manera. Esto quiere decir que, las transformaciones no son ejecutadas en el instante en que son declaradas si no que, se ejecutan cuando se solicita un resultado sobre cualquier RDD (haciendo que todas las transformaciones que participaron en la creación del mismo sean calculadas en ese instante).

# Transformaciones (III)



# Transformaciones (IV)

## Algunas de las transformaciones más usuales

- map
- filter
- flatMap
- union
- intersection
- distinct
- groupByKey
- reduceByKey
- sortByKey
- join
- cogroup
- coalesce

# Transformación (VI)

- Partimos de la base que ya tenemos como fichero de ejemplo

Crear un array de cadenas.

```
[6] cadenas={"David Martin","spark","Master Ciencia Datos", "spark", "Programación"}
    cadenas
```

```
{'David Martin', 'Master Ciencia Datos', 'Programación', 'spark'}
```

Cargar y paralelizar Cadenas

```
[7] from pyspark import SparkContext, SparkConf

    conf = SparkConf().setAppName("Colab").setMaster("local[*]")
    sc = SparkContext.getOrCreate(conf)
    cadenasRDD = sc.parallelize(cadenas,3)
    cadenasRDD
```

```
ParallelCollectionRDD[0] at readRDDFromFile at PythonRDD.scala:274
```

```
cadenasRDD.collect()
```

```
['Programación', 'Master Ciencia Datos', 'David Martin', 'spark']
```

# Transformación “map”

- La sintaxis es: map(f[, preservesPartitioning]) , en este caso se tiene una transformación sobre el RDD original que nos presenta el resultado de aplicar la función o transformación a cada elemento del RDD. En el ejemplo se ha convertido a mayúscula todos y cada uno de los elementos del RDD original, manteniendo este intacto y devolviendo el resultado en otro RDD.

Transformación MAP - Convertir cada palabra a mayúscula.

```
mayusculaRDD = cadenasRDD.map(lambda p:  p.upper())
mayusculaRDD.collect()
```

```
['PROGRAMACIÓN', 'MASTER CIENCIA DATOS', 'DAVID MARTIN', 'SPARK']
```

# Transformación “filter”

- La sintaxis es: filter(f), en este caso se tiene una transformación sobre el RDD original que nos presenta como resultado un nuevo RDD que es consecuencia de aplicar el predicado que se indida en la función a cada elemento para ver si se satisface o no.

Transformación Filter- En este caso genera una nueva transformación de la ya existente.

```
filtrado = mayusculaRDD.filter(lambda linea:linea.endswith('DATOS'))
filtrado.collect()
```

```
['MASTER CIENCIA DATOS']
```

# Transformación “flatMap”

- La sintaxis es: flatMap(f[, preservesPartitioning]), en este caso se tiene una transformación sobre el RDD original que nos presenta como resultado un nuevo RDD aplicando primero una función a todos los elementos de este RDD y luego aplanando los resultados.

Transformación flatMap

```
mayuscLength = cadenasRDD.flatMap(lambda x: [(x.split("\n"),1)])
mayuscLength.collect()
```

```
[(['Programación'], 1),
 (['Master Ciencia Datos'], 1),
 (['David Martin'], 1),
 (['spark'], 1)]
```

# Transformación “union”

- La sintaxis es: union(otherRDD), en este caso se tiene una transformación sobre el RDD original que nos presenta como resultado un nuevo RDD aplicando al primero de los RDD, la unión con el segundo RDD, obteniendo uno nuevo con todos los elementos en conjunto del primero y el segundo.

Transformación de Union. Une dos RDD, como unión de conjuntos

```
cadenasUnidas = cadenasRDD.union(mayusculaRDD)
cadenasUnidas.collect()
```

```
['Programación',
 'Master Ciencia Datos',
 'David Martin',
 'spark',
 'PROGRAMACIÓN',
 'MASTER CIENCIA DATOS',
 'DAVID MARTIN',
 'SPARK']
```

# Transformación “intersection”

- La sintaxis es: intersection(otherRDD), en este caso se tiene una transformación sobre el RDD original que nos presenta como resultado un nuevo RDD aplicando al primero de los RDD, la intersección con el segundo RDD, obteniendo uno nuevo con los elementos en conjunto del primero que están en el segundo.

Transformación Interseccion

```
interseccionRDD = cadenasUnidas.intersection(cadenasRDD)
cadenasUnidas.collect()
```

```
['Programación',
 'Master Ciencia Datos',
 'David Martin',
 'spark',
 'PROGRAMACIÓN',
 'MASTER CIENCIA DATOS',
 'DAVID MARTIN',
 'SPARK']
```

# Transformación “distinct”

- La sintaxis es: distinct([numPartitions]), en este caso se tiene una transformación sobre el RDD original que nos presenta como resultado un nuevo RDD sólo con los elementos no repetidos del RDD original. Opcionalmente a este método se le puede indicar el número de particiones que se desean realizar sobre el RDD resultado.

Transformación Distinct

```
[15] distinct = cadenasRDD.distinct()
     distinct.collect()
```

```
['Programación', 'David Martin', 'Master Ciencia Datos', 'spark']
```

```
distinct = cadenasRDD.distinct(1)
distinct.collect()
```

```
['Programación', 'Master Ciencia Datos', 'David Martin', 'spark']
```

# Transformación “groupByKey” (I)

- La sintaxis es: groupByKey(f[, numPartitions, partitionFunc]), en este caso se tiene una transformación sobre el RDD original que nos presenta como resultado un nuevo RDD agrupando los items.

```
repetidas=cadenasUnidas.union(mayusculaRDD)
pair = repetidas.map(lambda p: (p,1))
pair.collect()
```

```
[('Programación', 1),
 ('Master Ciencia Datos', 1),
 ('David Martin', 1),
 ('spark', 1),
 ('PROGRAMACIÓN', 1),
 ('MASTER CIENCIA DATOS', 1),
 ('DAVID MARTIN', 1),
 ('SPARK', 1),
 ('PROGRAMACIÓN', 1),
 ('MASTER CIENCIA DATOS', 1),
 ('DAVID MARTIN', 1),
 ('SPARK', 1)]
```

# Transformación “groupByKey” (I)

- La sintaxis es: groupByKey(f[, numPartitions, partitionFunc]), en este caso se tiene una transformación sobre el RDD original que nos presenta como resultado un nuevo RDD agrupando los items.

Agrupando los datos repetidos nos aparece tras cada elemento el número de veces

```
sorted(pair.groupByKey().mapValues(len).collect())
```

```
[('DAVID MARTIN', 2),
 ('David Martin', 1),
 ('MASTER CIENCIA DATOS', 2),
 ('Master Ciencia Datos', 1),
 ('PROGRAMACIÓN', 2),
 ('Programación', 1),
 ('SPARK', 2),
 ('spark', 1)]
```

# Transformación “reducedByKey” (I)

- La sintaxis es: reduceByKey(func[, numPartitions, partitionFunc]), en este caso se tiene una transformación sobre el RDD original que nos presenta como resultado un nuevo RDD combinando los valores de cada clave mediante una función de reducción asociativa y conmutativa

Transformacion ReducedbyKey

Otra forma de hacerlo con reducesByKey, se debe utilizar en agregaciones

```
reducido=pair.reduceByKey(lambda a,b: a+b)
for element in reducido.collect():
    print(element)
```

```
('Programación', 1)
('David Martin', 1)
('SPARK', 2)
('Master Ciencia Datos', 1)
('PROGRAMACIÓN', 2)
('MASTER CIENCIA DATOS', 2)
('DAVID MARTIN', 2)
('spark', 1)
```

# Transformación “reducedByKey” (II)

- La sintaxis es: reduceByKey(func[, numPartitions, partitionFunc]), en este caso se tiene una transformación sobre el RDD original que nos presenta como resultado un nuevo RDD combinando los valores de cada clave mediante una función de reducción asociativa y conmutativa

Uso de Transformación para ordenación sortedbykey

```
sort = reducido.sortByKey(2)
sort.collect()
```

```
[('DAVID MARTIN', 2),
 ('David Martin', 1),
 ('MASTER CIENCIA DATOS', 2),
 ('Master Ciencia Datos', 1),
 ('PROGRAMACIÓN', 2),
 ('Programación', 1),
 ('SPARK', 2),
 ('spark', 1)]
```

# Otras Transformaciones (I)

| Función Transformación | Sintaxis | Descripción |
|---|---|---|
| coalesce | coalesce(numPartitions[, shuffle]) | Devuelve un nuevo RDD que se reduce a numPartitions particiones. |
| cartesian | cartesian(other) | Devuelve el producto cartesiano de este RDD y otro, es decir, el RDD de todos los pares de elementos (a, b) donde a está en uno mismo y b está en otro. |
| cogroup | cogroup(other[, numPartitions]) | Para cada clave k en propio RDD origen u other, devuelve un RDD resultante que contiene una tupla con la lista de valores para esa clave en el original y other. |

# Otras Transformaciones (II)

| Función Transformación | Sintaxis | Descripción |
|---|---|---|
| join | join(other[, numPartitions]) | Retorna un RDD que contiene todos los pares de elementos con claves coincidentes en el original y other. |
| sotByKey | sortByKey([ascending, numPartitions, keyfunc]) | Devuelve el producto cartesiano de este RDD y otro, es decir, el RDD de todos los pares de elementos (a, b) donde a está en uno mismo y b está en otro. |
| groupByKey | groupByKey([numPartitions, partitionFunc]) | Agrupa los valores de cada clave en el RDD en una sola secuencia. |

# Otras Transformaciones (III)

| Función Transformación | Sintaxis | Descripción |
|---|---|---|
| sortBy | sortBy(keyfunc[, ascending, numPartitions]) | Ordena este RDD por la función clave dada |
| mapPartitions | mapPartitions(f[, preservesPartitioning]) | Devuelve un nuevo RDD aplicando una función a cada partición de este RDD. |
| mapPartitiosnWithIndex | mapPartitionsWithIndex(f[, …]) | Devuelva un nuevo RDD aplicando una función a cada partición de este RDD, mientras rastrea el índice de la partición original. |
| mapPartitionsWithSplit | mapPartitionsWithSplit(f[, …]) | Devuelva un nuevo RDD aplicando una función a cada partición de este RDD, mientras rastrea el índice de la partición original. |

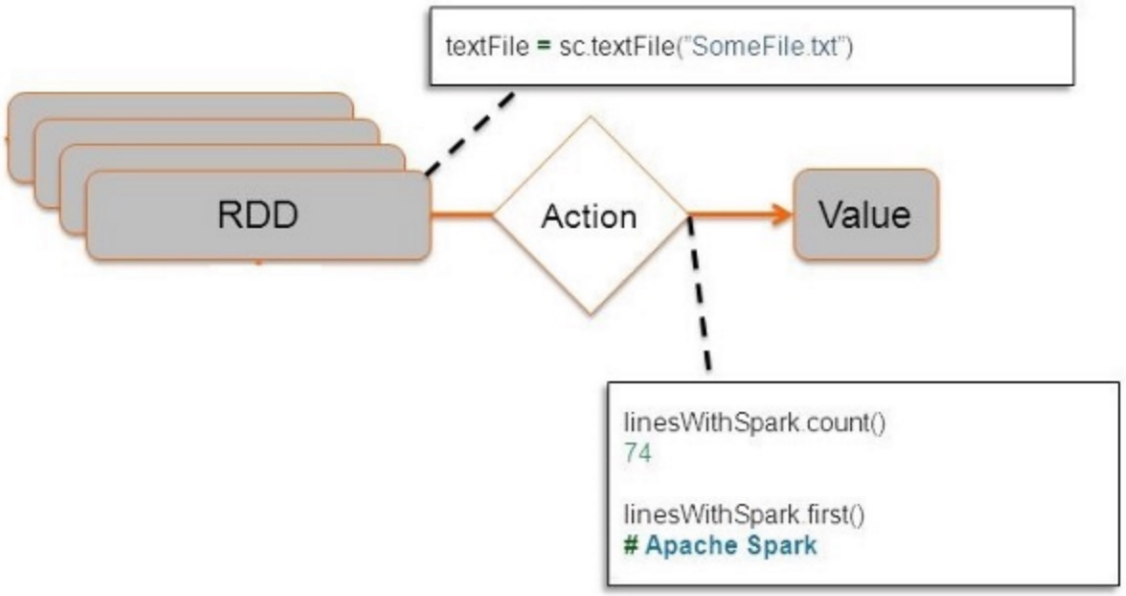
# Otras Transformaciones (IV)

- Para más información sobre transformaciones podemos visitar la web de referencia oficial de Apache Spark
- https://spark.apache.org/docs/latest/rdd-programming-guide.html#transformations
- Además podemos consultar la API de Python para Transformaciones con Spark:
- https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.RDD.html#pyspark.RDD

# Acciones (I)

- Una vez que los datos se encuentran en el formato que necesitaba para llevar acabo el análisis, se llega al punto de obtener los resultados.
- Para esto se introduce el segundo concepto, las acciones. Al igual que con las transformaciones existen varios tipos acciones que nos pueden ayudar a mostrar los datos de diferentes maneras.

# Acciones (II)


textFile = sc.textFile("SomeFile.txt")
RDD
Action
Value
linesWithSpark.count()
74
linesWithSpark.first()
# Apache Spark

# Acciones (III)

**Algunas de las acciones más usuales**

- reduce
- collect
- count
- first
- take
- foreach
- countByKey
- takeSample
- takeOrdened
- saveAsTextFile
- saveAsSequenceFile
- saveAsObjectFile

# Acciones (IV)

- Al igual que antes partimos del un array de cadenas.

Crear un array de cadenas.

```
[8]  cadenas=["David Martin","spark","Master Ciencia Datos", "spark", "Programación", "spark"]
     cadenas
```

```
['David Martin',
 'spark',
 'Master Ciencia Datos',
 'spark',
 'Programación',
 'spark']
```

# Acciones (V)

- Realizamos la carga y paralelización en un RDD

Cargar y paralelizar Cadenas

```
from pyspark import SparkContext, SparkConf

conf = SparkConf().setAppName("Colab").setMaster("local[*]")
sc = SparkContext.getOrCreate(conf)
cadenasRDD = sc.parallelize(cadenas,3)
cadenasRDD
```

```
ParallelCollectionRDD[0] at readRDDFromFile at PythonRDD.scala:274
```

# Acciones (VI)

- Igualmente podemos realizarlo con un array de números.

Crear un array de números.

```
numeros =[0,2,3,4,1,7,9,8,4,2,4,6]
numerosRDD=sc.parallelize(numeros,3)
numerosRDD.collect()
```

```
[0, 2, 3, 4, 1, 7, 9, 8, 4, 2, 4, 6]
```

# Acción “collect”

- La sintaxis es: collect(), la acción que se produce sobre el RDD original nos devuelve una lista que contiene todos los elementos de este RDD.

```
cadenasRDD.collect()
```

```
['David Martin',
 'spark',
 'Master Ciencia Datos',
 'spark',
 'Programación',
 'spark']
```

# Acción “first”

- La sintaxis es: first(), la acción que se produce sobre el RDD original nos devuelve el primer elemento de este RDD.

Acción first, nos muestra el primer elemento del contenido

```
[12] cadenasRDD.first()
1 s

'David Martin'
```

# Acción “count”

- La sintaxis es: count(), la acción que se produce sobre el RDD original nos devuelve el número de elementos en este RDD.

Acción count, nos muestra el numero de elementos del contenido

```
[13] cadenasRDD.count()+numerosRDD.count()
0 s

     18
```

# Acción “reduce”

- La sintaxis es: reduce(f), la acción que se produce sobre el RDD original reduce los elementos de este RDD utilizando el operador binario conmutativo y asociativo especificado (en este caso se ha aplicado la suma).

Acción Reduce

```
from operator import add
numerosRDD.reduce(add)
```

```
50
```

# Acción “take”

- La sintaxis es: take(num), la acción que se produce sobre el RDD original devuelve los primeros “num” elementos del RDD.

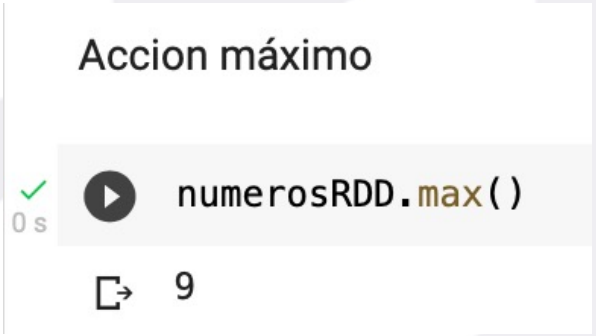
Accion take, tomaría el numero pasado como parámetro de primeros elementos

```
[15] numerosRDD.take(4)
0 s
```

```
[0, 2, 3, 4]
```

# Acción “max”

- La sintaxis es: max([key]), la acción que se produce sobre el RDD original encuentra el elemento máximo en este RDD.



# Acción “min”

- La sintaxis es: min([key]), la acción que se produce sobre el RDD original encuentra el elemento mínimo en este RDD.

# Otras Acciones (I)

| Función Acción | Sintaxis | Descripción |
|---|---|---|
| collectAsMap | collectAsMap() | Devuelva los pares clave-valor en este RDD al maestro como un diccionario. |
| collectWithJobGroup | collectWithJobGroup(groupId, description[, …]) | Cuando recopile el RDD, utiliza este método para especificar el grupo de trabajo. |
| countApprox | countApprox(timeout[, confidence]) | Versión parecida a count() que devuelve un resultado potencialmente incompleto dentro de un tiempo de espera, incluso si no han finalizado todas las tareas. |

# Otras Acciones (II)

| Función Acción | Sintaxis | Descripción |
|---|---|---|
| countApprox Distinct | countApproxDistinct([relativeSD]) | Devuelve el número aproximado de elementos distintos en el RDD. |
| countByKey | countByKey() | Cuente la cantidad de elementos para cada clave y devuelva el resultado como un diccionario. |
| countByValue | countByValue() | Devuelve el recuento de cada valor único en este RDD como un diccionario de (valor, recuento) pares. |
| mean | mean() | Calcula la media de los elementos de este RDD. |

# Otras Acciones (III)

| Función Acción | Sintaxis | Descripción |
|---|---|---|
| foreach | foreach(f) | Aplica una función a todos los elementos de este RDD. |
| foreachPartition | foreachPartition(f) | Cuente la cantidad de elementos para cada clave y devuelva el resultado como un diccionario. |
| countByValue | countByValue() | Aplica una función a cada partición de este RDD. |
| saveAsTextFile | saveAsTextFile(path[, compressionCodecClass]) | Guarda este RDD como un archivo de texto, utilizando representaciones de cadenas de elementos. |

# Otras Acciones (IV)

| Función Acción | Sintaxis | Descripción |
|---|---|---|
| saveAsHadoopDataset | saveAsHadoopDataset(conf[, keyConverter, …]) | Envía un RDD de Python de pares clave-valor (del formato RDD[(K, V)]) a cualquier sistema de archivos Hadoop, utilizando la antigua API Hadoop OutputFormat (paquete mapred). |
| saveAsHadoopFile | saveAsHadoopFile(path, outputFormatClass[, …]) | Envía un RDD de Python de pares clave-valor (del formato RDD[(K, V)]) a cualquier sistema de archivos Hadoop, utilizando la antigua API Hadoop OutputFormat (paquete mapred). |

# Otras Acciones (VI)

| Función Acción | Sintaxis | Descripción |
|---|---|---|
| saveAsNewAPIHadoopDataset | saveAsNewAPIHadoopDataset(conf[, …]) | Envía un RDD de Python de pares clave-valor (del formato RDD[(K, V)]) a cualquier sistema de archivos Hadoop, utilizando la nueva API Hadoop OutputFormat (paquete mapreduce). |
| saveAsNewAPIHadoopFile | saveAsNewAPIHadoopFile(path, outputFormatClass) | Envía un RDD de Python de pares clave-valor (del formato RDD[(K, V)]) a cualquier sistema de archivos Hadoop, utilizando la nueva API Hadoop OutputFormat (paquete mapreduce). |

# Otras Acciones (VII)

| Función Acción | Sintaxis | Descripción |
|---|---|---|
| sum | sum() | Envía un RDD de Python de pares clave-valor (del formato RDD[(K, V)]) a cualquier sistema de archivos Hadoop, utilizando la nueva API Hadoop OutputFormat (paquete mapreduce). |
| sumApprox | sumApprox(timeout[, confidence]) | Operación aproximada para devolver la suma dentro de un tiempo de espera o cumplir con la confianza. |
| stdev | stdev() | Calcula la desviación estándar de los elementos de este RDD. |

# Otras Acciones (VIII)

| Función Acción | Sintaxis | Descripción |
|---|---|---|
| takeOrdered | takeOrdered(num[, key]) | Devuelve los N elementos de un RDD ordenados en orden ascendente o según lo especificado por la función clave. |
| takeSample | takeSample(withReplacement, num[, seed]) | Devuelve un subconjunto muestreado de tamaño fijo de este RDD. |
| variance | variance() | Calcula la varianza de los elementos de este RDD. |

# Otras Acciones (IX)

- Para más información sobre acciones podemos visitar la web de referencia oficial de Apache Spark
- https://spark.apache.org/docs/latest/rdd-programming-guide.html#actions
- Además podemos consultar la API de Python para Acciones con Spark:
- https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.RDD.html#pyspark.RDD

# Descargas (I)

- El cuaderno de Google Colab con los ejemplos vistos en esta presentación sobre **transformaciones** puede verse en la siguiente dirección:
- https://drive.google.com/file/d/1LkvAIX6q7CHBgG0RU7dTzt-U2xMuTmPY/view?usp=sharing
- El cuaderno de Google Colab con los ejemplos vistos en esta presentación sobre **acciones** puede verse en la siguiente dirección:
- https://drive.google.com/file/d/155Nxvs0qFuWfv-5SA0wgbhK9FP-Upgsw/view?usp=sharing

# ¡Gracias!

UCO ONLINE