

Apache Spark - Core

Máster en Data Science y Big Data

Miguel Ángel Corella

mcorella@geoblink.com

Marzo 2022

Contenido

1. Introducción
2. Procesos
3. RDDs
4. Pair RDDs
5. Variables compartidas
6. Referencias

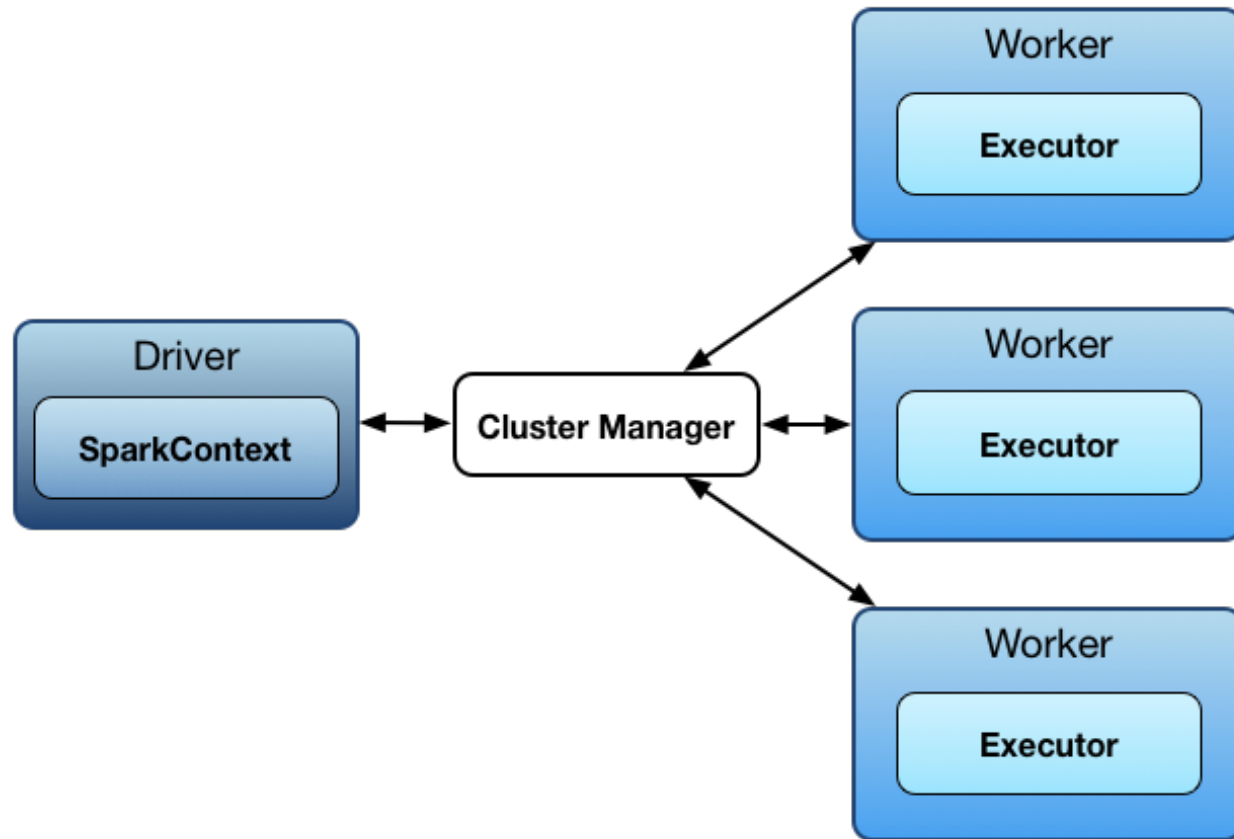
Introducción

¿Qué es Apache Spark Core?

- Se trata del componente fundamental de Spark.
- Es la base de todos los demás componentes del *framework* (SQL, Streaming, MLlib, GraphX).
- Al igual que Hadoop se componía de:
 - Almacenamiento de datos (NO ESTRUCTURAS DE DATOS): **HDFS**.
 - Procesamiento: **MapReduce**.
- Spark Core se puede “dividir” en elementos básicos:
 - Estructuras de datos (NO ALMACENAMIENTO DE DATOS): **RDDs**.
 - Procesamiento: **Transformaciones** / **Acciones**.

Procesos

Arquitectura de un proceso



Componentes de un proceso

- **Driver:**

- Componente principal de cualquier aplicación Spark. donde se definen RDDs, transformaciones y acciones sobre los mismos.
- Se ejecuta en el nodo principal de un *cluster*.

- **SparkContext:**

- Componente que da acceso a Spark, similar a una conexión a BD.
- Internamente crea el DAGScheduler que se encargará de la gestión de tareas y la comunicación con el gestor del *cluster*.

- **Cluster Manager:**

- Es el gestor de recursos/tareas del cluster que se encargará de gestionar la ejecución distribuida de las mismas (será el propio Spark, YARN o Mesos).


- **Worker/Executor:**

- Nodo de ejecución de tareas de datos.

Driver y Context por modo de ejecución

- **Spark como kernel:**
 - Driver: La propia consola o notebook de PySpark.
 - SparkContext: Creado automáticamente en el momento de la inicialización.
- **Spark como servicio:**
 - Driver: La propia consola o notebook de PySpark.
 - SparkContext: Creado manualmente como parte de la ejecución.
- **Spark como job:**
 - Driver: Creado automáticamente a partir del script que se envía (submit) al cluster.
 - SparkContext: Creado manualmente como parte de la ejecución.

Estructura de un proceso

- Todos los procesos de Spark Core (y de Spark en general) siguen, aproximadamente, la misma estructura.
1. Configuración de entorno / aplicación / contexto.
 2. Carga de datos en RDDs.
 3. Manipulación de RDDs.
 4. Generación de resultados.
 5. Cierre / liberación del entorno / aplicación / contexto → **IMPORTANTE**
- 
- A vertical bracket groups steps 2, 3, and 4. An arrow points from the middle of the bracket to the text 'Repetir n veces'.

RDDs

RDDs – Resilient Distributed Datasets

- Es la estructura de datos fundamental de Spark, que presenta las siguientes propiedades:
 - **Resilient:** tolerancia a fallos, replicación y particionado.
 - **Distributed:** distribución en diferentes máquinas físicas / *cluster* de forma que cualquier proceso ejecutado sobre los mismos se puede distribuir y paralelizar.
 - **Dataset:** puede contener tanto cualquier tipo de datos básico como compuestos (tuplas, listas, objetos...) de Python, Scala, R o Java.
 - **In-memory:** el contenido de un RDD se almacena en memoria tanto como sea posible (en términos de espacio y tiempo).
 - **Read-only:** el contenido de un RDD no puede modificarse.
- Se pueden crear de dos maneras:
 - Paralelizando una colección de datos creada localmente.
 - Cargando un fichero existente en un sistema de ficheros (local o distribuido).

Creación de RDDs

- Cualquier RDD debe crearse sobre un SparkContext (más sobre qué es un SparkContext un poco más adelante) utilizando alguna de las siguientes funciones:

Instrucción	Descripción
sc.parallelize(col)	Crea un RDD que distribuye el contenido de "col" entre los nodos del <i>cluster</i>
sc.textFile(path)	Crea un RDD sobre un fichero de un sistema de ficheros local (la ruta tiene que ser válida en TODOS los nodos del <i>cluster</i>) o distribuido

Juguemos un poco...



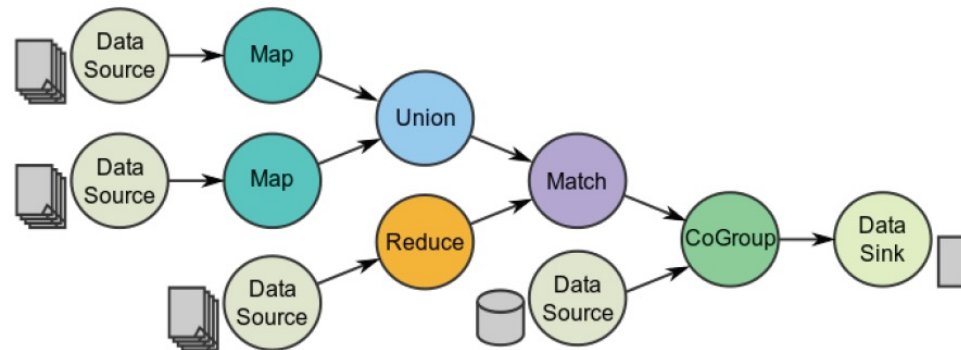
RDD - Creation.ipynb

Operaciones sobre RDDs

- Hay dos tipos de operaciones que se pueden realizar sobre un RDD:
 - **Transformaciones:** reciben un RDD como entrada y generan un RDD modificado a la salida.
 - **Acciones:** reciben un RDD como entrada, realizan algún tipo de computación sobre el mismo y generan como salida un valor resultante o una operación de persistencia/guardado de resultados.
- La clave de la velocidad de Spark es que **ÚNICAMENTE LAS ACCIONES** producen “realmente” una ejecución distribuida en el cluster → *Lazy evaluation*.
- Spark se encarga (automáticamente) de organizar la ejecución de operaciones sobre un RDD de forma que:
 - Se minimice el uso de memoria (e.g. parseo + filtrado vs. filtrado + parseo).
 - Se maximice la reutilización de datos ya cargados en memoria.

DAG y Scheduler

- Para conseguir esta optimización Spark crea, para cada programa ejecutado, un grafo acíclico dirigido (DAG) donde:
 - Los nodos son operaciones a realizar.
 - Las aristas establecen la precedencia de dichas operaciones.
- A su vez, crea un “orquestador” o Scheduler que:
 - Construye/recalcula el DAG con cada nueva **transformación** solicitada.
 - Convierte, cuando se ejecuta una **acción**, las operaciones en tareas ejecutables.



Transformaciones

- Una transformación se ejecuta sobre un RDD y genera otro RDD a la salida.
- Se pueden encadenar y se ejecutan en el orden óptimo al solicitar la primera acción.

Transformaciones mediante aplicación de una función

Instrucción	Descripción
<code>rdd.map(func)</code>	Aplica "func" a cada elemento del RDD (un resultado por elemento)
<code>rdd.flatMap(func)</code>	Aplica "func" a cada elemento del RDD y "aplana" el resultado
<code>rdd.filter(func)</code>	Filtra aquellos elementos del RDD para los que "func" devuelve False
<code>rdd.groupBy(func)</code>	Devuelve los elementos de RDD agrupados según el resultado de "func"
<code>rdd.sortBy(func, asc)</code>	Devuelve los elementos de RDD ordenados según el resultado de "func"

Transformaciones

Transformaciones basadas en funciones de conjunto

Instrucción	Descripción
<code>rdd.union(rdd)</code>	Devuelve la unión de dos RDD
<code>rdd.intersection(rdd)</code>	Devuelve la intersección de dos RDD
<code>rdd.cartesian(rdd)</code>	Devuelve el producto cartesiano de los elementos de dos RDD

Otras transformaciones

Instrucción	Descripción
<code>rdd.distinct()</code>	Devuelve los elementos únicos del RDD (no preserva el orden)
<code>rdd.sample(rep, prob, seed)</code>	Devuelve una muestra del RDD aplicando los parámetros establecidos

<http://training.databricks.com/visualapi.pdf>

Juguemos un poco...



RDD - Transformations.ipynb

Acciones

- Una acción se ejecuta sobre un RDD y produce:
 - La ejecución “real” de todas las transformaciones previas sobre los RDD involucrados.
 - La obtención de un resultado (RDD, objeto o fichero) tras la aplicación de una función (de usuario o no) sobre el RDD resultante de todas las transformaciones previas.

Acciones mediante aplicación de una función

Instrucción	Descripción
<code>rdd.reduce(func)</code>	Agrega el RDD aplicando “func” a cada par de elementos (recursivamente)

Acciones de conversión de datos Spark a Python

Instrucción	Descripción
<code>rdd.collect()</code>	Devuelve una colección con todos los elementos del RDD
<code>rdd.take(n)</code>	Devuelve una colección con los n primeros elementos del RDD
<code>rdd.first()</code>	Devuelve el primer elemento del RDD

Acciones

Acciones de aplicación de funciones matemáticas (“reduce predefinido”)

Instrucción	Descripción
<code>rdd.count()</code>	Cuenta el número de elementos en el RDD
<code>rdd.max()</code>	Devuelve el máximo de los elementos incluidos en el RDD
<code>rdd.min()</code>	Devuelve el mínimo de los elementos incluidos en el RDD
<code>rdd.sum()</code>	Devuelve la suma de los elementos del RDD
<code>rdd.mean()</code>	Devuelve la media de los elementos del RDD
<code>rdd.stdev()</code>	Devuelve la desviación estándar de los elementos del RDD
<code>rdd.variance()</code>	Devuelve la varianza de los elementos del RDD

Acciones de persistencia

Instrucción	Descripción
<code>rdd.saveAsTextFile(path)</code>	Guarda el RDD como ficheros de texto plano en “path” (directorio)

<http://training.databricks.com/visualapi.pdf>

Juguemos un poco...



RDD -Actions.ipynb

Pair RDDs

Pair RDDs

- Un Pair RDD es un tipo especial de RDD en el que la información almacenada tiene la **estructura clave-valor** (en Python una tupla/lista de 2 elementos)
- Se pueden crear mediante la aplicación de un map inicial a un RDD en el que se crea la estructura clave-valor manualmente (se devuelve una tupla/lista por cada elemento).
- Todas las **transformaciones y acciones disponibles para RDDs** pueden usarse también para Pair RDDs (teniendo cuidado en que ahora cada elemento es una tupla/lista).
- Adicionalmente, Spark incluye algunas **transformaciones y acciones específicas** que sólo aplican a Pair RDDs.

Transformaciones (sólo Pair RDDs)

Transformaciones basadas en operaciones por clave

Instrucción	Descripción
<code>rdd.sortByKey(asc)</code>	Devuelve el RDD con los elementos ordenados por clave
<code>rdd.reduceByKey(func)</code>	Devuelve un RDD resultante de aplicar "func" a los elementos de cada clave

Transformaciones basadas en cruces

Instrucción	Descripción
<code>rdd.join(rdd)</code>	Sobre (K, V) y (K, W) devuelve $(K, (V, W))$ para los K existentes en ambos RDD
<code>rdd.leftOuterJoin(rdd)</code>	join pero manteniendo los K que estén en el primer RDD y no en el segundo
<code>rdd.rightOuterJoin(rdd)</code>	join pero manteniendo los K que estén en el segundo RDD y no en el primero
<code>rdd.fullOuterJoin(rdd)</code>	join pero manteniendo todos los K aunque no haya coincidencia

<http://training.databricks.com/visualapi.pdf>

Juguemos un poco...



PairRDD - Transformations.ipynb

Acciones (sólo Pair RDDs)

Acciones basadas en operaciones por clave

Instrucción	Descripción
<code>rdd.countByKey()</code>	Cuenta el número de elementos en cada una de las claves

Acciones de extracción de elementos del Pair RDD

Instrucción	Descripción
<code>rdd.keys()</code>	Obtiene un RDD a partir de las claves del RDD original
<code>rdd.values()</code>	Obtiene un RDD a partir de los valores del RDD original

<http://training.databricks.com/visualapi.pdf>

Variables compartidas

Ejecución *standalone* vs *cluster*

- Imaginemos el siguiente código...

```
counter = 0
rdd = sc.parallelize(data)

def increment_counter(x):
    global counter
    counter += x

rdd.map(increment_counter)
print("Counter value: ", counter)
```

- ¿Qué valor tendrá “counter” al finalizar la ejecución del mismo sobre un cluster?

Variables compartidas

- NO podemos utilizar variables locales en funciones ejecutadas de forma distribuida en Spark.
- Aunque el resultado puede ser correcto ejecutado en local, no tiene por qué serlo al ejecutarlo en el cluster.
- Spark pone a nuestra disposición dos herramientas que nos permiten trabajar con “variables compartidas” entre nodos del cluster:
 - Accumulators.
 - Broadcast variables.

Accumulators

- Un accumulator es una **variable numérica** visible y accesible desde todos los nodos de un cluster involucrados en la ejecución de un proceso.
- Sobre esta variable numérica únicamente se pueden ejecutar **operaciones de incremento**, por lo que podremos utilizarlos para almacenar:
 - Contadores.
 - Agregaciones numéricas.
- Casos de uso:
 - Debug: contar “maps” en los que se da una situación específica.
 - Traza: número total de registros procesados hasta un momento dado.
 - ...

Accumulators

Creación de accumulators

Instrucción	Descripción
<code>sc.accumulator(initial)</code>	Crea un accumulator y lo inicializa al valor de "initial"

Incremento de accumulators

Instrucción	Descripción
<code>accum.add(value)</code>	Incrementa el valor del accumulator en "value"

Recuperación del valor de accumulators

Instrucción	Descripción
<code>accum.value</code>	Recupera el valor actual de un accumulator

Broadcast variables

- Una variable broadcast permite crear una **variable de sólo lectura** accesible desde todos los nodos de un cluster involucrados en la ejecución de un proceso.
- Son útiles cuando tenemos algún **dato / estructura (no excesivamente grande)** necesaria para la ejecución de tareas map / reduce de forma distribuida.
- **Evitan tener que distribuir y parsear** datos comunes a todas las tareas.
- Casos de uso:
 - Tareas “dependientes” de resultados previos: valores de filtrado a partir de resultados de map/reduce previos.
 - Tablas de asociación: traducción de códigos a descripciones.
 - ...

Broadcast variables

Creación de broadcast variables

Instrucción	Descripción
<code>sc.broadcast(var)</code>	Crea una variable compartida a partir del valor "local" de "var"

Recuperación del valor de broadcast variables

Instrucción	Descripción
<code>broadcastVar.value</code>	Recupera el valor de una variable compartida

Juguemos un poco...

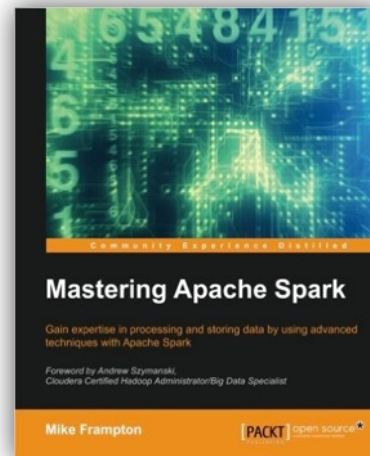
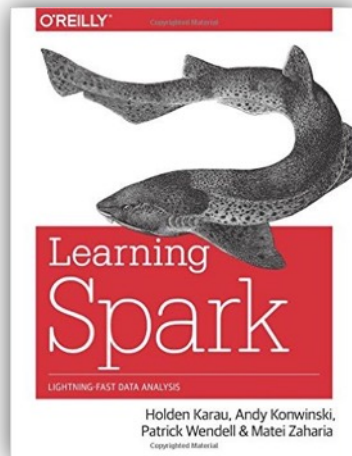


Shared variables.ipynb

Referencias

Referencias

- Documentación oficial:
 - <https://spark.apache.org>
- Tutoriales online:
 - https://www.tutorialspoint.com/apache_spark/
- Libros:





Afi Escuela

© 2022 Afi Escuela. Todos los derechos reservados.