SESIÓN ELEMENTOS BÁSICOS DE SPARK

CONTENIDOS:

- Configuración, conexión y contexto de spark.
- Rdd: qué es un rdd.
- Creación y carga de un rdd.
- Almacenamiento de un rdd.
- Pair rdd.
- Linaje y lazy evaluation.
- Transformaciones: qué es una transformación.
    - Principales métodos (filter, map, flatmap, sample, union, distinct, sortby).
- Acción: qué es una acción.
    - Principales acciones (collect, take, top, takesample, sum, mean, stdev).
- Job spark: qué es una job spark.

CONFIGURACIÓN, CONEXIÓN Y CONTEXTO DE SPARK

Apache Spark es un motor de procesamiento distribuido diseñado para manejar grandes volúmenes de datos en paralelo. Para trabajar con Spark, primero es necesario configurar el entorno y establecer la conexión adecuada.

Ilustración 1 Apache Spark

Configuración de Spark

Para ejecutar Spark, se puede optar por distintas configuraciones según el entorno:

- **Standalone:** Spark se ejecuta en un solo nodo sin dependencia de otros sistemas.

- **Cluster Manager (YARN, Mesos, Kubernetes):** Spark puede integrarse con gestores de clústeres para distribuir la carga de trabajo.
- **Local Mode:** Se ejecuta Spark en un solo nodo, útil para pruebas y desarrollo.

Conexión y Contexto de Spark

El **SparkContext** es el punto de entrada principal para interactuar con Spark. Es el encargado de coordinar los recursos y la ejecución de tareas en el clúster.

Para inicializarlo en Python con PySpark:

```
from pyspark import SparkContext

sc = SparkContext("local", "MiApp")  # "local" indica ejecución en un solo nodo
```

Ilustración 2 SparkContext

Aquí:

- **"local"** especifica el modo de ejecución.
- **"MiApp"** es el nombre de la aplicación en Spark.

En versiones más recientes, se usa **SparkSession**, que unifica **SparkContext**, **SQLContext** y **HiveContext**:

```
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("MiApp").getOrCreate()
sc = spark.sparkContext  # Obtener el SparkContext desde SparkSession
```

Ilustración 3 SparkSession

RDD: QUÉ ES UN RDD

Un **RDD (*Resilient Distributed Dataset*)** es la estructura fundamental de datos en Spark. Es una colección distribuida de elementos que se pueden procesar en paralelo.

Características principales de un RDD:

1. **Inmutable:** Una vez creado, no puede modificarse, solo transformarse en un nuevo RDD.
2. **Distribuido:** Los datos se dividen en particiones y se distribuyen entre los nodos del clúster.
3. **Resiliente:** Soporta fallos debido a su capacidad de reconstrucción a partir de su linaje.

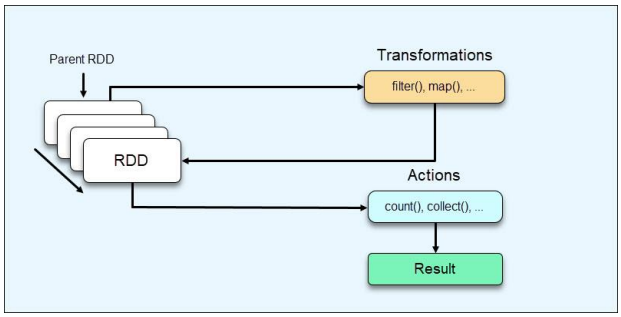

Ilustración 4 Resilient Distributed Datasets

CREACIÓN Y CARGA DE UN RDD

Existen dos maneras de crear un RDD:

1. **Desde una colección en memoria**

```
datos = [1, 2, 3, 4, 5]
rdd = sc.parallelize(datos)  # Convierte la lista en un RDD
```

Ilustración 5 Crear RDD desde colección en memoria

2. **Desde un archivo externo (HDFS, S3, local, etc.)**

```
rdd = sc.textFile("archivo.txt")  # Carga un archivo de texto como RDD
```

Ilustración 6 Crear RDD desde archivo externo

ALMACENAMIENTO DE UN RDD

Los RDD pueden almacenarse temporalmente en memoria o en disco para optimizar el rendimiento.

Persistencia de RDD

Spark ofrece distintas opciones para almacenar un RDD:

```
rdd.persist()  # Almacena en memoria
rdd.cache()    # Similar a persist(), pero con almacenamiento por defecto en memoria
```

Ilustración 7 Almacenamiento en memoria

También se puede elegir el nivel de persistencia:

```
from pyspark import StorageLevel
rdd.persist(StorageLevel.DISK_ONLY)  # Guarda solo en disco
```

Ilustración 8 Guardar en disco

PAIR RDD

Un Pair RDD es un tipo especial de RDD donde cada elemento es una tupla (clave, valor). Se usa para operaciones como agrupaciones y reducciones.

Ejemplo de creación:

```
rdd = sc.parallelize([("A", 1), ("B", 2), ("A", 3)])
```

Ilustración 9 Pair RDD

Operaciones comunes:

- **groupByKey():** Agrupa los valores por clave.
- **reduceByKey():** Aplica una función de reducción por clave.

Ejemplo:

```
rdd.reduceByKey(lambda x, y: x + y).collect()
# Resultado: [('A', 4), ('B', 2)]
```

Ilustración 10 Función de reducción por clave

LINAJE Y LAZY EVALUATION

Linaje (Lineage)

El linaje de un RDD se refiere a su historial de transformaciones. Spark usa este historial para reconstruir los datos en caso de fallos.

Lazy Evaluation

Spark usa **evaluación diferida**: las transformaciones no se ejecutan inmediatamente, sino que se almacenan en un DAG (*Directed Acyclic Graph*). La ejecución real ocurre solo cuando se invoca una acción.

Ejemplo:

```
rdd = sc.parallelize([1, 2, 3, 4])
rdd2 = rdd.map(lambda x: x * 2)  # Transformación (no ejecutada aún)
rdd2.collect()  # Acción (se ejecuta aquí)
```

Ilustración 11 Ejemplo Lazy Evaluation

TRANSFORMACIONES: QUÉ ES UNA TRANSFORMACIÓN

Las transformaciones generan un nuevo RDD a partir de uno existente. Son **lazy**, es decir, no se ejecutan hasta que se aplique una acción.

Principales transformaciones en Spark

1. **filter(f)** → Filtra elementos según una condición.
2. **map(f)** → Aplica una función a cada elemento.
3. **flatMap(f)** → Similar a map, pero aplana los resultados.
4. **sample(fraction, replacement)** → Toma una muestra aleatoria.

5. **union(otherRDD)** → Une dos RDD.
6. **distinct()** → Elimina duplicados.
7. **sortBy(f, ascending=True)** → Ordena los elementos.

```python
# 1. filter
rdd.filter(lambda x: x % 2 == 0).collect()  # Solo pares

# 2. map
rdd.map(lambda x: x * 2).collect()

# 3. flatMap
rdd = sc.parallelize(["hola mundo"])
rdd.flatMap(lambda x: x.split(" ")).collect()
# ['hola', 'mundo']

# 4. sample
rdd.sample(False, 0.5).collect()

# 5. union
rdd.union(rdd2).collect()

# 6. distinct
rdd.distinct().collect()

# 7. sortBy
rdd.sortBy(lambda x: x).collect()
```

Ilustración 12 Transformaciones Spark

ACCIÓN

Las acciones devuelven un resultado o guardan datos.

Principales acciones en Spark

1. **collect()** → Devuelve todos los elementos.
2. **take(n)** → Toma los primeros n elementos.
3. **top(n)** → Devuelve los n valores más altos.
4. **takeSample()** → Devuelve una muestra aleatoria.
5. **sum()** → Suma los valores.
6. **mean()** → Calcula el promedio.
7. **stdev()** → Devuelve la desviación estándar.

```python
# 1. collect
rdd.collect()

# 2. taken
rdd.take(3)

# 3. top
rdd.top(2)

# 4. takeSample
rdd.takeSample(False, 2)

# 5. sum
rdd.sum()

# 6. mean
rdd.mean()

# 7. stdev
rdd.stdev()
```

Ilustración 13 Principales Acciones

JOB SPARK: QUÉ ES UNA JOB SPARK

Un **Job** en Spark es una unidad de trabajo que se ejecuta en el clúster. Se activa cuando se llama a una acción en un RDD o DataFrame.

Ejemplo:

```python
rdd.count()  # Acción que dispara un Job
```

Ilustración 14 Job Spark

Cada job se divide en:

1. **Stages (etapas):** Secciones del job que pueden ejecutarse en paralelo.
2. **Tasks (tareas):** Unidades de ejecución en cada nodo.

Para monitorear los jobs, se puede acceder a la interfaz web de Spark en http://localhost:4040.

ACTIVIDAD PRÁCTICA GUIADA: Análisis de datos con Spark y RDDs

Objetivo

Aplicar los conceptos de configuración, conexión, RDDs, transformaciones y acciones en Apache Spark procesando datos reales.

Dataset

Usaremos un archivo CSV con transacciones bancarias o logs del sistema (podemos elegir una fuente real como los datasets abiertos de Kaggle o generar uno ficticio).

Ejemplo de datos (transacciones.csv):

```
id,usuario,monto,fecha,categoria
1,Juan,1500,2025-03-10,Supermercado
2,Ana,2000,2025-03-12,Tecnología
3,Carlos,500,2025-03-12,Supermercado
4,Juan,3000,2025-03-13,Tecnología
5,Ana,700,2025-03-14,Restaurante
```

Paso 1: Configuración y Conexión con Spark

Iniciar Spark en modo local.

```python
from pyspark.sql import SparkSession

# Crear sesión de Spark
spark = SparkSession.builder.appName("AnalisisTransacciones").getOrCreate()

# Obtener SparkContext
sc = spark.sparkContext
```

Paso 2: Carga de Datos en un RDD

Leer el archivo CSV como un RDD y analizar su contenido.

```
rdd = sc.textFile("transacciones.csv")

# Ver las primeras líneas
rdd.take(5)
```

Observación: La primera línea es el encabezado, lo eliminamos.

```
header = rdd.first()
rdd = rdd.filter(lambda x: x != header)
```

Paso 3: Transformaciones en el RDD

Convertir cada línea en una tupla (usuario, monto, categoria).

```
rdd_mapeado = rdd.map(lambda x: x.split(",")) \
                 .map(lambda x: (x[1], float(x[2]), x[4]))

rdd_mapeado.take(5)
```

Filtrar solo las transacciones mayores a $1000.

```
rdd_filtrado = rdd_mapeado.filter(lambda x: x[1] > 1000)
rdd_filtrado.collect()
```

Obtener los montos totales gastados por usuario.

```
rdd_gastos = rdd_mapeado.map(lambda x: (x[0], x[1])) \
                        .reduceByKey(lambda x, y: x + y)

rdd_gastos.collect()
```

Paso 4: Acciones y Resultados

Obtener estadísticas básicas del gasto total.

```
montos = rdd_mapeado.map(lambda x: x[1])
total = montos.sum()
promedio = montos.mean()
desviacion = montos.stdev()

print(f"Total: {total}, Promedio: {promedio}, Desviación estándar: {desviacion}")
```

Mostrar los 3 usuarios con mayores gastos.

```
top_usuarios = rdd_gastos.sortBy(lambda x: x[1], ascending=False).take(3)
print(top_usuarios)
```

Paso 5: Ejecución de un Job Spark

Cada acción ejecutada dispara un **Job** en Spark. Para ver los jobs:

- Acceder a http://localhost:4040 en el navegador.
- Explicar a los estudiantes cómo Spark maneja cada transformación.