**Datos Masivos**

**Clase 2**

**RDDs: Resilient Distributed Dataset**

Es una de las estructuras de datos fundamentales en el ecosistema de Apache Spark, utilizado para el procesamiento de grandes volúmenes de datos en entornos de Big Data. A continuación, se detallan sus características y funciones clave.

**Características de los RDD**

Inmutabilidad: Una vez que se crea un RDD, no se puede modificar. Esto asegura que los datos sean consistentes y evita problemas de concurrencia en entornos distribuidos.

Distribución: Los RDD están divididos en particiones que se distribuyen a través de un clúster, lo que permite el procesamiento paralelo. Cada partición puede ser procesada en un nodo diferente del clúster, optimizando así el rendimiento.

Resiliencia: Si una partición se pierde debido a un fallo, Spark puede regenerarla automáticamente a partir de otras particiones. Esto hace que los RDD sean tolerantes a fallos.

Se crean paralelizando o cargando directamente el tipo de dato.

**Operaciones sobre RDD**

Los RDD permiten realizar dos tipos principales de operaciones:

**Transformaciones**: Estas son operaciones que crean un nuevo RDD a partir de uno existente, como **map, filter y flatMap**. Las transformaciones son perezosas, lo que significa que no se ejecutan hasta que se realiza una acción sobre el RDD.

**Acciones**: Estas operaciones devuelven un valor al programa o envían datos a un sistema externo. Ejemplos incluyen **count, collect y saveAsTextFile**. Las acciones provocan la ejecución del flujo de trabajo definido por las transformaciones.

**Ciclo de Vida del RDD**

**El ciclo de vida** de un RDD incluye:

Generación: Creación inicial del RDD.

Transformaciones: Aplicación de transformaciones para generar nuevos RDD.

Caché: Almacenamiento en memoria para acceso rápido.

Acciones: Ejecución final que produce resultados visibles o exporta datos26.

En **resumen**, los RDD son esenciales para el manejo eficiente y escalable de grandes conjuntos de datos en aplicaciones de Big Data, permitiendo a los desarrolladores realizar operaciones complejas con facilidad y resiliencia frente a fallos.

**Lazy evaluation:** Es el hecho de que no se ejecutan las funciones que hacemos con RDDs, sino hasta que se llaman con un collect o alguna otra acción.

**Cache y Persist**

En el contexto de Apache Spark y el procesamiento de datos, **cache** y **persist** son dos conceptos clave relacionados con la gestión de la memoria y el rendimiento. Ambos se utilizan para almacenar datos en memoria, pero tienen diferencias en su implementación y uso.

**Cache**

El término **cache** se refiere a la técnica de almacenar en memoria los resultados de operaciones o consultas para acelerar el acceso a esos datos en futuras operaciones. Cuando se utiliza el comando CACHE, Spark guarda los datos en la memoria para que puedan ser reutilizados sin necesidad de volver a leerlos desde el disco, lo que reduce significativamente la latencia.

Características del **Cache**:

* **Almacenamiento en Memoria**: Los datos se almacenan en la RAM, lo que permite un acceso rápido.
* **Uso Repetido**: Es beneficioso cuando se espera acceder a los mismos datos múltiples veces, ya que evita la sobrecarga de lectura desde el disco.
* **Configuración de Niveles de Almacenamiento**: Se pueden especificar diferentes niveles de almacenamiento (como solo en memoria, en disco y memoria, etc.) al usar el comando CACHE

**Persist**

Por otro lado, **persist** es un concepto más general que permite almacenar un RDD (Resilient Distributed Dataset) o DataFrame en memoria o en disco con mayor flexibilidad. Al usar persist, puedes elegir entre varios niveles de almacenamiento, lo que te permite decidir cómo y dónde se almacenan los datos

Características del Persist:

* **Flexibilidad**: Permite elegir diferentes niveles de almacenamiento (por ejemplo, solo en memoria, solo en disco, o ambos) según las necesidades específicas del trabajo
* **Control sobre la Persistencia**: Puedes mantener los datos en memoria durante un tiempo determinado o hasta que ya no sean necesarios.
* **Uso Eficiente de Recursos**: Al permitir el almacenamiento tanto en memoria como en disco, persist ayuda a gestionar mejor los recursos del clúster.

**Comparación**

| **Característica** | **Cache** | **Persist** |
| --- | --- | --- |
| Almacenamiento | Solo en memoria | Memoria y/o disco |
| Flexibilidad | Menos opciones | Más opciones |
| Uso principal | Acceso rápido a datos repetidos | Control sobre persistencia |

En resumen, tanto **cache** como **persist** son herramientas valiosas para optimizar el rendimiento en Apache Spark. La elección entre uno u otro depende de las necesidades específicas del análisis y del uso esperado de los datos.

**Otras funciones:**

Partition: particiona el RDD

Repartition: Puedes volver a hacer partición hacia arriba

Coalesce: Puedes volver a hacer partición hacia abajo

getNumPartitions(): Obtienes el número de particiones.

Recomendaciones:

* Hacer una partición por cada 100MB a 1GB de datos (depende del equipo)
* No asignar todos los núcleos de CPU como núcleos de Spark, ya que todas las tareas necesitan memoria.
* Dos o tres particiones por núcleo de Spark
* No exceder la cantidad de memoria disponible asignada a Spark (tomar en cuenta otras aplicaciones en ejecución).

**Asignación de recursos de Spark**

defaultParallelism: Devuelve el número de núcleos de Spark asociado al contexto.

\*Ver libro de collab del profe, vienen sentencias para asignar núcleos o memoria\*