**Portafolio Módulo 7: Introducción al Procesamiento Distribuido Sistemas Big Data**

El siguiente Documento consolida el trabajo realizado durante el módulo 7 en el cual se desarrolla cada uno de los ejercicios indicados en clases y los que se deben desarrollar como término de cada clase.

**M7 AE 1 Big Data:Detecta las 5V en un caso real:**

**Caso:** Transporte Urbano, aplicación que muestra recorridos, ubicación de los buses/trenes en tiempo real, permitiendo al usuario conocer la ubicación y tiempo de llegada de los recorridos:

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Descripción** | **Ejemplo concreto** | **Herramienta** |
| **Volumen** | Gran cantidad de usuarios, requiriendo información en tiempo real de la ubicación y frecuencia de los transportes. Principalmente, usuarios en paraderos. | Ubicación buses en tiempo real (GPS), ubicación de usuarios (frecuencia) | Kafta, Spark, BDnoSQL, apache, MongoDB |
| **Variedad** | Datos Estructurados  (GPS, hora, coordenada),  Datos no Estructurados  (comentarios de los usuarios) | Fuentes: múltiples recorridos, ubicación en tiempo real, usuarios y buses de diversos recorridos, comentarios de los usuarios. | Talend ETL, AWS Glue |
| **Velocidad** | Acceso en tiempo real y almacenamiento para posteriores análisis. | El usuario debe tener la información en tiempo real. | Cassandra, DynamoDB, |
| **Veracidad** | Los datos deben ser fidedignos para estar a disposición de los usuarios en forma correcta (Fallo satélite posición de GPS) | Coordenadas correctas, posibles errores en sectores con pérdida de señal o defectuosos | Talend Data Quality, AWS Glue Data Quality, Apache Atlas |
| **Valor** | Tiempos de espera usuario (calidad del servicio), eficiencia de ruta, datos de GPS (tráfico) | Reducción de tiempos de espera, precisión de llegada del transporte | Power BI / Tableau |

**Diseña una arquitectura Big Data mínima**

Se requiere obtener información en tiempo real de la ubicación de los buses y de los usuarios. Debe disponer de la información de manera permanente y evitar caídas del sistema. Se recomienda un sistema distribuido con base de datos No SQL. Se requiere procesar la información en tiempo real para ser consultada por los usuarios. Se requiere mostrar información a los usuarios mediante una APP. Se requiere mantener información histórica para consultas posteriores. Permite tomar decisiones estratégicas, como aumentar la cantidad de buses, aumentar frecuencias o modificar recorridos.Tabla

El contenido generado por IA puede ser incorrecto.

**M7 AE2 Apache Spark**

**Identifica los componentes clave de Spark en un escenario real**

**Caso: Análisis de alto flujo en carreteras.**

Analizar el flujo del tráfico en tiempo real para poder implementar alternativas o sugerencias a los conductores mediante una APP (tipo WAZE).

1. **Componentes de SPARK que se utilizarían:**

**Spark Streaming:** Este módulo será el encargado de la **ingesta continua de datos**. Capturará el flujo de datos de los sensores de tráfico y los dividirá en micro-lotes para su procesamiento. Esto permite un análisis casi en tiempo real de las condiciones del tráfico.

**Spark SQL:** Se utilizará para estructurar los datos crudos en un formato tabular (DataFrame o Dataset), lo que facilita la realización de consultas complejas. Con Spark SQL, se pueden calcular métricas clave como la velocidad promedio, la densidad del tráfico y el número de vehículos por segmento de carretera.

**Spark MLlib:** Para tareas avanzadas como la **predicción de congestiones**, se aplicarán modelos de aprendizaje automático. Por ejemplo, un modelo de series temporales podría predecir la densidad del tráfico en los próximos 15 minutos basándose en datos históricos y en tiempo real, activando alertas preventivas.

**Spark Streaming:** Para procesamiento de flujos de datos robustos y de alto nivel. Permite el uso de estas APIs de DataFrames y SQL para procesar datos de streaming, lo que simplifica enormemente el código y la lógica de la aplicación.

1. **Representación Gráfica del Procesamiento en Clúster**

**Fuentes de Datos (Sensores, GPS):** Generan un flujo constante de datos de tráfico que son enviados a un sistema de mensajería como **Apache Kafka**.

**Driver:** Funciona como el orquestador principal de la aplicación. Se conecta a la fuente de streaming (Kafka) y define el plan de ejecución de las consultas. Coordina las tareas y la distribución de las particiones de datos.

**Cluster Manager (YARN, Mesos):** Se encarga de la asignación de recursos. Proporciona los recursos de CPU y memoria necesarios a los nodos de trabajo (Executors) para procesar el flujo de datos.

**Executors:** Son los nodos de trabajo que realizan el procesamiento paralelo de los datos. Cada **Executor** procesa una o más **particiones** de datos que contienen un subconjunto de los micro-lotes. Aquí se ejecutarán las transformaciones de Spark SQL y los modelos de ML.

**Resultados:** Los datos procesados (alertas de congestión, velocidad promedio, etc.) se pueden escribir en una base de datos para su visualización o en otra cola de mensajería para su consumo por otras aplicaciones en tiempo real.

**4. Justifica porqué se usaría SPARK y no otra herramienta:**

**Apache Spark** es más rápido que Hadoop MapReduce, ya que procesa los datos **en memoria**, Hadoop los procesa en disco, lo que reduce la latencia. Esta velocidad es crucial para la toma de decisiones en tiempo real, como la gestión de rutas de emergencia.

**Ecosistema Unificado:** Spark proporciona una suite completa para el análisis de datos de extremo a extremo, desde la ingesta (Spark Streaming) hasta el análisis exploratorio (Spark SQL), pasando por la construcción de modelos (MLlib). Esto elimina la necesidad de integrar múltiples tecnologías, simplificando el desarrollo y el mantenimiento.

**Flexibilidad:** El modelo de **DataFrames** permite a los desarrolladores escribir código de alto nivel y declarar lo que quieren hacer, dejando que el motor de optimización de Spark (Catalyst) determine la forma más eficiente de ejecutarlo.

**Escalabilidad Horizontal:** Spark es intrínsecamente escalable. El sistema puede manejar un aumento masivo en el volumen de datos de tráfico simplemente añadiendo más nodos de bajo costo al clúster, sin sacrificar el rendimiento.

**Comparativa entre Spark , Hadoop y Flink:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Criterio Clave** | **Apache Spark** | **Hadoop MapReduce** | **Apache Flink** |
| **Modelo de procesamiento** | Micro-lotes (batch) y streaming, **en memoria** | Por lotes (batch), **en disco** | Streaming nativo |
| **Uso de disco y memoria** | Principalmente en memoria, el disco se usa para desbordamiento y persistencia | Usa intensivamente el disco, lo que lo hace más lento | Principalmente en memoria, con gestión de memoria optimizada |
| **Lenguajes Soportados** | Scala Java Python y SQL | Java, Python | Scala Java Python y SQL |
| **Latencia** | Baja, gracias al procesamiento en memoria y micro-lotes | Alta, debido a la escritura en disco entre etapas | Muy baja, ideal para el procesamiento en tiempo real |
| **Casos de uso típicos** | Machine learning, SQL interactivo, análisis de datos en tiempo real y por lotes | Procesamiento de grandes volúmenes de datos por lotes (ETL) | Detección de fraudes, análisis en tiempo real, aplicaciones de eventos complejos |
| **Facilidad de Uso** | **muy fácil de usar** debido a sus API de alto nivel | **más difícil de usar** porque requiere que los desarrolladores escriban código de bajo nivel | se encuentra en un punto intermedio; |
| **Tolerancia a fallos** | Alta, usa RDDs para la recuperación de datos | Alta, mediante la replicación de datos de HDFS | Alta, con puntos de control (checkpoints) incrementales |
| **Herramienta para cada caso** | -Procesamiento por Lotes en LOG Históricos  -Modelo de aprendizaje Automático Interactivo |  | -Análisis en Tiempo Real de Redes Sociales |

**Preguntas a la clase:**

**¿Por qué Spark es más eficiente que Hadoop MapReduce en análisis iterativos?**

Spark es una aplicación más nueva, trabaja en memoria, no en disco como Hadoop, lo que mejora los tiempos de consulta.

**¿Qué ventajas trae Spark para flujos de datos en tiempo real?**

Spark Streaming divide los datos en micro-batchinig, lo que permite el trabajo en tiempo real. permite tolerancia a fallos.

**¿Qué diferencias hay entre RDD, DataFrame y Dataset? ¿Cuándo conviene usar cada uno?**

RDD (Resilient Distributed Dataset): Es la abstracción de bajo nivel original de Spark. Es una colección de elementos inmutable y particionada, que se puede procesar en paralelo. Permite un control completo sobre los datos y las transformaciones. No tiene un esquema de datos (esquema = nombre de columna, tipo de dato).

Uso ideal: Cuando necesitas un control granular y de bajo nivel sobre las transformaciones. Es útil para datos no estructurados o cuando las APIs de DataFrame o Dataset no son suficientes.

DataFrame: Es una abstracción de datos más avanzada que organiza los datos en columnas con nombre (similar a una tabla de base de datos relacional). Está optimizado con un motor de ejecución llamado Catalyst Optimizer que genera planes de consulta eficientes.

Uso ideal: Para datos estructurados o semi-estructurados (como JSON, CSV). Es la opción preferida para la mayoría de las tareas de procesamiento de datos, análisis y aprendizaje automático, ya que es más rápido y tiene una API más expresiva y orientada a la programación.

Dataset: Es una extensión del DataFrame que añade tipado fuerte y seguridad en tiempo de compilación. Combina las ventajas del DataFrame (optimización de Catalyst) con las de los RDD (tipado fuerte), lo que permite que el motor de ejecución entienda la estructura de los datos a nivel de objeto. Los Datasets son específicos para Scala y Java.

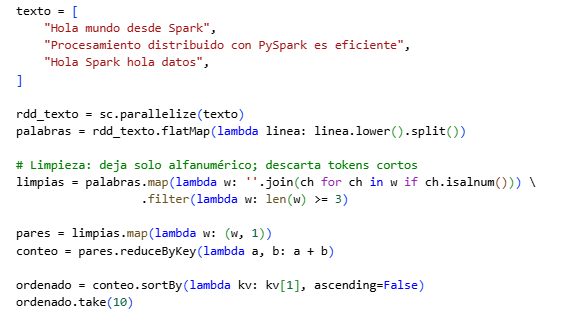
Uso ideal: Cuando la seguridad de tipos es fundamental, especialmente en aplicaciones de Scala y Java. Proporciona los beneficios de rendimiento de los DataFrames y la seguridad de tipos de los RDDs.

En resumen: se recomienda usar DataFrame para la mayoría de los casos de uso con datos estructurados. Si se necesita seguridad de tipos en Scala o Java, se debe optar por Dataset. Si se trabaja con datos no estructurados o se requiere un control de bajo nivel que no se encuentra en las otras APIs, se utilizaría RDD.

**M7 AE 3 Explora y transforma datos con RDDs**

En el ejercicio, se carga un conjunto de datos y se busca manipularlo para separar el texto en palabras y mostrar la frecuencia de cada palabra. El ejercicio está en Google Colab en el siguiente enlace:

<https://colab.research.google.com/drive/1HuOTtke7pikHHo6PucpKsX7yKyrMEkYx#scrollTo=37039684>



Se define la lista llamada **texto**.

**sc.parallelize(texto)**: convierte la lista de python en un RDD de Spark, y distribuye los elementos de la lista a través de los nodos del clúster de Spark.

**rdd\_texto.flatMap(lambda linea: linea.lower().split())**: transforma el RDD en una lista de palabras separadas, convirtiendo todo a minúsculas y usando el espacio como separador.

Map(): usa una función lambda para iterar sobre los caracteres de cada palabra, mantener solo los caracteres alfanuméricos y eliminar comas, espacios y puntos y

luego sólo considera palabras sobre 2 caracteres

Luego, **limpias.map(lambda w: (w, 1))** transforma cada palabra en un par clave valor, ejemplo (“hola”,1)

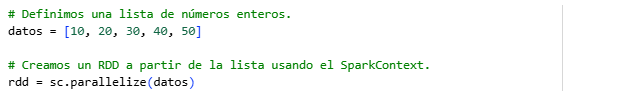
**pares.reduceByKey(lambdaa, b:a+b)** Esta es la operación central del conteo de palabras. **Agrupa los pares por su clave** (la palabra) y **suma sus valores** (los 1s). Por ejemplo, si hay tres pares ("hola", 1), ("hola", 1) y ("hola", 1), reduceByKey los combina para producir ("hola", 3).

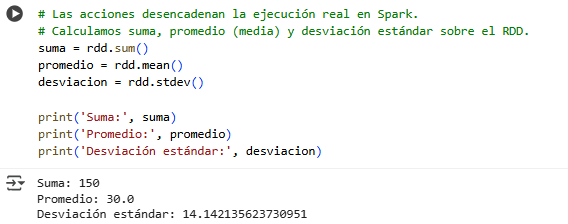
**conteo.sortBy(lambda kv: kv[1], ascending=False)** Esta línea ordena el resultado del conteo por el valor del par (cantidad de veces de cada palabra).

**ordenado.take(10)** muestra los primeros 10 elementos del RDD ordenado.

**Ejecuta un Job Spark completo con acciones y monitoreo:**

En el ejercicio se carga un RDD numérico desde una lista y se aplican transformaciones, esperando visualizar el momento en que se gatilla la acción:





**Preguntas a la Clase:**

**¿Qué diferencias hay entre un RDD y un DataFrame en cuanto a flexibilidad y rendimiento?**

Flexibilidad (RDD): Los RDDs son más flexibles que los DataFrames porque no imponen una estructura de datos estricta. Permiten trabajar con datos de cualquier tipo (no estructurados, semi-estructurados o estructurados) y ofrecen control total sobre las operaciones y particiones.

Rendimiento (DataFrame): Los DataFrames tienen un mejor rendimiento que los RDDs, especialmente para datos estructurados.

**¿Por qué Spark no ejecuta las transformaciones hasta que se invoca una acción?**

Spark utiliza evaluación perezosa (lazy evaluation), lo que significa que las transformaciones (como map, filter, join) no se ejecutan inmediatamente cuando se definen. En su lugar, Spark crea un Gráfico Acíclico Dirigido (DAG) de las transformaciones. La ejecución real de estas transformaciones solo se desencadena cuando se invoca una acción (como count, collect, save).

La razón principal de la evaluación perezosa es la optimización. Al construir el DAG completo antes de ejecutar cualquier cosa, Spark puede analizar todas las transformaciones y optimizar el plan de ejecución para minimizar el movimiento de datos y las operaciones innecesarias. Esto mejora el rendimiento general del trabajo y permite a Spark ser más eficiente en el uso de los recursos del clúster.

**¿Qué ventajas tiene dividir los datos en particiones? ¿Cómo afecta eso al rendimiento?**

La división de datos en particiones permite a Spark procesar los datos de forma paralela en diferentes nodos del clúster, lo que es esencial para el procesamiento de big data. Además, por ser resiliente, ante eventuales problemas se reasignan las tareas.

**M7 AE 4 Procesamiento de Datos Estructurados**

**Clase 37**