Universidad Nacional de La Plata
Facultad de Informática
Conceptos y Aplicaciones de Big Data

Informe Comparativo | Procesamiento por lotes en Apache Hadoop MapReduce y Apache Spark

Chaar Norberto Ariel

Plataformas de Desarrollo:

Google Colab: https://colab.research.google.com/
Apache Spark: https://spark.apache.org/
Apache Hadoop MapReduce: https://hadoop.apache.org/

Introducción

En este documento se responde a la consigna planteada en el Trabajo Práctico 2 de la asignatura, en cuanto a explicar las diferencias en complejidad de programación, según las soluciones planteadas para las consultas 1 a 4, correspondientes a los TP1 y TP2 propuestos, las cuales fueron implementadas utilizando MapReduce y Spark, respectivamente. El objetivo es analizar las diferencias en cuanto a la complejidad de programación, destacando las ventajas y desventajas de cada enfoque.

Consulta 1: La misión más jugada

Complejidad de programación de la solución en MapReduce (TP1):

- Múltiples Jobs: Se requieren dos jobs separados, lo que implica escribir código adicional para su gestión y el manejo de archivos intermedios.
- Definición explícita de funciones: Es necesario definir funciones map, combine y reduce para cada job.
- Manejo de datos intermedios: Se debe gestionar manualmente la lectura y escritura de los resultados entre jobs.

Complejidad de programación de la solución en Spark (TP2):

- Código más conciso: Todo el proceso se realiza en pocas líneas de código.
- API de alto nivel: Spark proporciona funciones integradas que simplifican operaciones comunes como reducción y ordenamiento.
- Sin gestión de datos intermedios explícita: Spark maneja internamente los datos intermedios, reduciendo la complejidad.

Consulta 2: El héroe más elegido en cada misión

Complejidad de programación de la solución en MapReduce (TP1):

- Múltiples Jobs y funciones personalizadas: Similar a la Consulta 1, se requiere manejar varios jobs y definir funciones específicas.
- Gestión de empates: Se debe implementar lógica adicional para manejar casos donde hay empates entre héroes.

Complejidad de programación de la solución en Spark (TP2):

- Uso de funciones integradas: Spark proporciona funciones como groupBy, agg y ventanas analíticas que simplifican la implementación.
- Manejo de empates simplificado: Las funciones de agregación permiten manejar empates de forma más natural.
- Menos código y más legibilidad: La solución es más directa y requiere menos código que en MapReduce.

Consulta 3: Jugadores que usaron más de H héroes distintos

Complejidad de programación de la solución en MapReduce (TP1):

- Manejo de duplicados: Se requiere una etapa para eliminar héroes duplicados por jugador.
- Múltiples Jobs: Nuevamente, es necesario encadenar jobs y manejar datos intermedios.

Complejidad de programación de la solución en Spark (TP2):

- Funciones de alto nivel: Uso de funciones como explode y countDistinct simplifica el proceso.
- Sin necesidad de eliminar duplicados manualmente: Spark maneja esto internamente.
- Código más limpio y directo.

Consulta 4: Jugadores que participaron al menos N veces en la misión más jugada y puntaje total mayor a P

Complejidad de programación de la solución en MapReduce (TP1):

- Encadenamiento de múltiples jobs: Necesidad de pasar resultados entre jobs.
- Gestión manual de parámetros y filtrado: Se debe manejar cuidadosamente los criterios de filtrado en el código.

Complejidad de programación de la solución en Spark (TP2):

- Reutilización de resultados: Spark permite almacenar y reutilizar datos en caché.
- Operaciones de filtrado y agregación sencillas: Uso de funciones integradas como filter, groupBy, count y sum.
- Código más conciso y fácil de mantener.

Conclusiones

Por un lado, MapReduce ofrece un abanico más reducido de opciones al programador y consigo una curva más corta de aprendizaje; sin embargo la implementación de una solución es más laboriosa. En cambio en Spark, manejando sus distintas APIs, se reduce la complejidad de programación, ofreciendo variadas abstracciones de alto nivel que simplifican operaciones comunes en procesamiento de datos.

El código en Spark es más legible y conciso, lo que facilita su mantenimiento, reduciendo la probabilidad de errores. Mientras que MapReduce requiere una mayor cantidad de código boilerplate, con definiciones explícitas de funciones map, reduce y combiners, además de la gestión manual de múltiples jobs y datos intermedios.

Por último, la gestión de múltiples etapas y datos intermedios es más eficiente en Spark, ya que el motor maneja internamente las optimizaciones y el flujo de datos, con menor intervención del programador. Se destaca que Spark proporciona ventajas claras reduciendo la complejidad de programación y así aumentando la productividad, permitiendo enfocarse más en la lógica del negocio y menos en los detalles de implementación.