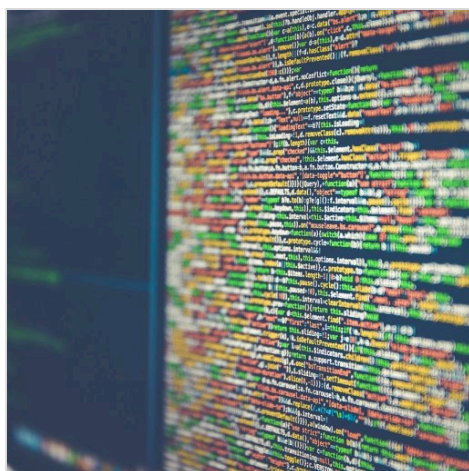


L U M B R E



Particionamiento, shuffle y datos sesgados en Spark

Grado: 6

Materia: Big Data y Procesamiento Distribuido · Ingeniería de Datos

11/10/2026

Antes de empezar

Del particionado local al barajar de datos entre nodos: cómo reconocer un straggler causado por skew y qué correctivos aplicar sin tirar de más clúster.

QUÉ APRENDERÁS

- Explicar el costo del shuffle en Spark, diagnosticar datos sesgados y aplicar técnicas (broadcast join, salted key, AQE) para equilibrar el paralelismo.

Particionamiento, shuffle y datos sesgados en Spark

VIDEO

Shuffle y particionamiento en Spark

Video explicativo (inglés): cómo funciona el shuffle en Apache Spark y cómo optimizar particiones y datos sesgados para ganar velocidad.



Escanea para verlo en Lumbre

https://www.youtube.com/watch?v=q1LtBU_ca20

El costo real está en mover, no en calcular

En Spark el cuello de botella rara vez es el CPU: es mover datos de un nodo a otro. Entender el particionado y el shuffle es lo que separa un pipeline que escala de uno que se atasca en el tercer día.

Un DataFrame se divide en **particiones**: bloques de filas que los ejecutores procesan en paralelo. El número de particiones fija el grado de paralelismo. Mientras los datos caben dentro de una partición y se procesan donde están, todo va rápido; el problema aparece cuando una operación necesita **reagrupar filas por clave** y estas viven en nodos distintos.

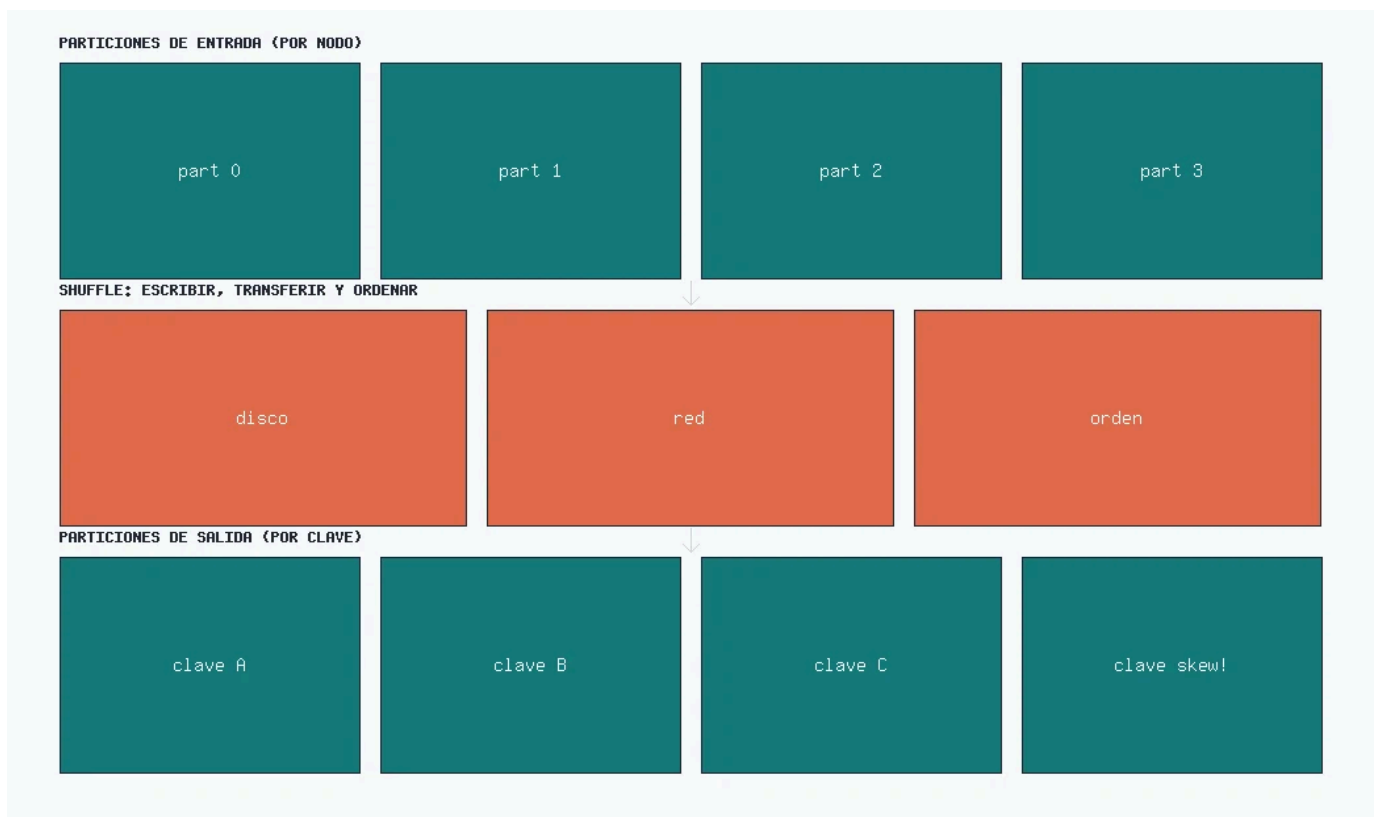
Qué es una partición

La partición es la unidad de trabajo en paralelo: una tarea por partición. Muy pocas particiones dejan núcleos ociosos; demasiadas generan un overhead de planificación que ahoga al scheduler. El valor por defecto de `spark.sql.shuffle.partitions` (200) es un punto de partida, no un acierto garantizado.

- Cada partición se procesa en una tarea, idealmente en el nodo donde ya residen sus datos.
- El objetivo es mover cómputo a los datos, no los datos al cómputo.
- El tamaño equilibrado de partición evita tareas rezagadas (stragglers).

El shuffle: cuando los datos cambian de nodo

Operaciones como `groupBy`, `join` o `repartition` rompen la localidad: para reunir todas las filas de una misma clave, Spark debe **barajar (shuffle)** datos entre ejecutores, escribiendo salidas a disco, transfiriéndolas por red y volviendo a particionarlas por clave.



El shuffle redistribuye filas por clave: la banda central es la más cara porque toca disco y red, y la de abajo muestra cómo una clave dominante concentra el trabajo.

Por qué el shuffle cuesta tanto

- Escribe y lee datos en disco en cada extremo.
- Satura la red del clúster al mover tablas completas.

- Serializa y deserializa objetos entre procesos.
- Ordena y agrupa, añadiendo CPU que ya no está donde estaban los datos.

Datos sesgados (data skew)

El **skew** ocurre cuando una clave concentra muchísimas más filas que las demás: una sola partición de salida se vuelve enorme mientras el resto termina rápido. Esa tarea lentísima es el *straggler* que define el tiempo total del job, y suele terminar en memoria insuficiente (OOM) o volcado a disco (spill).

De la señal en la UI de Spark al correctivo típico.

Síntoma	Causa probable	Remediación
Una tarea del stage dura 100x más	Clave con skew extremo (o nulas)	Salted key o AQE skew join
Executor muere por OOM	Partición de salida gigantesca	Aumentar particiones, aislar la clave
Spill a disco constante	Poca memoria para el agregado	Memorizar/optimizar, filtrar antes
Job lento sin skew claro	Demasiadas o muy pocas particiones	Ajustar shuffle.partitions, coalesce

Estrategias para evitar o aliviar el shuffle

- Broadcast join: para una tabla pequeña, repártela completa a cada nodo y elimina el shuffle del join.
- Salted keys: añade un prefijo aleatorio a la clave sesgada para dispersarla en varias particiones.
- Filtra y proyecta temprano: menos filas y columnas antes del groupBy o join.
- Cache/persist: guarda resultados intermedios que se reutilizan para no recalcular el shuffle.
- AQE (Adaptive Query Execution): en Spark 3 detecta skew y ajusta join y particiones en tiempo de ejecución.

Ajustar el paralelismo: repartition vs coalesce

repartition(n) redistribuye por completo (hace shuffle) y sirve para **aumentar** o balancear particiones. coalesce(n) reduce el número de particiones **sin** shuffle completo, ideal para juntar muchas particiones pequeñas antes de escribir. Usar repartition donde bastaba coalesce añade un shuffle costoso e innecesario.

Errores frecuentes

- Dejar las 200 particiones por defecto en datos de gigabytes o de megabytes, sin medir.
- Joins con claves nulas que caen todas en la misma partición.
- Reintentar agrandando memoria en vez de atacar el skew de la clave.

Ejemplo resuelto: join sesgado por clave nula

Un join de millones de filas se atasca en una sola tarea. Diagnóstico y corrección por etapas:

1. Inspeccionar la UI: un task del stage de join domina la duración.
2. Confirmar el skew: contar filas por clave y detectar una dominante (o nulas).
3. Separar las claves nulas y procesarlas aparte o rellenarlas con un prefijo aleatorio.
4. Aplicar salted key o habilitar AQE skew join sobre la clave grande.
5. Repetir la medición y verificar que el straggler desapareció.

¿Para qué sirve en la realidad?

La mayoría de los jobs de producción que "se ponen lentos de repente" no cambian de datos: cambian de distribución. Reconocer el shuffle y el skew permite arreglar el pipeline sin tirar de presupuesto en más clúster.

1. ¿Cuál es normalmente la operación más costosa en un job distribuido de Spark?

- ☐ El cálculo dentro de una partición que ya está en el nodo
- ☐ El shuffle: redistribuir filas entre nodos por clave
- ☐ Leer el esquema del DataFrame
- ☐ Nombrar las columnas

2. Usar coalesce(n) para reducir particiones suele ser más barato que repartition(n) porque evita un shuffle completo.

- ☐ Verdadero
- ☐ Falso

3. Cuando una sola clave concentra la mayoría de las filas y vuelve una tarea un straggler, decimos que hay sesgo de _____ (skew).

4. Clasifica cada medida según si reduce o provoca tráfico de shuffle.

Elementos:

- broadcast join de una tabla pequeña
- groupBy sobre una columna no particionada
- filtrar filas antes del join
- repartition(n) con nueva clave
- cachear un intermedio reutilizado

Reduce shuffle	. . .
Provoca shuffle	. . .

5. Une cada término con su definición.

5.	Partición	-----
5.	Shuffle	-----
5.	Straggler	-----
5.	Salted key	-----

Opciones: bloque de filas procesado en paralelo · redistribuir filas entre nodos por clave · tarea rezagada que define el tiempo total · prefijo aleatorio que dispersa una clave skew

ACTIVIDAD INTERACTIVA DISPONIBLE EN
LUMBRE

6. Shuffle y skew en Spark



Rendimiento distribuido: partición, shuffle y skew

Spark a escala

Particion

unidad de paralelismo

equilibrio y tamaño

Shuffle

groupBy y join

disco + red + orden

Skew

clave dominante

straggler y OOM

Mitigación

broadcast join

salted key

AQE y cache

El mapa mental interactivo es interactivo en la versión online.



Spark SQL: tuning de rendimiento (documentación oficial) <https://spark.apache.org/docs/latest/sql-performance-tuning.html> Referencia oficial de particionado, broadcast, AQE y configuración de shuffle.

Adaptive Query Execution en Spark 3 (Databricks) <https://www.databricks.com/blog/2020/05/29/adaptive-query-execution-speeding-up-spark-sql-at-runtime.html> Cómo Spark detecta skew y reoptimiza joins y particiones en tiempo de ejecución.

7. Tienes un job de join que antes corría en 10 minutos y ahora tarda 3 horas sin que haya crecido el volumen total. ¿Qué revisarías primero en la UI de Spark y qué hipótesis sobre la distribución de claves pondrías a prueba antes de tocar memoria o particiones?

Referencia curricular

Grado:	6
Materia:	Big Data y Procesamiento Distribuido · Ingeniería de Datos

Versión interactiva completa (con actividades, videos y autoevaluación):



<https://univ.lumbre.lat/object/cdia-bigd-004-particionamiento-shuffle-y-datos-sesgados-en-spark>

Material educativo generado desde Lumbre · lumbre.lat