UAB
Universitat Autònoma de Barcelona

Antonio Espinosa

Apache Spark

Spark: análisis de datos con MapReduce iterativo

- ¿Por qué usar Apache Spark?
- Arquitectura Spark
- Transformaciones de datos en Spark
- Ejemplos de uso

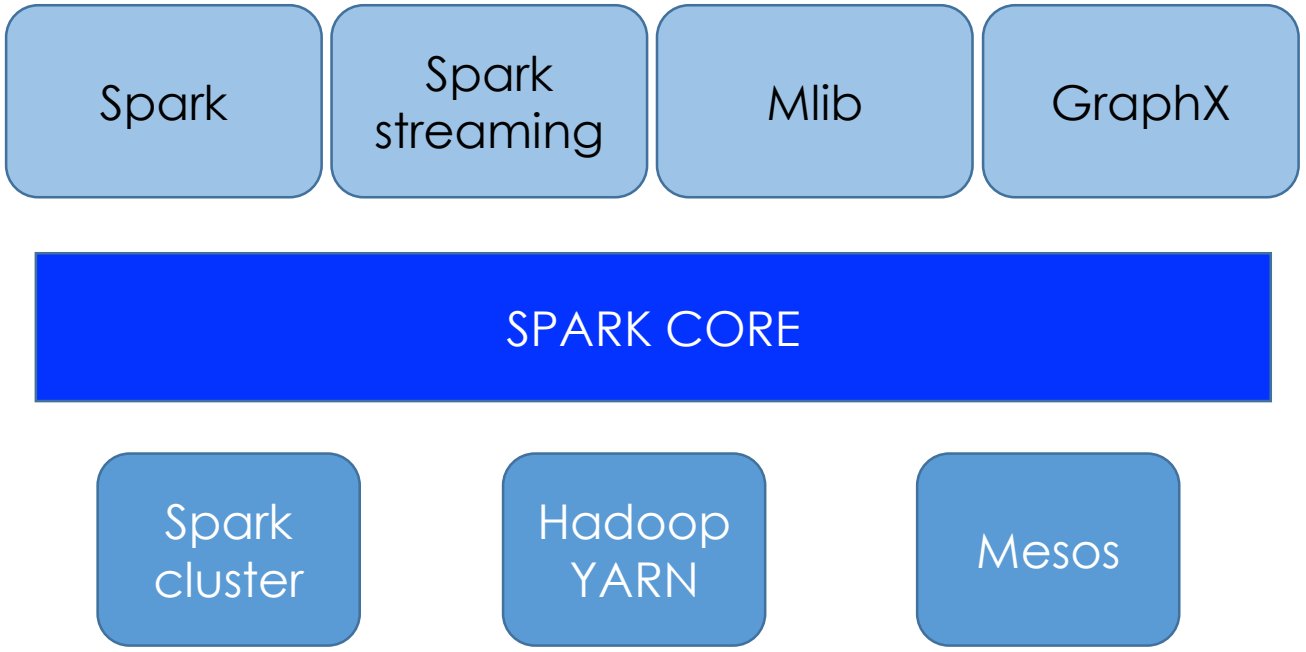
¿Por qué Apache Spark?

- El paradigma map reduce es útil para procesar grandes volúmenes de datos
- Entornos como Hadoop ofrecen un número limitado de modelos de flujo de datos
  - Muchas aplicaciones quedan fuera de este modelo simplificado
- Hay flujos de datos iterativos que pueden beneficiarse del reuso de los datos una vez éstos ya están cargados en memoria.
  - Típico caso: machine learning

Ventajas de Spark

- Facilidad de diseñar DAGs:
  - El motor de Spark permite crear cadenas de pasos en forma de DAGs
  - Permite expresar algoritmos y pipelines más complejas englobadas en la misma sesión de análisis
- Simplicidad
  - El entorno de programación se ha simplificado y es más compacto que el de Hadoop
- Versatilidad
  - Es un entorno de propósito general con diversas extensiones como Spark Streaming, GraphX para procesar grafos y Mlib, una biblioteca de machine learning

Arquitectura de Spark



Procesamiento de datos con Spark

Características de Spark

- **Almacenamiento**: proporciona flexibilidad para almacenar datos en memoria, replicada en distintos nodos o persistida en disco.
- **Multi-lenguaje**: Spark está desarrollado en Scala, pero ofrece APIs para Java, Scala, Python y R
- **Independencia** del framework: Spark ofrece soporte para YARN y Mesos como gestores de recursos
- **Terminal interactiva** (REPL): los trabajos Spark pueden lanzarse desde una aplicación o un terminal cuando necesitamos interactuar rápidamente con el conjunto de datos

Reducir operaciones de E/S

- **Resilient Distributed Datasets (RDD):** se quiere reducir el volumen de operaciones de entrada/salida y mantener el conjunto de datos en memoria
- **RDDs** son colecciones de elementos serializables. Pueden ser particionadas y distribuidas entre varios servidores
- Todas las aplicaciones crean **RDDs** de una fuente de entrada, aplica transformaciones a los **RDDs** y finalmente se almacenan los resultados de interés

Transformaciones de datos

- **Map**: aplica una función a cada elemento de un RDD para producir un nuevo RDD
- **Filter**: toma una función booleana y la aplica a cada elemento del RDD retornando un nuevo RDD conteniendo solo los elementos para los que la función ha retornado un valor cierto
- **keyBy**: retorna un grupo de valores asociados por su clave
- **Join**: suma dos parejas clave/valor por sus claves

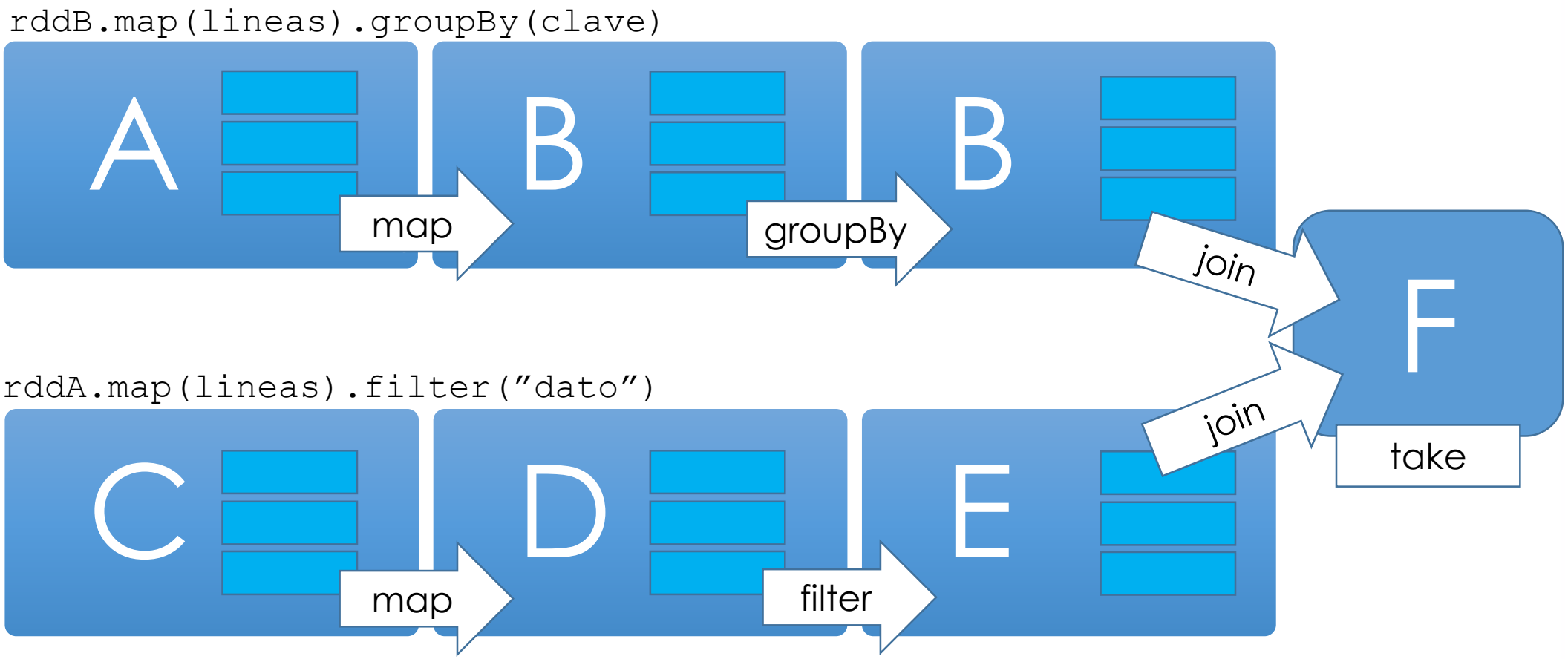
Ejemplo de trabajo Spark

```
rddA.map(lineas).filter(”dato”)

rddB.map(lineas).groupBy(clave)

rddB.join(rddA, clave).take(40)
```

Flujo de datos Spark (DAG)



Contar palabras con Spark

```
val spark = new SparkContext(m,a,[carpeta],[jars])
val datos = spark.textFile("hdfs://mi_carpeta")
val c = datos.flatMap(linea => linea.split(" ")).
                    map(palabra => (palabra, 1)).
                    reduceByKey(_ + _)
c.saveAsTextFile("hdfs://mi_carpeta_salida")
```

Cuándo usar Spark

- Spark es sencillo de usar, extensible, con amplio soporte y rápido
- En los últimos años ha desplazado a Hadoop como framework de referencia
- Se ha convertido en una de las primeras opciones a considerar para implementar aplicaciones de *machine learning* en grandes conjuntos de datos

Apache Spark

UAB
Universitat Autònoma
de Barcelona