Tema 4: Introducción a Spark

Máster Universitario en Ingeniería Informática

Extracción y Explotación de la Información

Índice

¿Qué es Spark?

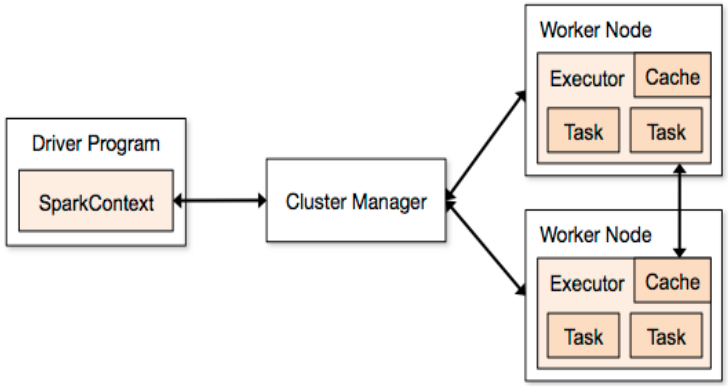
- Spark es un framework de Apache http://spark.apache.org/ para el procesamiento de datos a gran escala, como Hadoop.

- Spark aporta, frente a otros frameworks predecesores como MapReduce, su velocidad, facilidad de uso y funciones analíticas sofisticadas.

- Con respecto a la velocidad, Spark puede conseguir tiempos de latencia de milisegundos en la carga de datos masivos. MapReduce utiliza la memoria principalmente para computación. Spark la utiliza tanto para cómputo como almacenamiento.
- El entorno Spark se ejecuta en variedad de gestores de clusters, incluyendo YARN (Hadoop), MESOS así como en modo independiente (standalone).

Componentes de un cluster Spark

- En un cluster Spark hay 3 partes características:
  - El controlador del programa (proceso que ejecuta el main de la aplicación)
  - El cluster manager (como el master)
  - Los nodos trabajadores o “executors” (como slaves)



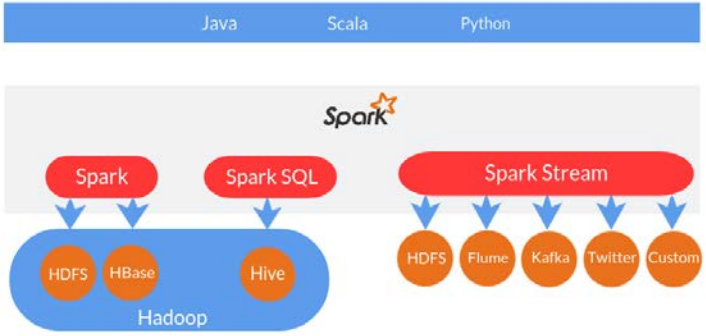
- El SparkContext es el objeto del programa principal (archivo JAR de Java o código Python/Scala) que se conecta al manager (Spark standalone, YARN,..), el cual se encarga de asignar recursos.
- Una vez conectado, el SparkContext toma los executors en los nodos del cluster (que se encargan de ejecutar las operaciones y almacenar los datos de la aplicación)
- Por último, el SparkContext envía las tareas (tasks) a los executors para ejecutarlas.

Componentes de un cluster Spark

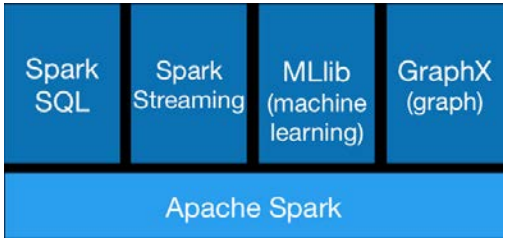
- Conviene señalar que cada aplicación representa una instancia de SparkContext.
- Y cada instancia se ejecuta en un conjunto independiente de executors (con múltiples jobs corriendo simultáneamente) en forma de JVMs, es decir, ese conjunto de executors sólo corren tareas y guardan datos de una misma aplicación.
- Spark incluye un planificador de recursos dentro de cada SparkContext/aplicación.
- De nuevo, recordar que el cluster manager donde se ejecuta Spark se encarga de la planificación de recursos entre diferentes aplicaciones (standalone mode, YARN, etc..)
  - Standalone mode: Por defecto, en este modo las aplicaciones corren en orden FIFO y cada una utilizará todos los nodos disponibles. Se puede limitar el número de nodos (spark.cores.max) así como el uso de memoria (spark.executor.memory)
  - YARN: La opción --num-executors del YARN client controla el número de executors a utilizar en el cluster, mientras --executor-memory and --executor-cores controlan los recursos per executor.
  - MESOS: En este otro gestor de clúster también es posible el control de recursos.

¿Por qué Spark?

- Proporciona computación distribuida in memory para incrementar velocidad y procesamiento de datos sobre mapreduce.
- Puede procesar datos estructurados en Hive y datos streaming (HDFS, Flume, Twitter,..)



- Es posible ir más allá de las operaciones en lotes de map-reduce:
  - Spark entiende SQL (puede usar Hive, MongoDB, CassandraDB)
  - Procesamiento de streaming
  - Machine learning (MLIB)
  - Cálculo de grafos (GraphX)
  - Integración con R (Spark R) y análisis iteractivos.



¿Spark reemplaza a Hadoop?

- Hadoop representa el procesamiento paralelo de datos tradicionalmente utilizado para correr jobs map/reduce.
- Estos jobs pueden ser largos en ejecución y tomar minutos u horas en completar.
- Spark se ha diseñado para correr sobre Hadoop y es una alternativa al modelo map/reduce que se puede emplear para procesamiento de datos stream en tiempo real y búsquedas interactivas que acaban en segundos (porque Spark es rápido, muy rápido).
- Spark usa mas RAM en lugar operaciones I/O en red y disco, lo cual lo hace más rápido que hadoop en comparación. Pero como usa más RAM necesitaría la máquina física en dedicación exclusiva para producir resultados efectivos.

¿Spark reemplaza a Hadoop?

- Hadoop sería un framework de propósito más general y Spark representaría una alternativa a Map/Reduce.
- Después de YARN y Hadoop 2.0, Spark puede correr sobre HDFS junto con otros componentes de ecosistema Hadoop.
- Por tanto, Spark se convierte en un motor de data processing que proporciona mayor capacidad al stack de Hadoop.
- La elección Hadoop/Spark dependerá, en cierta medida, de la evolución en el tiempo de estos entornos.

Desarrollos con Spark

- Es incuestionable que cuando una tecnología gusta a desarrolladores, éstos se la adoptan y empiezan a utilizarla.
- Spark se ha programado con lenguaje Scala (lenguaje funcional y orientado a objetos).
- Gracias a Scala, se pueden programar de forma concisa y fluida muchas soluciones que antes requerían muchas líneas. Sin embargo, también se puede programar en Java y, sobre todo, Python y R.
- Arquitectura Kafka+Spark+NoSQL+Scala se ha está convirtiendo en una solución demandada.
- IBM, Microsoft, Amazon, Google, ... integran sus productos de BigData con Spark (lo habitual en empresas, centros de investigación, etc.. ha sido siempre comprar cualquier tecnología de XXXX, con XXXX algunas de las empresas mencionadas).

Índice

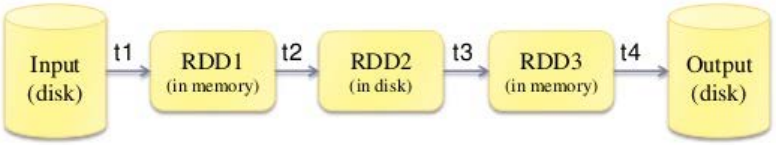
- ¿Qué es Spark?
- Elementos de Spark
  - Los datos
  - Los operadores
  - Las tareas
  - Los executors

Los datos

- En Spark las estructuras de datos se manejan a través de los Resilient Distributed Datasets (RDD), datos distribuidos resistentes:
  - Se definen como colecciones (mediante partición) de registros inmutables (sólo lectura).
  - Se almacenan en RAM o en almacenamiento persistente (HDFS,Hbase, etc..).
  - Sobre los RDD se pueden llevar a cabo distintas operaciones, que reciben el nombre de operadores.
- Los RDD se pueden almacenar en distintos niveles de forma que se puedan reutilizar de forma eficiente en diferentes operaciones en paralelo del cluster (persistencia):
  - Sólo en RAM (cache)
  - Sólo en disco
  - En memoria y disco

Los datos

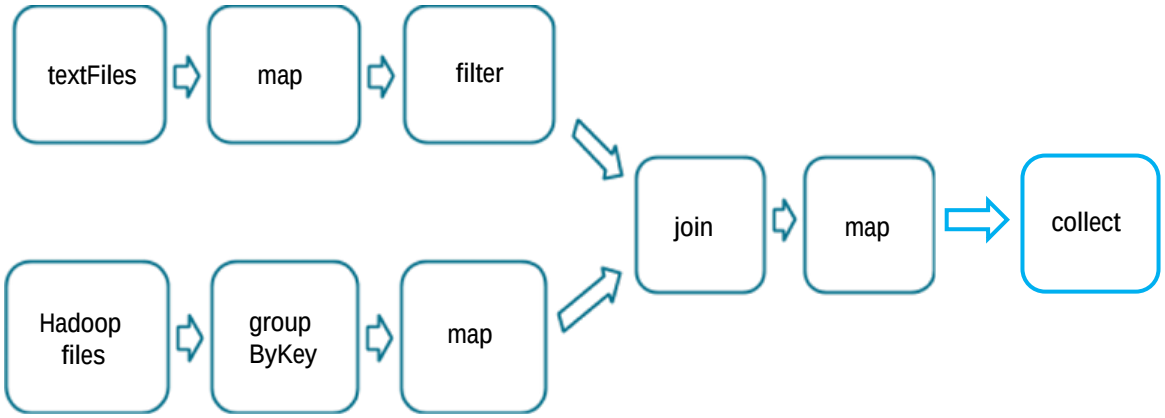
- El primer tipo de operador son las transformaciones, que consisten en convertir un RDD en otro RDD (aplicar un mapper, filtrar registros, hacer join con otros datos, etc..).



- La operación de persistencia de los datos (métodos persist() o cache()) es un tipo de transformación.
- Las transformaciones son operaciones ociosas (lazy). Spark las recuerda hasta que llegue el momento de ejecutarlas.
- El otro tipo de operador son las acciones, que consisten en hacer algo sobre los datos (aplicar una operación de reducción, almacenar en archivo, etc..)
- Las acciones NO son ociosas; en el momento de ejecutar una acción es cuando se ejecutan también las transformaciones.

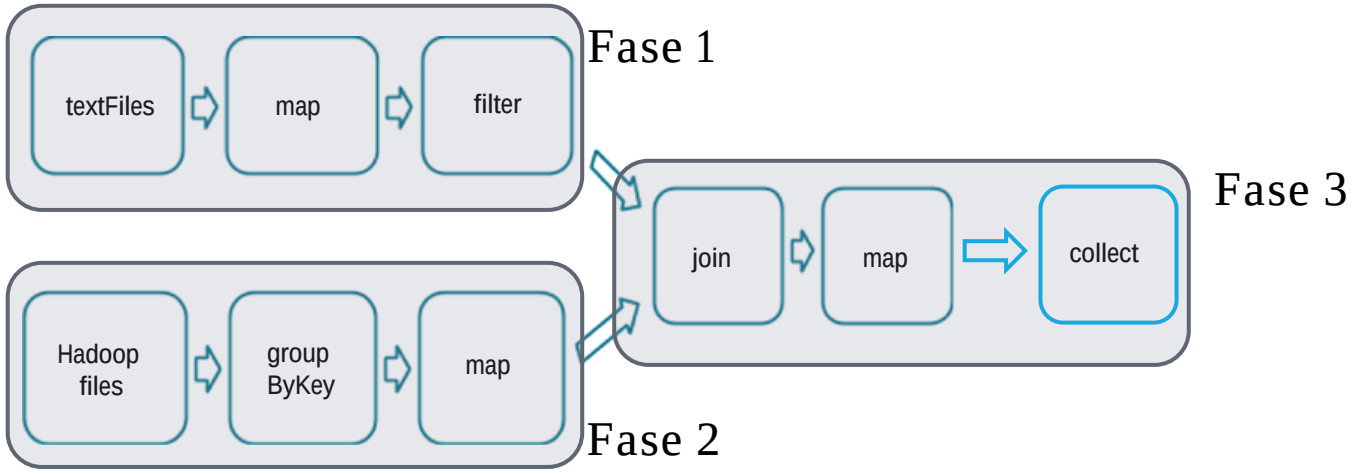
Los operadores

- Una vez creado el RDD y definidos los operadores mediante un código intérprete (Scala, Python, etc..), Spark crea un grafo DAG (grafo dirigido acíclico) para representar la computación.



Las tareas

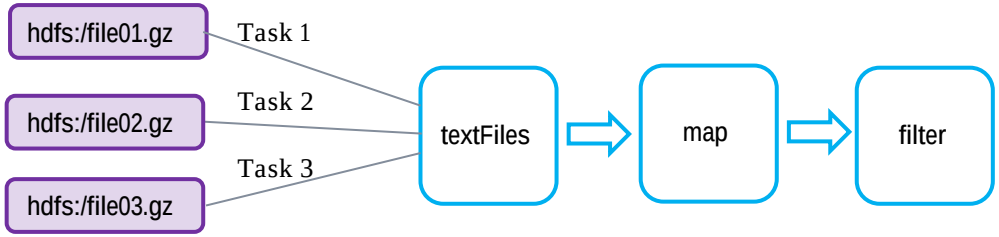
- Cuando el usuario ejecuta una acción (representada en el último nodo), el grafo pasa a un planificador que lo divide en fases (stages), las cuales se clasifican en map o reduce.



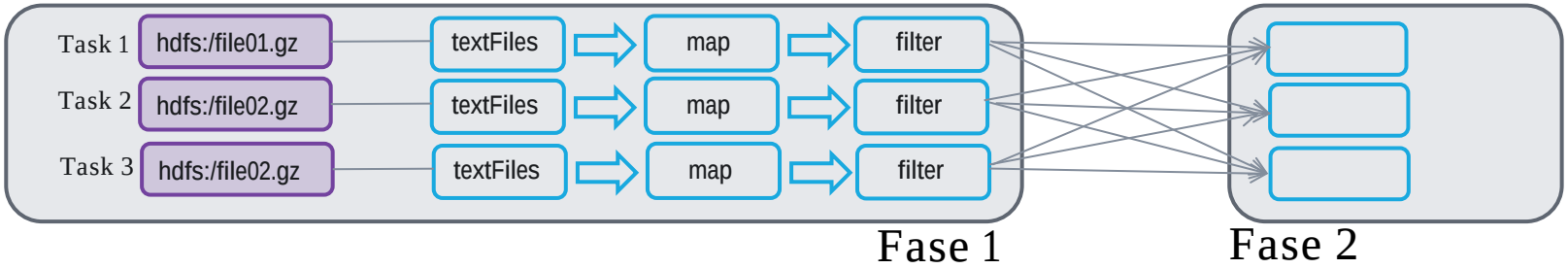
- Cada fase es creada para ejecutar varios operadores juntos en pipeline (tubería) en forma de tareas (por ejemplo, se puede crear una fase con muchas operaciones de map).
- Cada tarea se compone, por tanto, de datos + computación.

Las tareas

- Una tarea se ejecuta en una partición de los datos.



- Las tareas pasan a un planificador de tareas, quien se encarga de ejecutarlas en los executors a través del cluster manager.



- Los datos de las diferentes particiones se redistribuyen al pasar de una fase a otra (shuffle). Los shuffles se deben reducir en la medida de los posible.

Los executors

- Como se mencionó anteriormente, los executors son procesos lanzados por la aplicación en los nodos workers.
- Estos procesos ejecutan las tareas y mantienen los datos en memoria o disco a lo largo de los nodos.
- Cada aplicación tiene sus propios executors.