**TRABAJAR CON PYSPARK**

## Introducción a PySpark

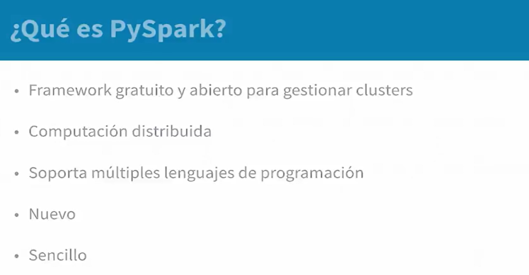
En este vídeo vamos a explicar qué es Spark, por qué es tan importante en el mundo del "big data" y cómo podemos usarlo con Python a través de PySpark. Una aclaración antes de nada: para entender de manera suficientemente rigurosa qué es el Spark se necesita dominar conceptos complejos y comprender algunos de los mecanismos internos en profundidad. Aquí solo vamos a mostrar una explicación simplificada y seguramente algunos de estos conceptos no queden 100 % claros. Esto no es un problema. El objetivo de este vídeo es ofrecer una panorámica general del Spark y mostrar sus pilares fundamentales para entender su utilidad.

Spark, primero de todo, es un "framework" gratuito y de código abierto para la computación en "clusters" de ordenadores. ¿Pero qué es un "cluster"?

**Un "cluster" es una agrupación o conglomerado de ordenadores con un mismo "hardware" común que se comportan como si fuesen un único ordenador, que se usan mayoritariamente para procesar cálculos complejos o grandes volúmenes de datos.**

Una de las principales ventajas de Spark es que combina esta capacidad de computación distribuida con un código sencillo y elegante, que además ofrece soporte a múltiples lenguajes de programación, entre ellos el Python. de la fusión entre Python y Spark nace el paquete PySpark.

Spark fue creado en la Universidad de Berkeley en California en 2014 y es considerado el primer "software" de código abierto que **hace la** **programación distribuida realmente accesible a los científicos de datos**, no solo por su gratuidad, sino también por su relativa sencillez.



Una manera sencilla de entender Spark es compararlo con su predecesor más inmediato, el MapReduce, el cual supuso una revolución total en la manera de trabajar con grandes conjuntos de datos.

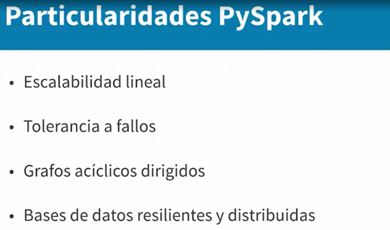
El MapReduce ofrecía un modelo de programación relativamente simple para escribir programas que se podían ejecutar paralelamente en miles de máquinas simultáneamente sin ninguna dificultad añadida.

El código para usar MapReduce en nuestro ordenador o en los servidores de una empresa de Silicon Valley es el mismo. Gracias a su arquitectura interna, el método MapReduce logra prácticamente una **relación lineal de escalabilidad**. Eso permite que, **si nuestro proyecto multiplica por dos el volumen de datos con el que trabajamos, vamos a necesitar solamente el doble de ordenadores procesando la información con el mismo código y no una reestructuración del mismo**, lo que es una ventaja enorme.

El Spark mantiene la escalabilidad lineal y la tolerancia a fallos del MapReduce, pero a su vez amplia sus puntos fuertes gracias a dos nuevas funcionalidades.

Aquí vamos a destacar dos:

* los **grafos acíclicos dirigidos**, del inglés "directed acyclic graph",
* y las **bases de datos distribuidas y resilientes**, del inglés "resilient distributed dataset".



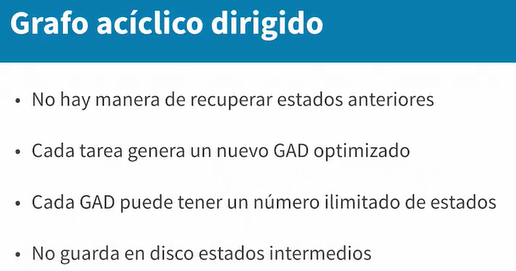
En primer lugar, el **grafo acíclico dirigido**, como su nombre indica, es un grafo dirigido que no tiene ciclos, es decir, que para cada nodo o elemento del grafo no hay ningún camino directo que comience y finalice en dicho nodo.

En otras palabras, desde una etapa de trabajo o estado concreto de nuestro proyecto vamos a realizar múltiples procesos, transformaciones de nuestros datos, acciones, etc., algunos de los cuales van a realizarse de forma paralela, pero a lo largo de todos los procesos nunca vamos a regresar al estado en el que nos encontrábamos al principio. Esta es la clave.

Spark soporta el flujo de datos acíclico. Cada tarea de Spark crea un grafo cíclico dirigido de etapas de trabajo para que se ejecuten en un determinado "cluster".

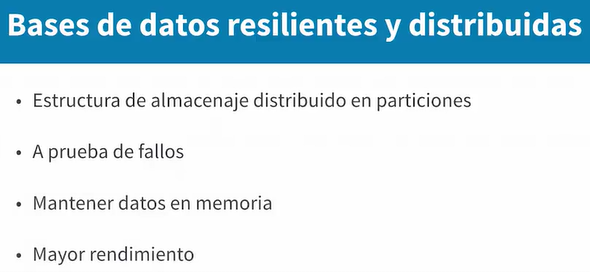
En comparación con MapReduce, el cual crea un grafo cíclico dirigido con dos estados predefinidos, el 'map' y el 'reduce', los grafos acíclicos dirigidos creados por Spark pueden tener cualquier número de etapas.

Spark, con esta estructura, es más rápido que MapReduce, por el hecho de que no tiene que escribir en disco los resultados obtenidos en las etapas intermedias del grafo. MapReduce, sin embargo, sí que debe escribir en disco los resultados entre las etapas 'map' y 'reduce'.



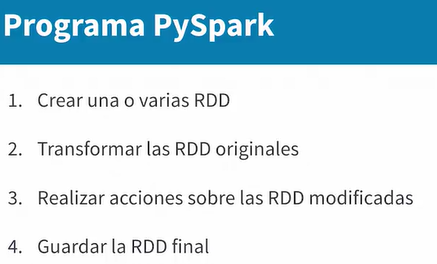
Vamos a hablar ahora de las **"resilient distributed dataset",** también llamadas **RDD**. Los "resilient distributed dataset" son una estructura de almacenaje distribuida en particiones, lo que **permite a los programadores realizar operaciones sobre grandes cantidades de datos en "clusters" de manera rápida y a prueba de fallos**.

Esta nueva estructura de datos se originó en respuesta a los problemas que se generaban al manejar datos de manera ineficiente, especialmente a la hora de ejecutar algoritmos iterativos y procesos de minería de datos. En ambos casos, mantener los datos en memoria y de manera distribuida mejora el rendimiento considerablemente.



Para entender el proceso general que se realiza a trabajar con Spark vamos a describir cómo se organizan las tareas principales y para qué sirve cada una de ellas.

**Un programa típico de Spark se estructura de la siguiente forma.**



En primer lugar se crea un "resilient distributed dataset" a partir de una variable de entorno llamada 'Context'. Esto se realiza leyendo datos de un fichero, bases de datos o cualquier otra fuente de información, ya sea estática o dinámica.

En segundo lugar, una vez creado el "resilient distributed dataset" inicial, se realizan las transformaciones deseadas para crear más objetos RDD a partir del primero. Estas transformaciones se expresan en términos de programación funcional y no eliminan el RDD original, sino que crean nuevos.

En tercer lugar se realizan acciones sobre estos RDD y posiblemente más transformaciones sobre los datos.

En último lugar, una vez llevados a cabo los procesos de acción y transformación, los objetos RDD que hayamos creado deben converger para crear el RDD final, que es nuestro objetivo de análisis. Estos pueden tener la forma de un solo valor: un vector de resultados, un nuevo "dataset", etc., en función de nuestros objetivos, así que tendremos que ser conscientes de esto a la hora de orientar todo el conjunto de transformaciones y acciones.

Es muy posible que los datos con los que se necesite trabajar estén en diferentes objetos RDD, por lo que Spark define dos tipos de opciones de transformación:

* la **transformación estrecha, "narrow transformation**",
* o la **transformación ancha, "wide transformation**".

A continuación vamos a hablar de estos dos tipos de transformación.



La **"narrow transformation"** se utiliza cuando los datos que se necesitan tratar están en la misma partición del RDD y no es necesario realizar una mezcla de dichos datos para obtenerlos todos. Algunos ejemplos son las funciones 'filter', 'sample', 'map' o 'flatmap'.

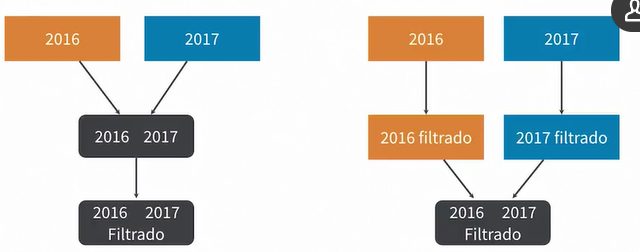
Y **l**a **"wide transformation**" se utiliza cuando la lógica de nuestro proceso necesita datos que se encuentran en diferentes particiones de un RDD y es necesario mezclar dichas particiones para agrupar los datos necesarios en un RDD determinado. Ejemplos de "wide transformation" pueden ser el 'groupByKey' o 'reduceByKey'.

¿Pero cómo combina Spark los gráficos acíclicos dirigidos y los RDD? Vamos a verlo con un ejemplo práctico.

Supongamos que tenemos dos RDD que queremos procesar conjuntamente, las ventas de los años 2017 y 2016, en dos objetos distintos. Pero por otro lado solo queremos ver las ventas de productos de estos dos años que superen los, digamos, 1000 euros, para solo trabajar con los productos más exitosos.

Hay dos maneras de plantear el problema:

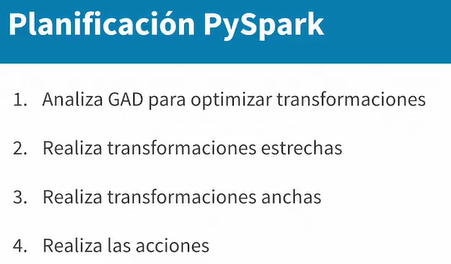
primero juntamos las dos bases de datos y luego aplicamos el filtro o, al revés, primero filtramos las dos bases de datos y luego las unimos.



¿Qué opción es mejor? Depende de qué lenguaje de programación.

En Spark la mejor opción es la segunda, ya que la operación de juntar las dos bases de datos es más costosa que la de filtrar por un valor, así que es preferible aplicar el juntado cuando ya tenemos menos datos, pero afortunadamente no tendremos que preocuparnos por esto, ya que Spark va a analizar nuestro proceso de datos y va a escoger dentro de los diferentes grafos cuál es el que ofrece una mayor velocidad de ejecución.

El plan de ejecución que Spark realiza es el siguiente:



Primero se analiza el gráfico acíclico dirigido para determinar el orden de las transformaciones.

En segundo lugar, con el fin de minimizar el mezclado de datos, primero se realizan las transformaciones "narrow" en cada RDD,

seguidamente se realizan las transformaciones "wide" a partir de los RDD sobre los que se han realizado las transformaciones "narrow" y

finalmente se ejecutan las acciones deseadas.

En este vídeo hemos visto una panorámica general de qué es el Spark, por qué es popular y una breve descripción de su funcionamiento. Los conceptos claves que debemos recordar son

* los gráficos acíclicos dirigidos,
* los "resilient distributed datasets" y
* la lógica de acciones y transformaciones.

## Sintaxis en PySpark. Qué necesitamos saber

En los siguientes minutos te voy a acompañar en los primeros pasos a realizar cuando usamos PySpark por primera vez. Vas a ver que el tipo de sintaxis es muy parecido a la del Python estándar, aunque conviene ser consciente de que las cosas funcionan de manera bastante distinta aquí, ya que no todo lo que ejecutamos se realiza al instante, sino que las transformaciones por ejemplo no se ejecutan hasta que no especifiquemos ninguna acción. Lo primero que vamos a necesitar, aparte de importar los paquetes, es generar una variable de configuración. Vamos a usarlo con esta función y vamos a tener que especificar un 'Master', aquí vamos a poner 'local', ya que como no estamos gestionando nada desde un "cluster" estamos actuando en local, y vamos a darle un nombre, puede ser cualquiera –'Mi programa', para que se vea–, y luego vamos a tener que generar un contexto que va a depender de la configuración que acabamos de generar. Es importante saber que 'Context' solo podemos generar uno, así que si ejecutamos esta misma celda varias veces en nuestro "notebook" va a generar algún error si no hemos reiniciado nuestro "kernel". Ejecutamos, vamos a ver el objeto que hemos creado y aquí nos muestra nuestra versión, el nombre del 'Master' y nuestra aplicación. Si accedemos a este enlace, nos va a redirigir a una nueva página web donde vamos a ver el tipo de trabajadores que tenemos, los RDD que hemos creado, los procesos que han funcionado, aquellos que han fallado... No es especialmente importante si lo que nos importa es sencillamente escribir código, pero es bueno tener controlado que podemos gestionar diversas variables y procesos a través de este "link". Voy a cargar un ejemplo que he generado a través de esta institución, la voy a llamar 'lines', 'textfile ("ejemplo.txt")' y voy a ejecutarlo para que veas qué tipo de objeto es. Como puedes comprobar, Spark no es demasiado atractivo a la vista en cuanto a la gestión del "output". Aquí he cargado un archivo de texto llano, que lo que es sencillamente es la entrada de Apache Spark de la Wikipedia, que la he pegado en un archivo, lo he guardado y ahora vamos a procesarlo para ver qué tipo de procesos podemos realizar con Spark. Como podemos ver aquí, si ejecutamos 'lines' no obtenemos una cabecera como podríamos obtener en 'pandas', por ejemplo, sino que lo que estamos obteniendo es que hemos generado un RDD. Vamos a explorar cuantas filas tiene 'lines' y aquí nos muestra el número de entradas que estamos gestionando, cada una de las filas cuando he copiado y pegado este archivo. Podemos pedirle que nos muestre la primera, todo esto que está ejecutando aquí son acciones y nos muestra el título de la página que he copiado. Te animo a que, una vez terminado este vídeo, copies y pegues alguna entrada de la Wikipedia en un archivo de texto y lo explores un poco usando este tipo de comandos para ver cómo funcionan de una manera más práctica. También podemos realizar alguna transformación. Por ejemplo, vamos a crear un nuevo RDD y le vamos a decir que esto es un remuestreo del 10 % de los casos que tenemos en el original sin reemplazo. Ejecutamos esto y hemos visto que esto es instantáneo. ¿Por qué es instantáneo? Porque no ha ejecutado nada. Esto ha quedado pendiente de que ejecutemos alguna acción. Vamos a hacerlo y cuando lo hagamos se va a ejecutar esta transformación. Vamos a crear 'lines2', 'first' y vamos a ver qué ha cambiado. Aquí nos está mostrando otra fila de nuestro archivo de texto.