

0 Ejercicio: Programa para contar palabras	2
1 ¿Qué es Spark?	3
1.1. Spark core API	5
1.2 Spark Components	9
1.3. Comparativa Spark & Hadoop	11
1.4. Arquitectura de la computación en cluster	19
1.5. Evaluación perezosa	22
2 RDD / Data Sets / Data frames (v1.1 actualizado)	22
RDD	25
Data Frame	27
Data Set	28
3. Map Reduce	30
4 Spark Streaming	35

0 Ejercicio: Programa para contar palabras

Objetivo:

Desarrollar un programa en Python sin usar las APIs de Spark, para posteriormente compararlo con el mismo programa realizado con Spark.

Descargar el fichero del quijote.

```
wget  
https://gramatica.usc.es/~gamallo/aulas/lingcomputacional/corpus/quijote-es.txt
```

Realizar un programa que cuente el número de veces que aparece cada palabra en el libro. Listar las “palabras-numero de veces” que aparecen en el libro de manera descendiente.

```
en      27800  
un      12000  
hidalgo  76  
caballero 60  
gigantes 50  
....
```

1 ¿Qué es Spark?

Apache Spark (<https://spark.apache.org/docs/latest/>) es un sistema de computación/ procesamiento en clúster de código abierto que permite procesar tanto tareas batch/lotas como tareas en streaming. El procesamiento batch se refiere a procesar grandes cantidades de información que se han almacenado/recuperado previamente. Mientras que el procesamiento en streaming significa el procesar datos tal y como van llegando al sistema (casi en tiempo real).



Ref: <https://data-flair.training/blogs/spark-tutorial/>



Hadoop

- https://fp.uoc.fje.edu/blog/que-es-hadoop-y-para-que-se-aplica-al-big-data/?utm_medium=cpc&utm_source=googlemax&utm_campaign=cap_fp_es&utm_keyword=&gclid=CjwKCAiA866PBhAYEiwANkIneKenuGDIC75s8qkoOa0PzgN1p_Al2HTsrkTAIHzzzCNN3SV1wMJTtxoCKSMQAvD_BwE
- https://es.wikipedia.org/wiki/Apache_Hadoop

Hadoop es una forma de almacenamiento masivo para cualquier tipo de datos que tiene la capacidad de operar tareas de forma casi ilimitada y con un enorme poder de procesamiento. Hadoop es capaz de almacenar y procesar grandes cantidades de datos de cualquier tipo, de manera distribuida .

¿Por qué es importante Hadoop?

- Capacidad de almacenar y procesar enormes cantidades de cualquier tipo de datos, al instante. Con el incremento constante de los volúmenes y variedades de datos, en especial provenientes de medios sociales y la Internet de las Cosas (IoT), ésta es una consideración importante.
- Poder de cómputo. El modelo de cómputo distribuido de Hadoop procesa big data a gran velocidad. Cuantos más nodos de cómputo utiliza usted, mayor poder de procesamiento tiene.
- Tolerancia a fallos. El procesamiento de datos y aplicaciones está protegido contra fallos del hardware. Si falla un nodo, los trabajos son redirigidos automáticamente a otros nodos para asegurarse de que no falle el procesamiento distribuido. Se almacenan múltiples copias de todos los datos de manera automática.
- Flexibilidad. A diferencia de las bases de datos relacionales, no tiene que procesar previamente los datos antes de almacenarlos. Puede almacenar tantos datos como desee y decidir cómo utilizarlos más tarde. Eso incluye datos no estructurados como texto, imágenes y videos.
- Bajo costo. La estructura de código abierto es gratuita y emplea hardware comercial para almacenar grandes cantidades de datos.
- Escalabilidad. Puede hacer crecer fácilmente su sistema para que procese más datos son sólo agregar nodos. Se requiere poca administración.

El Hadoop Distributed File System (HDFS) es un sistema de archivos distribuido, escalable y portátil escrito en Java para el framework Hadoop.

Spark Puede acceder a datos de HDFS, Cassandra, HBase, Hive, Tachyon y cualquier fuente de datos de Hadoop. Sin embargo, Spark es independiente de Hadoop, ya que tiene su propio gestor de clusters (dispone de varios tipos de gestor Standalone, YARN y Mesos). Básicamente, spark corre sobre un cluster hadoop y lo usa para almacenar datos.

Debido al cambio de contexto que está sufriendo la ciencia de datos, el nuevo enfoque de Apache Spark pasando del procesamiento en batch al streaming, permitiendo la exploración de datos ad hoc, así como la aplicación más sencilla de algoritmos de machine learning, es el factor de diferenciación básico de este proyecto.

Procesamiento de datos

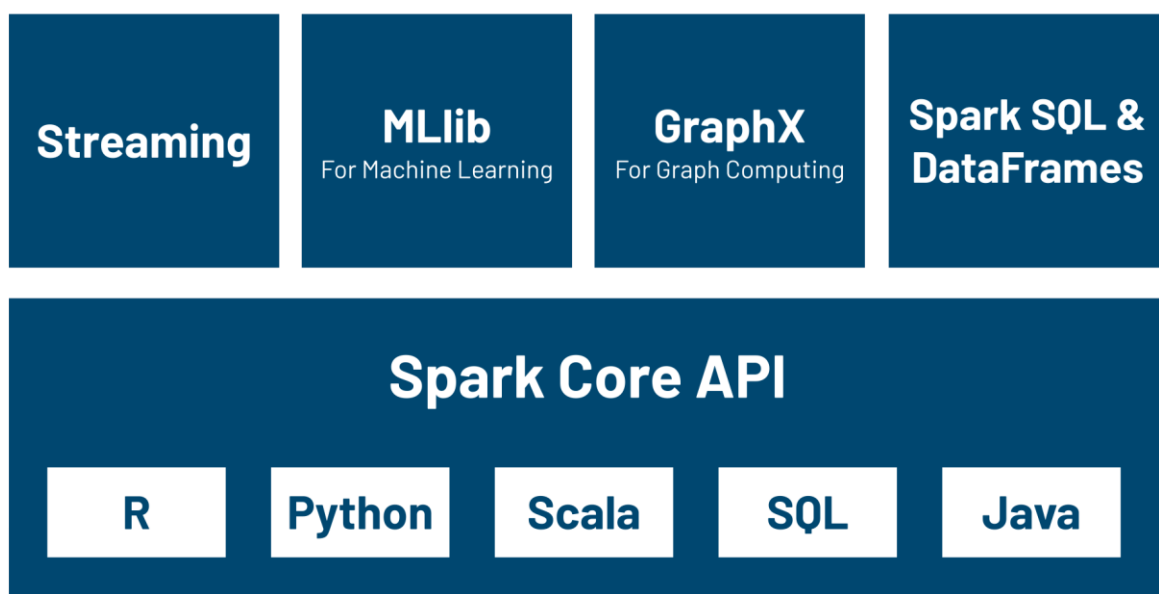
Antes, no había ningún motor de computación de propósito general en la industria, ya que

- Para realizar el procesamiento por lotes, utilizamos Hadoop MapReduce.
- Para realizar el procesamiento de flujos, utilizamos Apache Storm / S4.
- Para el procesamiento interactivo, utilizamos Apache Impala / Apache Tez.
- Para realizar el procesamiento de gráficos, utilizamos Neo4j / Apache Giraph.

Apache Storm es un motor de procesamiento de flujos de datos en tiempo real. Mientras que Apache Spark es un motor de computación de propósito general, e incluye Spark Streaming para manejar datos en streaming (casi en tiempo real) <https://data-flair.training/blogs/apache-storm-vs-spark-streaming/>

Por lo tanto, no había ningún motor potente en la industria que pudiera procesar los datos tanto en tiempo real como en modo batch. Además, existía el requisito de que un motor pueda responder en instantáneamente (casi en tiempo real) y realizar el procesamiento en memoria.

Apache Spark, ofrece procesamiento de flujos en tiempo real, procesamiento interactivo, procesamiento de gráficos, procesamiento en memoria así como procesamiento por lotes. Incluso con una velocidad muy rápida, facilidad de uso e interfaz estándar.



1.1. Spark core API

Spark está escrito en Scala pero proporciona una API para distintos lenguajes : Scala, Java, Python (PySpark) y R

Pyspark

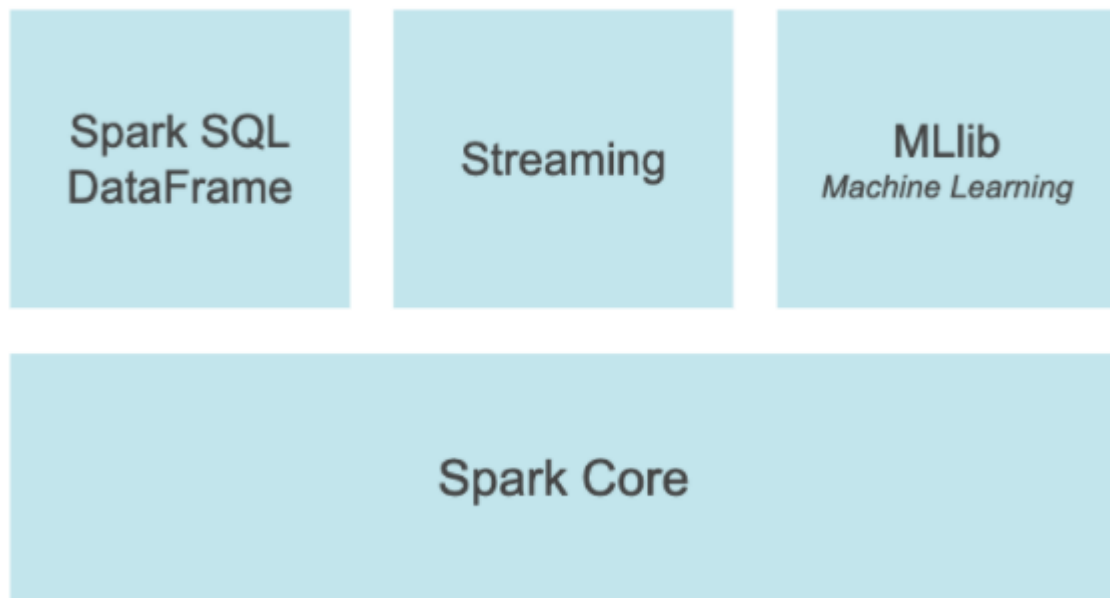
- <https://sparkbyexamples.com/pyspark-tutorial/>

PySpark es una librería de Spark escrita en Python para ejecutar aplicaciones de Python usando las capacidades de Apache Spark, usando PySpark podemos ejecutar aplicaciones paralelamente en el cluster distribuido (múltiples nodos). En otras palabras, PySpark es una API de Python para Apache Spark.



Spark está escrito en Scala y más tarde debido a su adaptación a la industria su API PySpark se liberó para Python usando Py4J. Py4J es una librería de Java que se integra dentro de PySpark y permite a python interactuar dinámicamente con los objetos de la JVM, de ahí que para ejecutar PySpark también se necesite tener instalado Java junto con Python, y Apache Spark.

Si trabajamos con pyspark debemos saber que algunas funcionalidades no están disponibles. Pyspark no soporta la API GraphX y tampoco soporta los data sets. A continuación se muestran las funcionalidades que soporta pyspark



SCALA

Se trata de un lenguaje de programación con recorrido ya que tiene dos décadas en el mercado. Además, **Scalable language** (Scala), es un lenguaje híbrido entre **programación orientada a objetos y programación funcional**. Por lo que, al tener las ventajas de uno y otro, es un lenguaje bastante funcional y práctico.

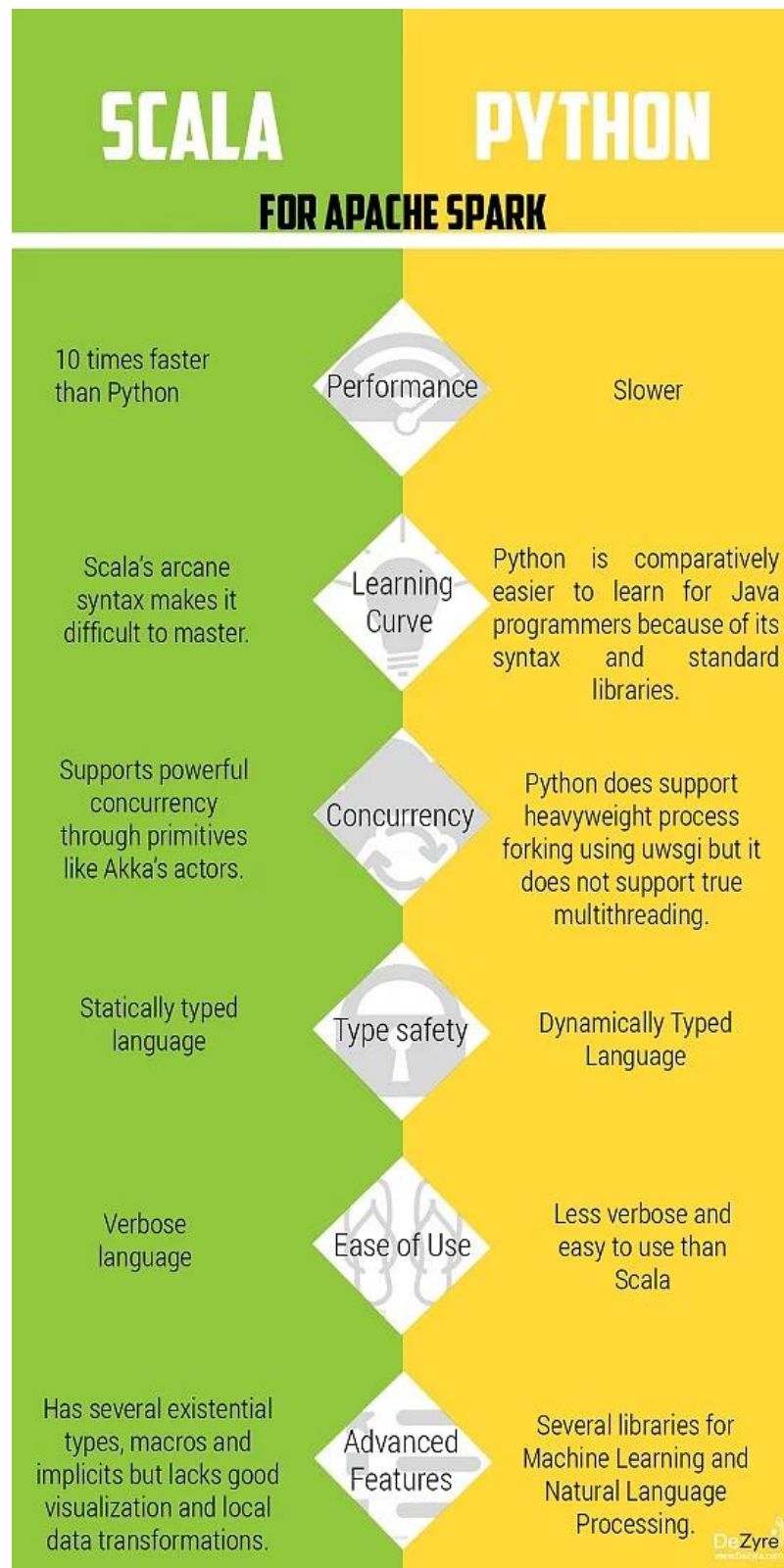
Tiene menos código para realizar algunas funciones en comparación con otros lenguajes. Esto es de utilidad debido a que se puede reducir el código a la mínima expresión y así leerlo más rápido para corregir posibles problemas.

Además, es compatible con la máquina virtual de Java, esto significa que podrás reusar librerías de Java en tus aplicaciones Scala, tendrás compatibilidad con el código en Java y te podrás beneficiar de una comunidad consolidada en el panorama de la programación.

Comparativa entre scala y python

<https://www.projectpro.io/article/scala-vs-python-for-apache-spark/213#toc-1>

<https://docs.scala-lang.org/scala3/book/scala-for-python-devs.html>



1.2 Spark Components

Streaming

Ref: <https://bigdatadummy.wordpress.com/2017/05/12/spark-streaming/>

Spark **Streaming** es un componente de **Spark** que permite el procesamiento escalable, de alto rendimiento y con tolerancia a fallos de flujos de datos en **streaming**



Como idea general se podría decir que Spark Streaming toma un flujo de datos continuo, lo convierte en un flujo discreto denominado DStream, para que el core de Spark lo pueda procesar



Dstream o stream discreto: no es más que una abstracción proporcionada por Spark Streaming que representa a una secuencia de RDDs ordenados en el tiempo que cada uno de ellos guarda datos de un intervalo concreto. Con esta abstracción se consigue que el core lo analice sin enterarse de que está procesando un flujo de datos, ya que el trabajo de crear y coordinar los RDDs es realizado por Spark Streaming.



Spark Streaming no opera en base a flujos continuos sino a micro-batches que tienen un tiempo de intervalo entre ellos (típicamente de menos de 5 segundos). Es importante que entendamos las consecuencias que esto puede tener. Por un lado, se puede configurar y reducir el intervalo a menos de un segundo, lo cual nos daría un desempeño casi de tiempo real, pero con un alto costo en recursos de procesamiento. Adicionalmente, un argumento en contra del esquema de micro-batches es que puede ser que los datos no se reciban en el orden exacto en el que sucedieron. Esto puede o no ser relevante dependiendo de la aplicación específica. Por ejemplo, en un timeline de Twitter tal vez no sea indispensable que los tweets sean procesados exactamente en el mismo orden en el que fueron generados.

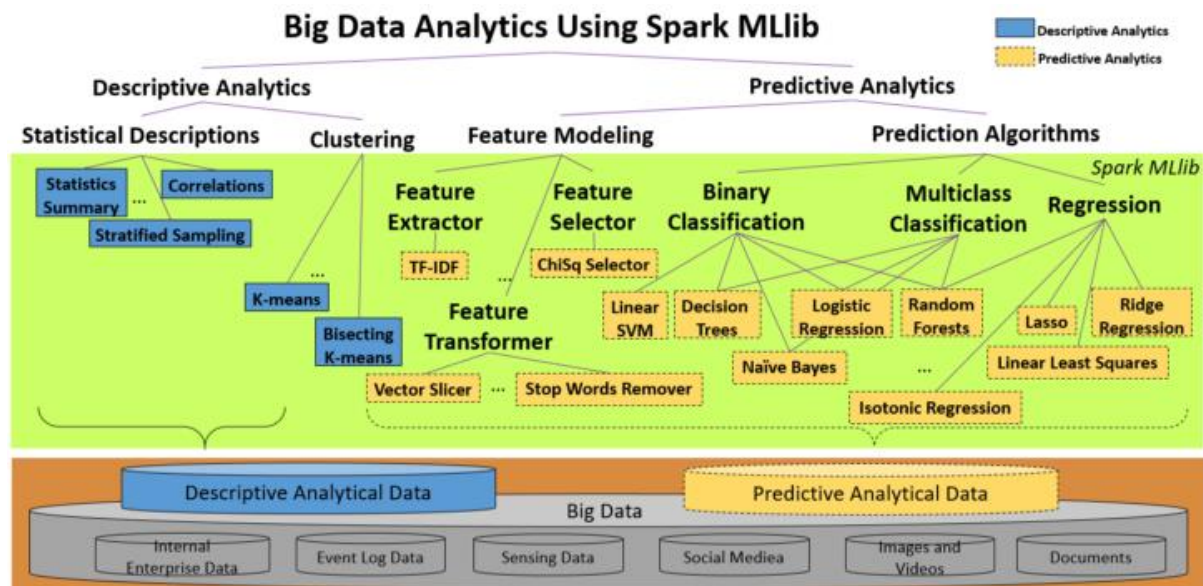
En spark streaming se pueden usar ventanas temporales para determinar los datos a procesar (<https://riptutorial.com/Download/apache-spark-es.pdf> pag 16- 22)

MLlib

Spark MLlib es la librería de Machine Learning (ML) de Apache Spark. El framework de computación distribuida que incorpora esta librería permite hacer uso de una serie de algoritmos de Machine Learning. ... La API principal o Spark ML, basada en DataFrames y que es esta dentro del paquete ML

Spark MLlib aporta algoritmos tanto de aprendizaje supervisado como no supervisado que ofrecen soluciones a las tres técnicas más utilizadas en el mundo del Machine Learning:

- **Clasificación y regresión:** Algoritmos supervisados que clasifican o predicen valores: Multilayer perceptron classifier, Decision trees, Random forest.
- **Clusterización:** Algoritmos no supervisados que agrupan datos en diferentes clusters en función de la semejanza entre ellos: K-means, LDA y GMM
- **Filtrado colaborativo:** Técnicas de recomendación basado en datos de preferencias de los usuarios: ALS



GraphX

GraphX es la API de Spark para grafos y computación paralela de grafos. Incluye una creciente colección de algoritmos de grafos y constructores para simplificar las tareas de análisis de grafos. GraphX amplía el RDD de Spark con un RDD para grafos

Spark SQL

Spark SQL es un módulo de Apache Spark para el procesamiento de datos estructurados distribuidos. ... Conceptualmente es equivalente a una tabla en una base de datos relacional. Los DataFrames en Spark tienen las mismas capacidades que los RDDs, como por ejemplo su inmutabilidad, residen en memoria, tolerantes a fallos, computación distribuida. Las consultas usan una sintaxis parecida a SQL.

```
spark.sql("SELECT name FROM parquetFile WHERE age >= 13 AND age <= 19")
```



1.3. Comparativa Spark & Hadoop

Ref: <https://data-flair.training/blogs/spark-vs-hadoop-mapreduce/>

Hadoop MapReduce lee y escribe datos desde el disco, eso repercute en su rendimiento.

Spark extiende Hadoop Map Reduce, incluyendo consultas iterativas en memoria y procesamiento en streaming. Spark es 100 veces más rápido que Hadoop y 10 veces

mas rápido para acceder a los datos desde disco. Podemos decir que Spark es un Hadoop mejorado.

Spark  vs  Hadoop MapReduce		
Factors	Spark	Hadoop MapReduce
Speed	100x times than MapReduce	Faster than traditional system
Written In	Scala	Java
Data Processing	Batch / real-time / iterative / interactive / graph	Batch processing
Ease of Use	Compact & easier than Hadoop	Complex & lengthy
Caching	Caches the data in-memory & enhances the system performance	Doesn't support caching of data

- Apache Spark - Es un framework de big data de código abierto. Proporciona un motor de procesamiento de datos más rápido y de propósito general. Spark está diseñado básicamente para la computación rápida. También cubre una amplia gama de cargas de trabajo, por ejemplo, por lotes, interactivo, iterativo y streaming.
- Hadoop MapReduce - También es un marco de trabajo de código abierto para escribir aplicaciones. También procesa datos estructurados y no estructurados que se almacenan en HDFS. Hadoop MapReduce está diseñado para procesar un gran volumen de datos en un clúster de hardware básico. MapReduce puede procesar datos en modo batch.

Velocidad

- Apache Spark - Spark es una herramienta de computación en clúster muy rápida. Apache Spark ejecuta aplicaciones hasta 100 veces más rápido en memoria y 10 veces más rápido en disco que Hadoop. Debido a la reducción del número del ciclo de lectura/escritura en disco y el almacenamiento de datos intermedios en memoria Spark lo hace posible.
- Hadoop MapReduce - MapReduce lee y escribe desde el disco, como resultado, ralentiza la velocidad de procesamiento.

Dificultad

- Apache Spark - Spark es fácil de programar ya que dispone de operadores de alto nivel con RDD - Resilient Distributed Dataset.

- Hadoop MapReduce - En MapReduce, los desarrolladores tienen que codificar a mano todas y cada una de las operaciones, lo que hace que sea muy difícil de trabajar.

Fácil de gestionar

- Apache Spark - Spark es capaz de realizar operaciones por lotes/Batch, interactivas y de Machine Learning y Streaming en el mismo cluster. Como resultado, hace que sea un motor de análisis de datos completo. Por lo tanto, no es necesario gestionar diferentes componentes para cada necesidad. La instalación de Spark en un cluster será suficiente para manejar todos los requerimientos.
- Hadoop MapReduce - Como MapReduce sólo proporciona el motor de lotes/batch. Por lo tanto, dependemos de diferentes motores. Por ejemplo, Storm, Giraph, Impala, etc. para otros requisitos. Por lo tanto, es muy difícil gestionar muchos componentes.

Análisis en tiempo real

- Apache Spark - Puede procesar datos en tiempo real, es decir, datos procedentes de flujos de eventos en tiempo real a un ritmo de millones de eventos por segundo, por ejemplo, datos de Twitter o de Facebook. El punto fuerte de Spark es la capacidad de procesar flujos en directo de forma eficiente.
- Hadoop MapReduce - MapReduce falla cuando se trata de procesar datos en tiempo real, ya que fue diseñado para realizar el procesamiento por lotes de cantidades voluminosas de datos.

Latencia

- Apache Spark - Spark proporciona computación de baja latencia.
- Hadoop MapReduce - MapReduce es un marco de computación de alta latencia.

Modo interactivo

- Apache Spark - Spark puede procesar datos de forma interactiva.
- Hadoop MapReduce - MapReduce no tiene un modo interactivo.

Streaming

- Apache Spark - Spark puede procesar datos en tiempo real a través de Spark Streaming.
- Hadoop MapReduce - Con MapReduce, sólo puede procesar datos en modo batch.

Facilidad de uso

- Apache Spark - Spark es más fácil de usar. Ya que su abstracción (RDD) permite al usuario procesar los datos utilizando operadores de alto nivel. También proporciona APIs en Java, Scala, Python y R.
- Hadoop MapReduce - MapReduce es complejo. Como resultado, necesitamos manejar APIs de bajo nivel para procesar los datos, lo que requiere mucha codificación manual.

Recuperación

- Apache Spark - RDDs permite la recuperación de particiones en nodos fallidos mediante el recálculo del DAG, a la vez que soporta un estilo de recuperación más similar al de Hadoop mediante checkpointing, para reducir las dependencias de un RDDs.
- Hadoop MapReduce - MapReduce es naturalmente resistente a los fallos del sistema. Por lo tanto, es un sistema altamente tolerante a los fallos.

Planificador

- Apache Spark - Debido a la computación en memoria Spark actúa su propio programador de flujo.
- Hadoop MapReduce - MapReduce necesita un programador de trabajos externo, por ejemplo, Oozie para programar flujos complejos.

Tolerancia a fallos

- Apache Spark - Spark es tolerante a fallos. Como resultado, no hay necesidad de reiniciar la aplicación desde cero en caso de cualquier fallo.
- Hadoop MapReduce - Al igual que Apache Spark, MapReduce también es tolerante a fallos, por lo que no es necesario reiniciar la aplicación desde cero en caso de cualquier fallo.

Seguridad

- Apache Spark - Spark es un poco menos seguro en comparación con MapReduce porque sólo admite la autenticación a través de una contraseña secreta compartida.
- Hadoop MapReduce - Apache Hadoop MapReduce es más seguro debido a Kerberos y también soporta Listas de Control de Acceso (ACLs) que son un modelo de permiso de archivo tradicional.

Costo/recursos

- Apache Spark - Como Spark requiere una gran cantidad de memoria RAM para ejecutar en la memoria. Por lo tanto, aumenta el clúster, y también su coste.
- Hadoop MapReduce - MapReduce es una opción más barata disponible al compararla en términos de coste.

Lenguaje desarrollado

- Apache Spark - Spark está desarrollado en Scala.
- Hadoop MapReduce - Hadoop MapReduce está desarrollado en Java.

Categoría

- Apache Spark - Es un motor de análisis de datos. Por lo tanto, es una opción para los científicos de datos.
- Hadoop MapReduce - Es un motor de procesamiento de datos básico.

Licencia

- Apache Spark - Licencia Apache 2
- Hadoop MapReduce - Licencia Apache 2

Soporte de sistemas operativos

- Apache Spark - Spark es compatible con varias plataformas.
- Hadoop MapReduce - Hadoop MapReduce también es compatible con varias plataformas.

Lenguaje de programación

- Apache Spark - Scala, Java, Python, R, SQL.
- Hadoop MapReduce - Principalmente Java, otros lenguajes como C, C++, Ruby, Groovy, Perl, Python también son compatibles con Hadoop streaming.

Soporte de SQL

- Apache Spark - Permite al usuario ejecutar consultas SQL utilizando Spark SQL.
- Hadoop MapReduce - Permite a los usuarios ejecutar consultas SQL utilizando Apache Hive.

Escalabilidad

- Ambos son altamente escalables

Aprendizaje automático

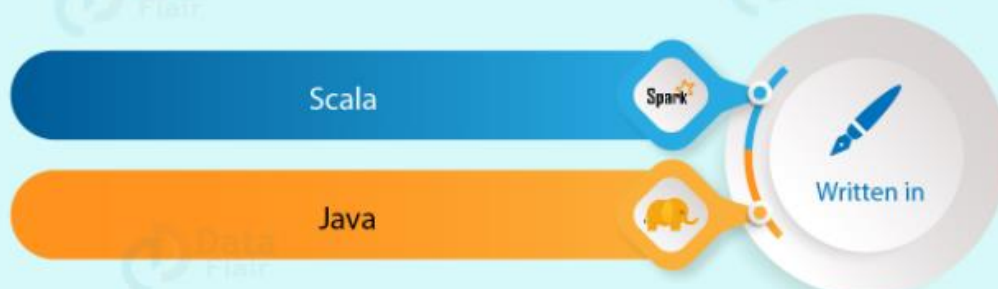
- Apache Spark - Spark tiene su propio conjunto de aprendizaje automático, es decir, MLlib.
- Hadoop MapReduce - Hadoop requiere una herramienta de machine learning por ejemplo Apache Mahout.

Caching

- Apache Spark - Spark puede almacenar en caché los datos en la memoria para futuras iteraciones. Como resultado, mejora el rendimiento del sistema.
- Hadoop MapReduce - MapReduce no puede almacenar los datos en memoria para futuras necesidades. Por lo tanto, la velocidad de procesamiento no es tan alta como la de Spark.

Requisitos de hardware

- Apache Spark - Spark necesita un hardware de nivel medio y alto.
- Hadoop MapReduce - MapReduce funciona muy bien en hardware básico.





Machine Learning

Spark

MLlib on top of Spark provides ML algo with lightening fast speed

Hadoop

Mahout on top of Hadoop provides ML algo

Cluster of 8000 nodes operational

Spark



Scalability

Hadoop has been tested on 15000 nodes

Hadoop



Latency

Spark

Provides low-latency computing

Hadoop

A high latency computing framework

Process real time data stream

Spark



Streaming

Process data only in batch mode

Hadoop



Caching

Spark

Caches the data in-memory & enhances the system performance

Hadoop

Doesn't support caching of data

Higher than Hadoop

Spark



Cost

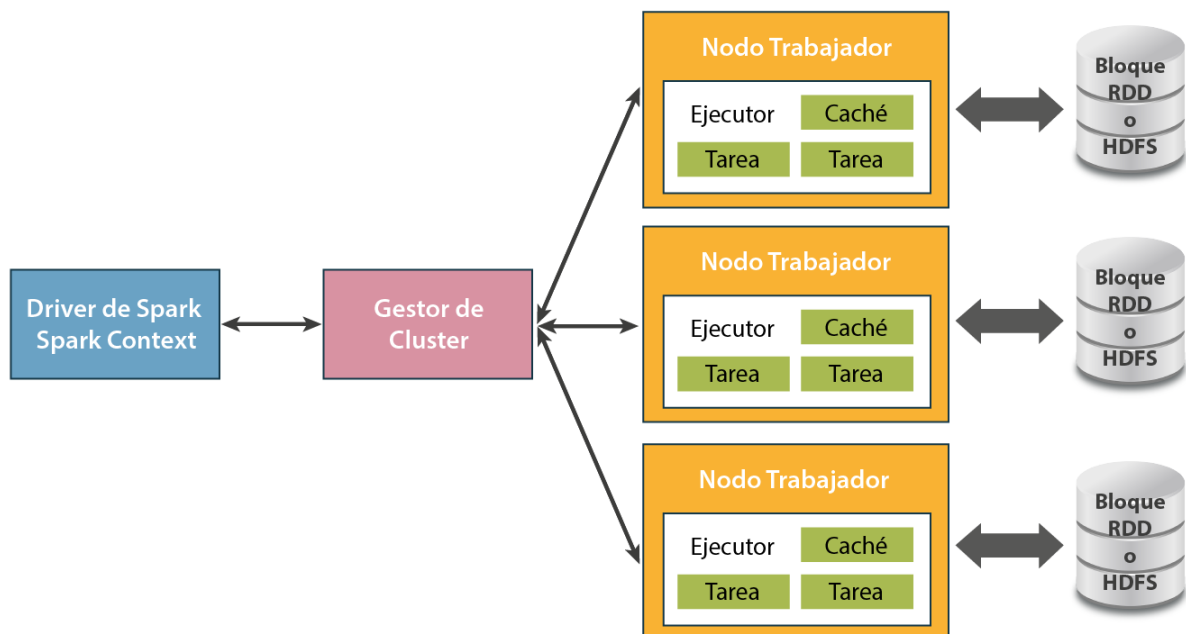
Low cost

Hadoop

1.4. Arquitectura de la computación en cluster

Ref: <https://spark.apache.org/docs/latest/cluster-overview.html>

Las aplicaciones para Spark se ejecutan como un grupo independiente de procesos en clústeres, coordinados por el objeto [SparkContext](#) (controlador) . Más específicamente, SparkContext puede conectarse a gestores de clúster (gestor de clústeres independiente de Spark, Mesos, YARN o Kubernetes) que son los encargados de asignar recursos en el sistema. Una vez conectados, Spark puede encargar que se creen ejecutores o executors, que son procesos que ejecutan cálculos y almacenan datos para su aplicación. A continuación, envía el código de su aplicación (definido por archivos JAR o Python pasados a SparkContext) a los ejecutores. Los trozos de código propio de los que se encargan estos ejecutores son denominados tasks o tareas.

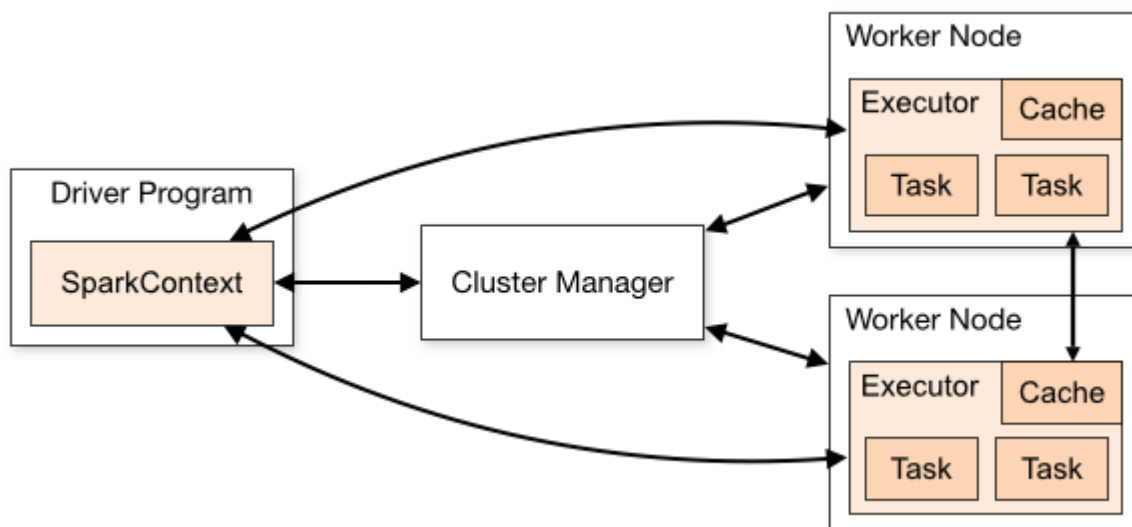


Componentes del clúster de Spark

1. Cada aplicación tiene sus propios procesos ejecutores, que permanecen activos durante toda la aplicación y ejecutan tareas en múltiples hilos. Esto tiene la ventaja de aislar las aplicaciones entre sí, tanto en el lado de la programación (cada controlador programa sus propias tareas) como en el lado del ejecutor (las tareas de diferentes aplicaciones se ejecutan en diferentes JVM). Sin embargo, también significa que los datos no pueden ser compartidos

entre diferentes aplicaciones Spark (instancias de SparkContext) sin escribirlos en un sistema de almacenamiento externo.

2. Spark es agnóstico al gestor de clústeres subyacente. Mientras pueda adquirir procesos ejecutores, y éstos se comuniquen entre sí, es relativamente fácil ejecutarlo incluso en un gestor de clúster que también soporte otras aplicaciones (por ejemplo, Mesos/YARN/Kubernetes).
3. El programa controlador debe escuchar y aceptar conexiones entrantes de sus ejecutores durante toda su vida (por ejemplo, ver `spark.driver.port` en la sección de configuración de la red). Como tal, el programa controlador debe ser direccionable en red desde los nodos trabajadores.
4. Dado que el controlador programa las tareas en el clúster, debe ejecutarse cerca de los nodos trabajadores, preferiblemente en la misma red de área local. Si desea enviar solicitudes al clúster de forma remota, es mejor abrir una RPC al controlador y hacer que envíe operaciones desde la cercanía que ejecutar un controlador lejos de los nodos trabajadores.



Tipos de gestores de Cluster:

El sistema soporta actualmente varios gestores de clústeres:

- Standalone (<https://spark.apache.org/docs/latest/spark-standalone.html>) - un sencillo gestor de clústeres incluido con Spark que facilita la configuración de un clúster.
- ~~Apache Mesos - un gestor de clústeres general que también puede ejecutar Hadoop MapReduce y aplicaciones de servicio. (Deprecated)~~
- Hadoop YARN (<https://spark.apache.org/docs/latest/running-on-yarn.html>) - el gestor de recursos de Hadoop 2.
- Kubernetes - un sistema de código abierto para automatizar el despliegue, el escalado y la gestión de aplicaciones en contenedores.

Spark context: Se trata del contexto básico de Spark, desde donde se crean el resto de variables que maneja el framework.

Término	Significado
Application	un programa spark que que consiste en un driver y los diferentes ejecutores.
Application jar	A jar que contiene la aplicación spark
Driver program	el proceso que corre la funcion principal de la aplicación y crea el contexto spark.
Cluster manager	Un servicio externo para conseguir recursos para el cluster(e.g. standalone manager, Mesos, YARN, Kubernetes)
Deploy mode	establece donde ejecuta el proceso driver. En el modo "cluster", el spark lanza el driver dentro del cluster. En el modo "client", el driver corre fuera del cluster.
Worker node	cualquier nodo que puede ejecutar código de la aplicación en el cluster.
Executor	Es un proceso lanzado por una aplicación en un nodo, Este ejecuta tareas y tiene datos en memoria o los almacena en disco. Cada aplicación tiene sus ejecutores.
Task	una unidad de trabajo que se manda al ejecutor
Job	A parallel computation consisting of multiple tasks that gets spawned in response to a Spark action (e.g. <code>save</code> , <code>collect</code>); you'll see this term used in the driver's logs.
Stage	Each job gets divided into smaller sets of tasks called <i>stages</i> that depend on each other (similar to the map and reduce stages in MapReduce); you'll see this term used in the driver's logs.

1.5. Evaluación perezosa

- <https://riptutorial.com/Download/apache-spark-es.pdf>

Spark utiliza la evaluación perezosa ; eso significa que no hará ningún trabajo, a menos que realmente tenga que hacerlo. Ese enfoque nos permite evitar el uso innecesario de la memoria, lo que nos permite trabajar con big data.

Una transformación se evalúa de forma perezosa y el trabajo real ocurre cuando se produce una acción .

Ejemplo:

```
In [1]: lines = sc.textFile(file)
In [2]: errors = lines.filter(lambda line: line.startsWith("error"))
In [3]: errorCount = errors.count() // an action occurred, let the party start!
Out[3]: 0 // no line with 'error', in this example
```

En [1] le dijimos a Spark que leyera un archivo en un RDD, llamado lines . Spark nos escuchó y nos dijo: "Sí, lo haré", pero en realidad aún no había leído el archivo.

En [2], estamos filtrando las líneas del archivo, asumiendo que su contenido contiene líneas con errores que están marcados con un error en su inicio. Entonces le pedimos a Spark que cree un nuevo RDD, llamado errors , que tendrá los elementos de las lines RDD, que tenían la palabra error al comienzo.

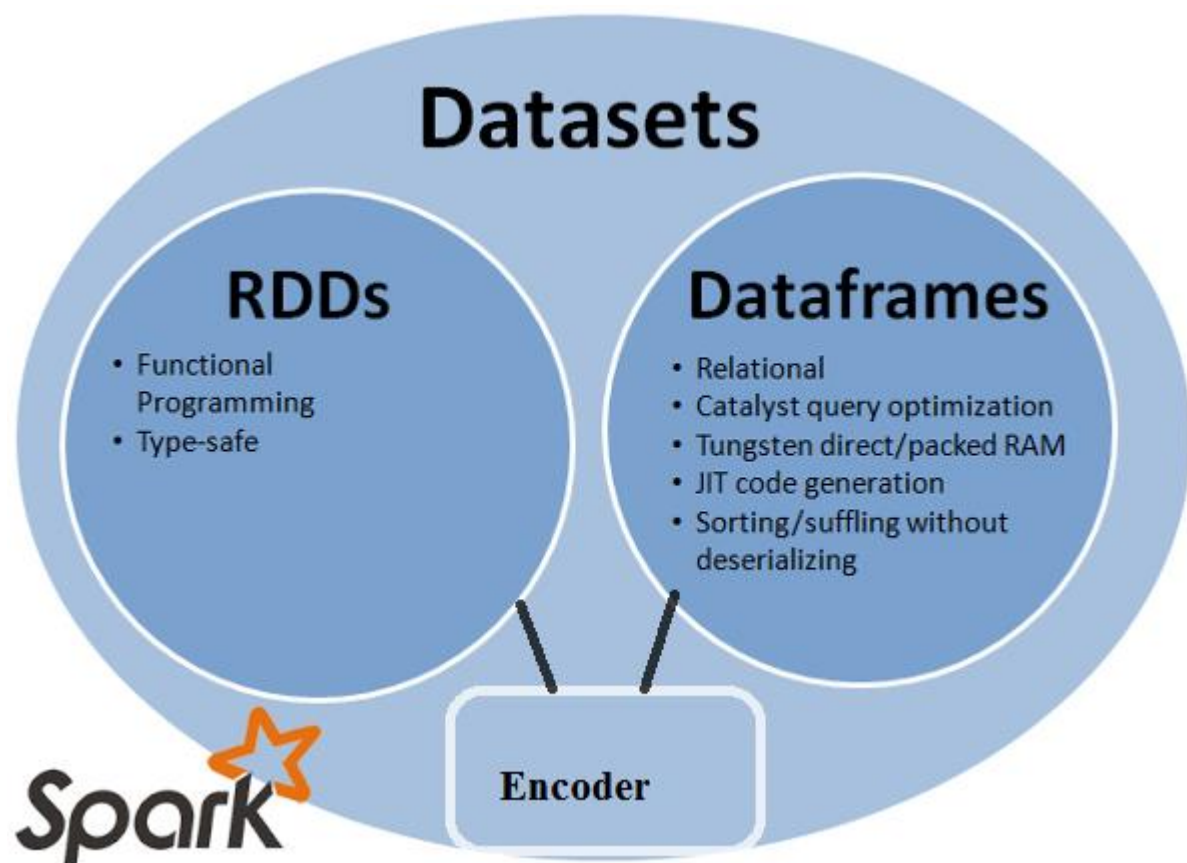
En [3] , le pedimos a Spark que cuente los errores , es decir, que cuente el número de elementos que tiene el RDD llamado errors. count() es una acción , que no deja ninguna opción a Spark, sino a realizar la operación, para que pueda encontrar el resultado de count(). Como resultado, cuando [3] se alcanza, entonces y sólo entonces:

- El archivo se leerá en textFile() (debido a [1])
- se filtrarán las líneas (debido a [2])
- count() se ejecutará, debido a [3]

2 RDD / Data Sets / Data frames (v1.1 actualizado)

Refs:

- <https://spark.apache.org/docs/latest/rdd-programming-guide.html>
- <https://data-flair.training/blogs/apache-spark-dataset-tutorial/>
- <https://www.analyticsvidhya.com/blog/2020/11/what-is-the-difference-between-rdds-dataframes-and-datasets/>
- RDD, data frame y dataset: <https://programmerclick.com/article/2504300331/>
- <https://qastack.mx/programming/31508083/difference-between-dataframe-dataset-and-rdd-in-spark>
- <https://foroayuda.es/diferencia-entre-dataframe-dataset-y-rdd-en-spark/>

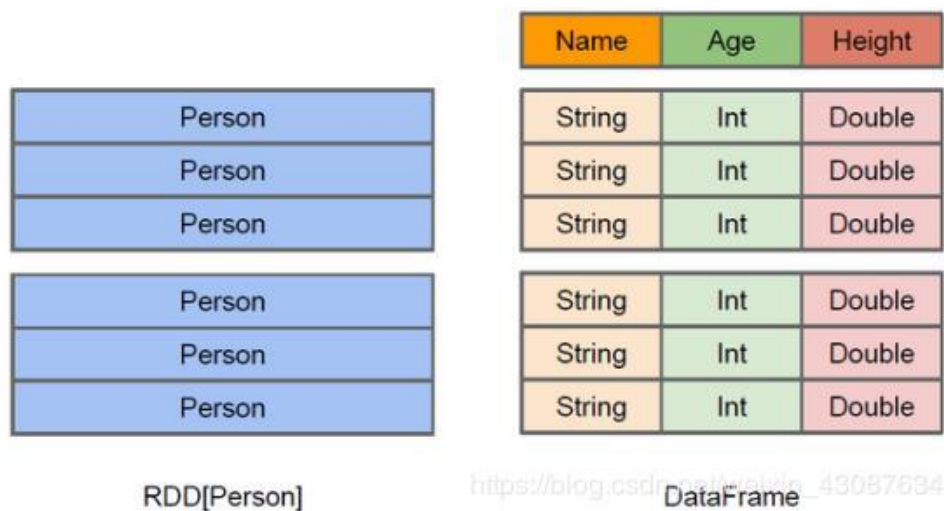


Spark Dispone de tres abstracciones para trabajar con datos.

RDD (Spark1.0) —> Dataframe(Spark1.3) —> Dataset(Spark1.6)

- Data Frames. En un data frame los datos se organizan en columnas con nombre. Por ejemplo una tabla en una base de datos relacional. Es una colección distribuida mutable de datos.
 - El DataFrame en Spark permite a los desarrolladores imponer una estructura a una colección distribuida de datos, permitiendo una abstracción de mayor nivel.
 - Es **Más óptimo que los RDD** por dos motivos
 - Los datos existen en la memoria no heap de manera binaria, lo que ahorra mucho espacio y también elimina las restricciones de GC
 - El Plan de consultas está optimizado por el optimizador de catalizador de Spark
 - La **desventaja de Dataframes** es la falta de verificaciones de seguridad de tipo en tiempo de compilación, lo que causa errores de tiempo de ejecución.

- Dataset. Dataset API es una extensión de DataFrames que proporciona una interfaz de programación orientada a objetos y segura de tipos. Es una colección inmutable fuertemente tipada de objetos que se asignan a un esquema relacional.
 - Una extensión de la API Dataframe.
 - Estilo API fácil de usar, con características de verificación de seguridad de tipo y optimización de consultas de Dataframe.
 - El conjunto de datos admite códecs, que pueden evitar la deserialización de todo el objeto al acceder a datos, lo que mejora la eficiencia.
 - Dataframe es una columna especial de Dataset, DataFrame = Dataset [Row], por lo que puede convertir Dataframe a Dataset como método. Row es un tipo. Al igual que Car y Person, toda la información de la estructura de la tabla está representada por Row.
 - DataSet está fuertemente tipado.
 - Proporciona lo mejor de RDD y Dataframe: RDD (programación funcional, tipo seguro), DataFrame (modelo relacional, optimización de consultas, ejecución de tungsteno, clasificación y barajado)
 -
- RDD (Resilient Distributed Datasets o, en castellano, «conjuntos distribuidos y flexibles de datos»)
 - Es una colección “de **solo lectura**” **distribuida en memoria**. Esto quiere decir que está particionada entre los distintos workers de Spark. “procesamiento en paralelo.
 - Son inmutables: cuando transformamos un nuevo RDD realmente estamos creando uno nuevo.
 - **Tolerante a fallos**
 - Su evaluación es **perezosa**. Con los RDD's estamos definiendo un flujo de información, pero no se ejecuta en el momento de definición, sino en el momento en el que se evalúe aplicando una acción sobre el RDD.
 - El mayor **beneficio de RDD es la simplicidad**, y la API es muy fácil de usar.
 - La **desventaja de RDD es la limitación de rendimiento**, es un objeto de memoria JVM que determina la existencia de limitaciones de GC y el aumento del costo de serialización de Java cuando aumentan los datos

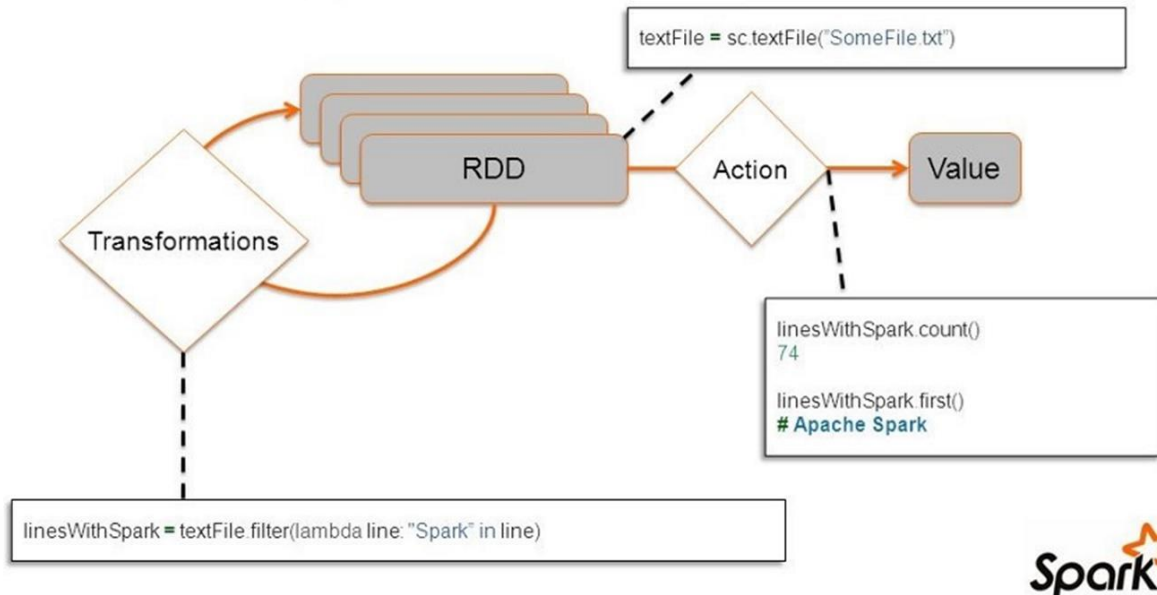


Comparación:

- Si se dan los mismos datos a estas tres estructuras de datos, darán el mismo resultado después de cada cálculo. La diferencia es su eficiencia de ejecución y método de ejecución.
- Los tres tienen mecanismos perezosos.
- En versiones posteriores de Spark, DataSet reemplazará gradualmente RDD y DataFrame como la única interfaz API.
- RDD no conoce los campos, DataFrame solo conoce el campo, pero no conoce el tipo de campo. DataSet conoce no solo los campos, sino también el tipo de campo, por lo que hay una comprobación de errores más estricta. Es una analogía entre los objetos JSON y los objetos de clase.
- RDD se usa generalmente con spark mlib, RDD no es compatible con la operación sparksql
- Tanto DataFrame como DataSet admiten operaciones sparksql, como select, groupby, etc
- DataFrame y DataSet admiten algunos métodos de almacenamiento particularmente convenientes, como guardar como csv,

RDD

Un RDD no solo se encarga de brindar acceso a datos si no también dispone de conjunto de funciones para manipular los datos



Las transformaciones definirán cómo cambiará el flujo de información generando un nuevo RDD. Con estas transformaciones no se está evaluando el RDD, sino creando uno nuevo

Una transformación típica sería el filtrar datos que coincidan con una condición

```
pythonLines = lines.filter(lambda line: "Python" in line)
```

Las acciones nos permitirán evaluar un RDD y devolver un resultado. De esta forma se ejecuta todo el flujo de datos definido

Nota: Antes de Spark 2.0, la principal interfaz de programación de Spark era el conjunto de datos distribuidos resistente (RDD). Después de Spark 2.0, los RDDs son reemplazados por Dataset, que está fuertemente tipado. La interfaz RDD sigue siendo válida, pero por eficiencia se recomienda utilizar Dataset, que tiene un mejor rendimiento que RDD. Consulte la guía de programación de SQL para obtener más información sobre Dataset.

Dependiendo del origen de los datos que contiene, diferenciamos dos tipos de RDDs:

- Colecciones paralelizadas basadas en colecciones.
- Datasets de Hadoop creados a partir de ficheros almacenados en [HDFS](#).

:

Ejemplos RDD

Existen 3 maneras de crear un RDD

<https://www.analyticsvidhya.com/blog/2020/11/what-is-the-difference-between-rdds-dataframes-and-datasets/>

1- Paralelizando un colección de datos

```
# parallelizing data collection
my_list = [1, 2, 3, 4, 5]
my_list_rdd = sc.parallelize(my_list)
```

2 Referenciado a un fichero de datos existente

```
## 2. Referencing to external data file
file_rdd = sc.textFile("path_of_file")
```

3. Creando una RDD desde un RDD existente

leer <https://openwebinars.net/blog/apache-spark-rdds-vs-dataframes/>
<https://aprenderbigdata.com/introduccion-apache-spark/>
<https://sparkbyexamples.com/pyspark-rdd/>
<https://sparkbyexamples.com/pyspark-tutorial/>

Data Frame

Es un conjunto de Datos organizado en Columnas.

Dataframe proporciona una optimización automática, pero carece de seguridad de tipos en tiempo de compilación.

<https://sparkbyexamples.com/pyspark/pyspark-where-filter/>
https://spark.apache.org/docs/3.1.1/api/python/getting_started/quickstart.html#DataFrame-Creation

Creacion de Data Frames

Un DataFrame de PySpark se puede crear a través de `pyspark.sql.Session.createDataFrame` pasándole una lista de listas, tuplas, diccionarios y `pyspark.sql.Rows`, un DataFrame de pandas y un RDD consistente en dicha lista.

pyspark.sql.Session.createDataFrame coge el argumento schema para especificar el esquema del DataFrame. Cuando se omite, PySpark infiere el esquema correspondiente tomando una muestra de los datos.

Maneras de crear un data frame

SPARKSESSION	RDD	DATAFRAME
createDataFrame(rdd)	toDF()	toDF(*cols)
createDataFrame(dataList)	toDF(*cols)	
createDataFrame(rowData,columns)		
createDataFrame(dataList,schema)		

ref: <https://sparkbyexamples.com/pyspark/different-ways-to-create-dataframe-in-pyspark/>
<https://sparkbyexamples.com/pyspark-tutorial/#pyspark-dataframe>
<https://sparkbyexamples.com/spark/spark-dataframe-map-maptypes-column/#maptypes-map>

Los data frame de spark funcionan más rápidos que los dataframe de pandas, gracias a la paralelización en múltiples cores y máquinas

Data Set

Es una colección de datos pero con Estructura “como clases” y mantiene los beneficios de los RDDs (tolerante a fallos y perezoso)

Dataset se añade como una extensión de Dataframe. Dataset combina las características de RDD (es decir, la seguridad de tipos en tiempo de compilación) y Dataframe (es decir, la optimización automática de Spark SQL).

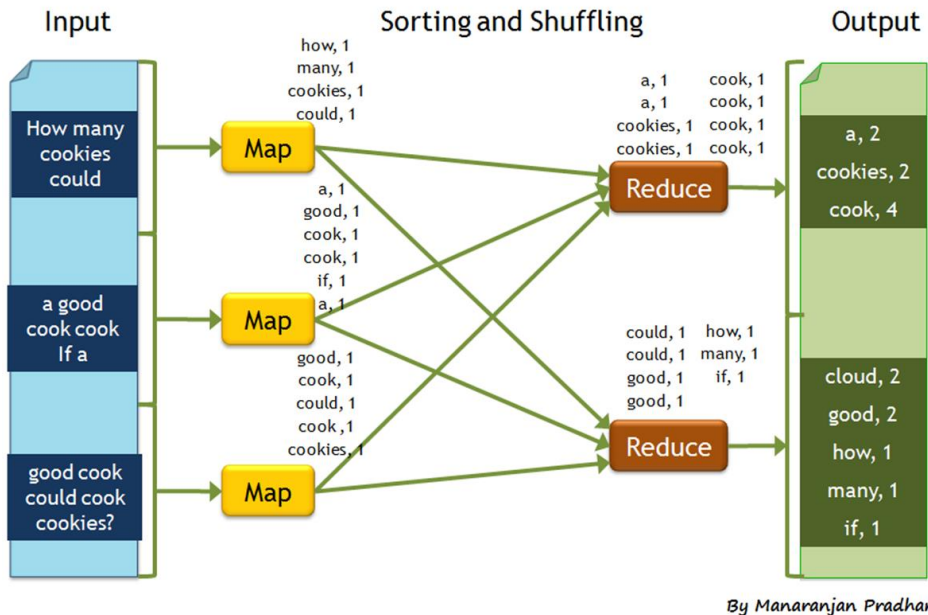
Un Dataset es una colección distribuida de datos. Dataset es una nueva interfaz añadida en Spark 1.6 que proporciona las ventajas de los RDDs (tipado fuerte, capacidad de utilizar potentes funciones lambda) con las ventajas del motor de ejecución optimizado de Spark SQL. Un Dataset puede construirse a partir de objetos JVM y luego manipularse utilizando transformaciones funcionales (map, flatMap, filter, etc.). La API de Dataset está disponible en Scala y Java. Python no tiene soporte para la API de Dataset. Pero debido a la naturaleza dinámica de Python, muchas de las ventajas de la API Dataset ya están disponibles (por ejemplo, se puede acceder al campo de una fila por su nombre, naturalmente `row.columnName`). El caso de R es similar.

Un DataFrame es un Dataset organizado en columnas con nombre. Es conceptualmente equivalente a una tabla en una base de datos relacional o a un DATA FRAME en R/Python, pero con más optimizaciones. Los DataFrames pueden construirse a partir de una amplia gama de fuentes como: archivos de datos estructurados, tablas en Hive, bases de datos externas o RDDs existentes. La API de DataFrame está disponible en Scala, Java, Python y R. En Scala y Java, un DataFrame está representado por un Dataset de filas. En la API de Scala, DataFrame es simplemente un alias del tipo `Dataset[Row]`. Mientras que, en la API de Java, los usuarios necesitan utilizar `Dataset<Row>` para representar un DataFrame.

ref: <https://spark.apache.org/docs/3.1.1/sql-programming-guide.html>

3. Map Reduce

- MapReduce es un modelo de programación para dar soporte a la computación paralela sobre grandes colecciones.
- Su nombre se debe a dos importantes métodos Map y Reduce



- Transformaciones, que crean nuevos conjuntos de datos, como puede ser la operación map() que pasa cada elemento por una determinada función y devuelve un nuevo RDD con el resultado
 - map, flatmap ,filter
 - union, intersection, distinct, join
 - groupByKey, reduceByKey, sortByKey

Transformaciones

- | | |
|----------------|---------------|
| ▶ map | ▶ groupByKey |
| ▶ filter | ▶ reduceByKey |
| ▶ flatMap | ▶ sortByKey |
| ▶ union | ▶ join |
| ▶ intersection | ▶ cogroup |
| ▶ distinct | ▶ coalesce |

- Acciones, que devuelven un valor al driver del clúster después de llevar a cabo una computación sobre el conjunto de datos. Un ejemplo de este tipo es la función `reduce()`, que agrega todos los elementos de un RDD mediante una función y devuelve el resultado.
 - `reduce`,
 - `first`, `take`
 - `count`, `collect`, `main`, `min`,
 - `saveAsTextFile`
 - `countByKey`
 - `foreach`

<https://spark.apache.org/docs/latest/rdd-programming-guide.html#transformations>

Ejemplos :

- `map`: aplica una función a cada elemento de la colección:
`intValues.map(_.toString) // RDD[String]`
- `filter`: selecciona el subconjunto de elementos que cumplen una determinada expresión booleana:
`IntValues.filter(_.isOdd) // RDD[Int]`
- `flatMap`: además de realizar una función `map`, aplica un método `flatten`:
`textFile.map(_.split(" ")) textFile.flatMap(_.split(" "))`

Funciones Lamba

Lambda es una manera de escribir/definir funciones Inline (anónimas) de manera rápida

lambda parámetros: expresión

lambda x, y : x ** y

```
def cuadrado(x):
    return x ** 2
cuad = cuadrado (x)

cuad = lambda x: x ** 2
```

Ejemplos funciones lambda:

```
words = lines.flatMap(lambda line: line.split(" "))
pairs = words.map(lambda word: (word, 1))
wordCounts = pairs.reduceByKey(lambda x, y: x + y)
```

- <https://realpython.com/python-lambda/>
- <https://towardsdatascience.com/lambda-functions-with-practical-examples-in-python-45934f3653a8>
-

Ejemplo: Programa que cuenta palabras en spark haciendo uso de las funciones Map y reduce. (Lenguaje scala)

1. **Map:** Dividiremos las frases en palabras y las asignaremos un 1.
2. **Reduce:** Agregaremos las palabras y sumaremos el contador asignado anteriormente, de tal modo que las palabras que sean iguales se sumarán dando como resultado el número total de palabras iguales.

En primer lugar, utilizando el context de Spark (spark), accedemos al context SparkContext que nos provee del método `textFile` que permite la lectura de un fichero y convertirlo en un RDD.

```
val textFile = spark.sparkContext.textFile("ruta/del/ficher.txt")
```

Una vez tenemos el RDD es necesario realizar un `split` para dividir cada frase en palabras. La idea es que por cada registro original se generen `n` registros donde `n` es el número de palabras en cada registro. Al querer generar más de un RDD por cada uno de los RDD es necesario utilizar la función `flatMap` en lugar de la función `map`.

Después, con la función `map`, les asignamos el 1 generando una tupla clave-valor. Finalizamos utilizando la función `reduceByKey` que permite agregar los datos por la clave (Word) pasándole una función de agregación, en este caso, la suma de sus valores.

```
val counts = textFile.flatMap(line => line.split(" "))
  .map(word => (word, 1))
  .reduceByKey(_ + _)
```

con la función `collect`, transformamos el RDD en un Array, el cual podemos recorrer e imprimir.

```
counts.collect.foreach(println _)
```



```
File Edit View Search Terminal Help
(caballero,100)
(quédesele,1)
(mata,2)
(medio?,1)
(cupo,2)
(descubrieron,3)
(describen,1)
(ama,,7)
(tragedia,,1)
(Que,16)
(despojado,1)
(confianza,2)
(llenas,2)
(ejecuta,1)
(triste,6)
(Traian,1)
(fendientes,,1)
(decantado,1)
(vivientes,,1)
(despedian,2)
(caldeos,,1)
(italiano,,1)
(tal,,4)
(tenga,10)
(subiendo,1)
(encajallo,1)
(trofeo,1)
(derribó,3)
(azogado,,1)
(saludado,1)
(describe,1)
(libros,,11)
(hallaren,,1)
(créame,1)
(ahorrar,1)
```

La función `swap` intercambia el orden de una tupla. Es necesario usar la función `map` para aplicar esta función a cada una de las tuplas que forman el RDD.

La función `take` funciona como la función `collect` pero recibiendo como parámetros el número de registros del RDD que formarán parte del array.

```
scala> counts.take(10).foreach(println _)
(2989,)
(2975,que)
(2780,de)
(2479,y)
(1406,la)
(1401,a)
(1177,el)
(1119,en)
(822,no)
(753,se)
scala> 
```

4 Spark Streaming

ref: <https://spark.apache.org/docs/latest/streaming-programming-guide.html>

Spark Streaming puede ingerir datos de un amplio de fuentes, incluyendo flujos provenientes de Apache Kafka, Apache Flume, Amazon Kinesis y Twitter, así como de sensores y dispositivos conectados por medio de sockets TCP. También se pueden procesar datos almacenados en sistemas de archivos como HDFS o Amazon S3.



A grandes rasgos, lo que hace Spark Streaming es tomar un flujo de datos continuo y convertirlo en un flujo discreto —llamado DStream— formado por paquetes de datos. Internamente, lo que sucede es que Spark Streaming almacena y procesa estos datos como una secuencia de RDDs (Resilient Distributed Data).

Spark streaming no es un streaming en tiempo real puro, más bien se trata de procesar micro-streams/micro-batch cada poco tiempo (0,5 , 1, 5, 10 seg) “Casi tiempo real”. Si necesitáramos un streaming puro, spark streaming no sería la herramienta más adecuada.

Adicionalmente, un argumento en contra del esquema de micro-batches es que puede ser que los datos no se reciban en el orden exacto en el que sucedieron.



Spark Streaming proporciona una abstracción de alto nivel llamada DStream, que representa un flujo continuo de streams de datos. Un DStream se puede crear ya sea a partir de flujos de datos de entrada o fuentes como Kafka, Flume y Kinesis. Internamente la representación de un DStream es una secuencia de RDDs

El intervalo de los batch se debe configurar en función de las necesidades de la aplicación y de los recursos disponibles. En base al intervalo, a la cantidad de datos a procesar y el procesamiento a aplicar, las necesidades varían considerablemente.

Información sobre ajuste de rendimiento →

<https://spark.apache.org/docs/latest/streaming-programming-guide.html#setting-the-right-batch-interval>

Flujo /etapas del procesamiento de datos en streaming

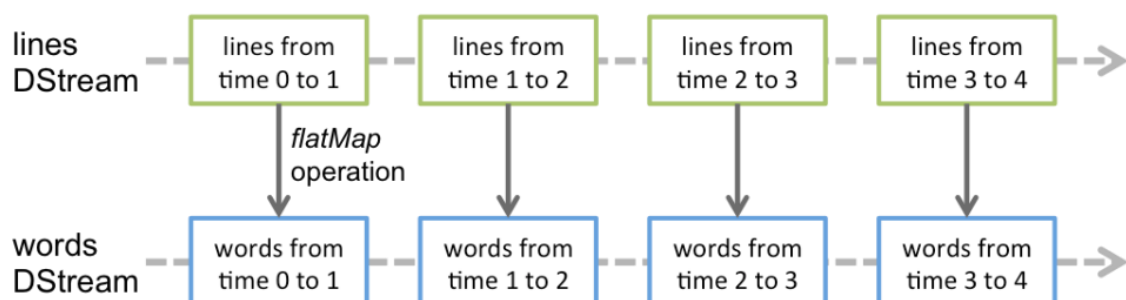
1. se reciben los datos
2. se transforman los datos usando DStream y RDD transformaciones
3. se guardan/envían los datos → bases de datos , brokers

Discretized streams

DStreams (Discretized Streams): es la abstracción básica proporcionada por Spark Streaming. Representa un stream continuo de datos, ya sea el flujo de entrada recibido desde una fuente, o el flujo de datos procesados de salida. La representación de un DStream es una secuencia de RDDs ordenados en el tiempo, cada uno de ellos guardando datos para un intervalo concreto.

DStreams (Discretized Streams): es la abstracción básica proporcionada por Spark Streaming. Representa un stream continuo de datos, ya sea el flujo de entrada recibido desde una fuente, o el flujo de datos procesados de salida. La representación de un DStream es una secuencia de RDDs ordenados en el tiempo, cada uno de ellos guardando datos para un intervalo concreto.

Cualquier operación realizada sobre un DStream se traduce en una operación sobre cada uno de los RDDs que lo forman



Las transformaciones que se pueden aplicar sobre un DStream son prácticamente las mismas que se pueden aplicar sobre un RDD, aunque existen tres funciones a las que vamos a prestar más atención

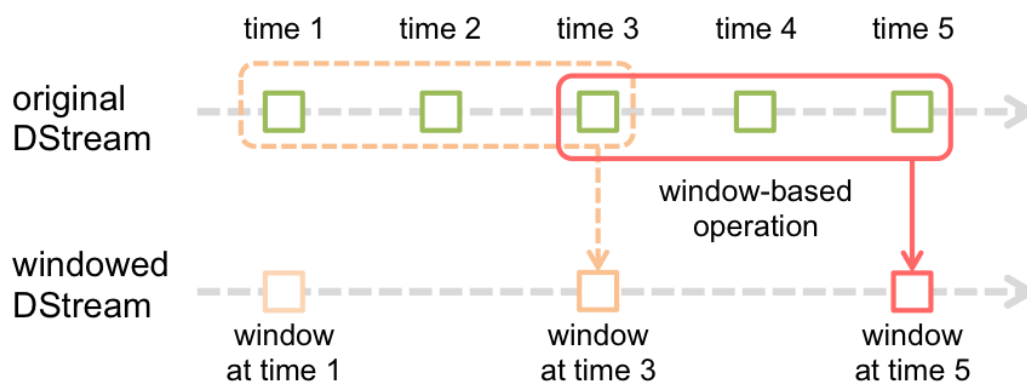
- **UpdateStateByKey:** esta operación permite mantener un estado arbitrario mientras se actualiza continuamente con nueva información. Para poder utilizarlo hay que:
 - Definir el estado. Puede ser un tipo de dato arbitrario.
 - Definir la función de actualización de estado. Especificar en una función como actualizar el estado utilizando el estado anterior y los nuevos valores de un stream de entrada.
 - En cada batch Spark aplicará la función de actualización de estado para todas las claves existentes, independientemente de si se tienen nuevos datos o no. Si la función devuelve "none" entonces se eliminará el par de clave-valor.

https://github.com/apache/spark/blob/v3.2.1/examples/src/main/python/streaming/stateful_network_wordcount.py

```
def updateFunction(newValues, runningCount):  
    if runningCount is None:  
        runningCount = 0  
    return sum(newValues, runningCount) # add the new values with the previous  
running count to get the new count
```

```
runningCounts = pairs.updateStateByKey(updateFunction)
```

- **Transform:** son operaciones que permiten aplicar funciones de RDD a RDD sobre un DStream. También puede utilizarse para aplicar funciones definidas por el usuario y que no están definidas en la API DStream.
- **Window (Ventanas deslizantes)** : las operaciones de ventana actúan sobre los datos de una duración concreta. Si necesitamos disponer de los datos de los último diez intervalos de tiempo para realizar algún cálculo podemos utilizar alguna de las operaciones de ventana.



window length: 3 - la duración de la ventana

sliding interval: 2 - el intervalo “deslizante” con el que se procesa la ventana

El tamaño de los dos parámetros anteriores debe ser un múltiplo del tiempo de recepción para generar un DStream

Ejemplo: se quiere calcular un contador de palabras introducidas en los últimos 30 segundos y queremos que el cálculo se realice cada 10 segundos

```
# Reduce last 30 seconds of data, every 10
seconds
windowedWordCounts =
pairs.reduceByKeyAndWindow(lambda x, y: x + y,
30, 10)
```

Tansformation	Meaning
window(windowLength, slideInterval)	Return a new DStream which is computed based on windowed batches of the source DStream.
countByWindow(windowLength, slideInterval)	Return a sliding window count of elements in the stream.

reduceByWindow(func, windowLength, slideInterval)	Return a new single-element stream, created by aggregating elements in the stream over a sliding interval using func. The function should be associative and commutative so that it can be computed correctly in parallel.
reduceByKeyAndWindow(func, windowLength, slideInterval, [numTasks])	When called on a DStream of (K, V) pairs, returns a new DStream of (K, V) pairs where the values for each key are aggregated using the given reduce function func over batches in a sliding window. Note: By default, this uses Spark's default number of parallel tasks (2 for local mode, and in cluster mode the number is determined by the config property spark.default.parallelism) to do the grouping. You can pass an optional numTasks argument to set a different number of tasks.
reduceByKeyAndWindow(func, invFunc, windowLength, slideInterval, [numTasks])	A more efficient version of the above reduceByKeyAndWindow() where the reduce value of each window is calculated incrementally using the reduce values of the previous window. This is done by reducing the new data that enters the sliding window, and "inverse reducing" the old data that leaves the window. An example would be that of "adding" and "subtracting" counts of keys as the window slides. However, it is applicable only to "invertible reduce functions", that is, those reduce functions which have a corresponding "inverse reduce" function (taken as parameter invFunc). Like in reduceByKeyAndWindow, the number of reduce tasks is configurable through an optional argument. Note that checkpointing must be enabled for using this operation.
countByValueAndWindow(windowLength, slideInterval, [numTasks])	When called on a DStream of (K, V) pairs, returns a new DStream of (K, Long) pairs where the value of each key is its frequency within a sliding window. Like in reduceByKeyAndWindow, the number of reduce tasks is configurable through an optional argument.

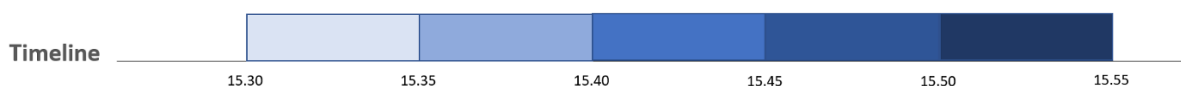
ref: ventanas <https://towardsdatascience.com/spark-3-2-session-windowing-feature-for-streaming-data-e404d92e267>
<https://databricks.com/blog/2021/10/12/native-support-of-session-window-in-spark-structured-streaming.html>

Tipos de ventanas temporales

- tumbling windows

Las ventanas "tumbling" pueden representarse como un grupo de periodos de tiempo adyacentes, igualmente divididos, sin tener intervalos de intersección. Los datos introducidos pueden estar sujetos a una ventana individual.

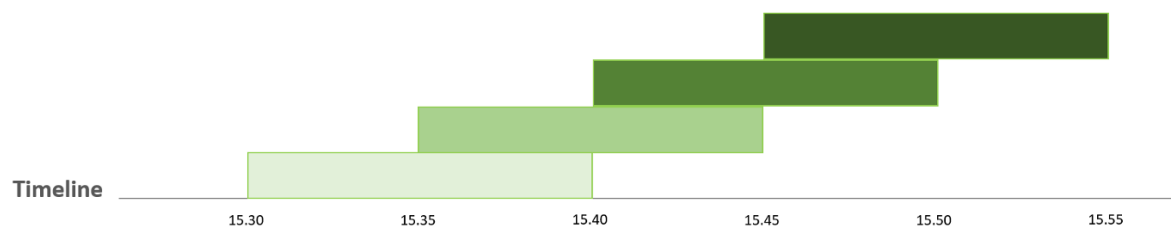
Cuando se observan en una línea de tiempo, las ventanas pueden verse como una secuencia seguida de forma estática como en la siguiente imagen.



- Sliding windows: la ventana de mueve en intervalos. durante ese intervalo se calcula el valor

Las ventanas deslizantes pueden tener periodos de tiempo que se cruzan cuando el lapso de tiempo contiene un intervalo más corto que el rango de la ventana. En estos casos, los elementos con marca de tiempo pueden encontrarse en más de una ventana.

Cuando se observan en una línea de tiempo, las ventanas pueden superponerse o no según la longitud del intervalo de tiempo cuando se comparan con la longitud de la ventana deslizante principal de manera estática.

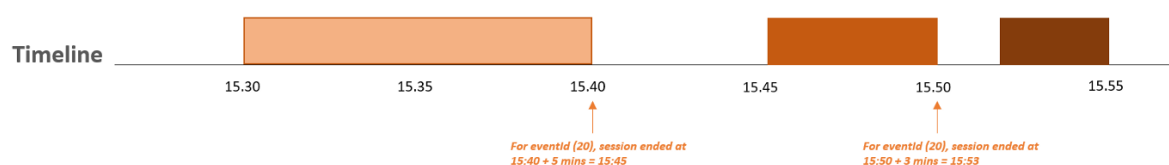


- session windows: dinamico

Las ventanas de sesión tienen una característica diferente en comparación con los dos tipos anteriores. La ventana de sesión tiene un tamaño dinámico de la longitud de la ventana, en función de las entradas. Una ventana de sesión comienza con una entrada y se expande si la siguiente entrada se ha recibido dentro de la duración de la brecha. Una ventana de sesión se cierra cuando no se recibe ninguna entrada dentro de la duración de la brecha después de recibir la última entrada. Esto permite agrupar eventos hasta que no haya nuevos eventos durante un tiempo determinado (inactividad).

Funciona de manera similar a una sesión en un sitio web que tiene tiempo de espera de la sesión - si te conectas a un sitio web y no muestras ninguna actividad durante algún tiempo, el sitio web te pedirá que mantengas el estado de conexión y forzará el cierre de la sesión, si sigues inactivo después de que se haya superado el tiempo de espera. El tiempo de espera de la sesión se prolonga siempre que se muestra actividad.

Aplicando esto a la ventana de sesión: se inicia una nueva ventana de sesión cuando se produce un nuevo evento, como un trabajo de streaming, y los siguientes eventos dentro del tiempo de espera se incluirán en la misma ventana de sesión. Cada evento ampliará el tiempo de espera de la sesión, lo que introduce una característica diferente en comparación con las otras ventanas de tiempo: la duración de la ventana de sesión no es estática, mientras que las ventanas móviles y deslizantes tienen una duración estática.



- **foreachRDD:** es la función que más se suele usar para guardar/enviar los datos a un sistema externo tras su procesamiento.

Normalmente para mandar datos al exterior, se requiere crear una conexión (por ejemplo TCP a un servidor remoto) para enviar los datos a un sistema remoto

```
def sendPartition(iter):  
    connection = createNewConnection()  
    for record in iter:  
        connection.send(record)  
    connection.close()  
  
dstream.foreachRDD(lambda rdd:  
    rdd.foreachPartition(sendPartition))
```

Checkpoint

Una aplicación de streaming tiene que estar operativa 24/7 y por eso debe ser tolerante a fallos que no sean propios de la aplicación. Por eso Spark Streaming necesita verificar la suficiente información a un sistema de almacenamiento con tolerancia a fallos de tal manera que pueda recuperar los datos perdidos.

```
# Create a local StreamingContext with two working thread and batch interval of 1 second  
sc = SparkContext("local[2]", "NetworkWordCount")
```

Cuando ejecute un programa de Spark Streaming localmente, no utilice "local" o "local[1]" como URL principal. Cualquiera de ellas significa que sólo se utilizará un hilo para ejecutar las tareas localmente. Si está utilizando un DStream de entrada basado en un receptor (por ejemplo, sockets, Kafka, etc.), entonces el único hilo se utilizará para

ejecutar el receptor, sin dejar ningún hilo para procesar los datos recibidos. Por lo tanto, cuando se ejecuta localmente, siempre use "local[n]" como la URL maestra, donde n > número de receptores a ejecutar (ver Propiedades de Spark para información sobre cómo establecer la maestra).

Streaming con Data Frames

También se pueden usar los dataframes y sql para procesamiento en streaming.

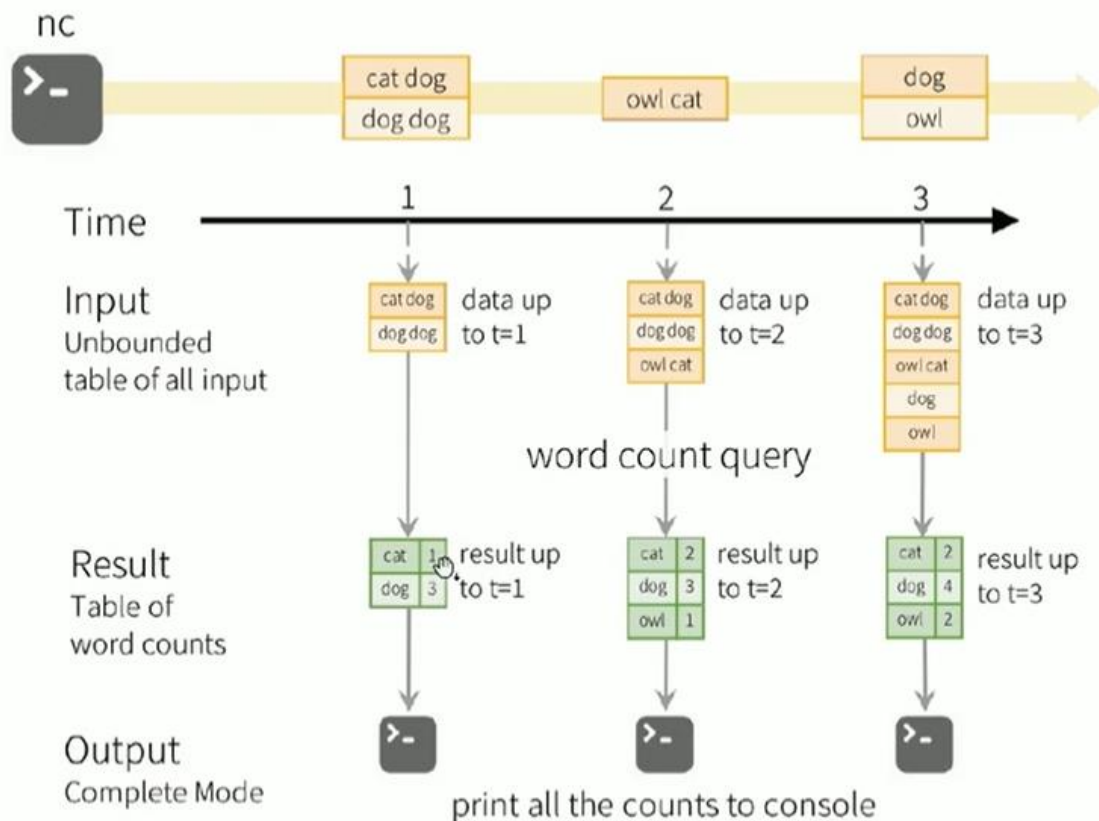
```
def process(time, rdd):  
    print("===== %s =====" % str(time))  
    try:  
        # Get the singleton instance of SparkSession  
        spark = getSparkSessionInstance(rdd.context.getConf())  
  
        # Convert RDD[String] to RDD[Row] to DataFrame  
        rowRdd = rdd.map(lambda w: Row(word=w))  
        wordsDataFrame = spark.createDataFrame(rowRdd)  
  
        # Creates a temporary view using the DataFrame  
        wordsDataFrame.createOrReplaceTempView("words")  
  
        # Do word count on table using SQL and print it  
        wordCountsDataFrame = spark.sql("select word, count(*) as total  
from words group by word")  
        wordCountsDataFrame.show()  
    except:  
        pass
```

Structured Streaming

Structured Streaming es un mecanismo de procesamiento en tiempo real que se construye por encima de Spark SQL. Funciona con Data frames

Modos de procesar la información.

- Complete
- Append
- Update



Spark streaming con twitter

<https://www.toptal.com/apache/tutorial-apache-spark-streaming-identificando-los-hashtags-de-tendencia-de-twitter>

Integracion kafka con spark streaming

With Kafka Direct API

In Spark 1.3, we have introduced a new Kafka Direct API, which can ensure that all the Kafka data is received by Spark Streaming exactly once. Along with this, if you implement exactly-once output operation, you can achieve end-to-end exactly-once guarantees. This approach is further discussed in the [Kafka Integration Guide](#).

<https://spark.apache.org/docs/latest/streaming-kafka-0-10-integration.html>

<https://spark.apache.org/docs/latest/structured-streaming-kafka-integration.html>

<https://karthiksharma1227.medium.com/integrating-kafka-with-pyspark-845b065ab2e5>

<https://medium.com/geekculture/integrate-kafka-with-pyspark-f77a49491087>

Referencias

- <https://sparkbyexamples.com/pyspark-tutorial/>
- <https://aprenderbigdata.com/introduccion-apache-spark/>