Contenido

[1. Autores del trabajo, planificación y entrega 2](#_Toc448254544)

[1.1 Autores 2](#_Toc448254545)

[1.2 Planificación 2](#_Toc448254546)

[1.3 Entrega 2](#_Toc448254547)

[2. Requisitos del prototipo a implementar 3](#_Toc448254548)

[2.1 Requisitos funcionales 3](#_Toc448254549)

[2.2 Otros requisitos 3](#_Toc448254550)

[3. Criterios de comparación en la implementación 4](#_Toc448254551)

[3.1 Criterio 1: Nombre del criterio 4](#_Toc448254552)

[3.2 Criterio 2: Nombre del criterio 4](#_Toc448254553)

[3.N Criterio N: Nombre del criterio 4](#_Toc448254554)

[4. Proyecto de implementación de un prototipo del sistema utilizando la tecnología A 5](#_Toc448254555)

[4.1 Documentación de diseño 5](#_Toc448254556)

[4.2 Documentación de construcción 5](#_Toc448254557)

[4.3 Documentación de pruebas 5](#_Toc448254558)

[4.4 Documentación de instalación 5](#_Toc448254559)

[4.5 Manual de usuario 5](#_Toc448254560)

[5. Proyecto de implementación de un prototipo del sistema utilizando la tecnología B 6](#_Toc448254561)

[5.1 Documentación de diseño 6](#_Toc448254562)

[5.2 Documentación de construcción 6](#_Toc448254563)

[5.3 Documentación de pruebas 6](#_Toc448254564)

[5.4 Documentación de instalación 6](#_Toc448254565)

[5.5 Manual de usuario 6](#_Toc448254566)

[6. Comparación de las dos implementaciones 7](#_Toc448254567)

[6.1 Evaluación de los criterios en la implementación usando la tecnología A 7](#_Toc448254568)

[6.2 Evaluación de los criterios en la implementación usando la tecnología B 7](#_Toc448254569)

[7. Comparación de la implementación de las tecnologías 8](#_Toc448254570)

[8. Conclusiones 10](#_Toc448254571)

# 1. Autores del trabajo, planificación y entrega

## 1.1 Autores

Los componentes del grupo T7, Javier Cia, Alberto Moraga y Jorge Benitez

## 1.2 Planificación

En este apartado se debe incluir un enlace (URL) compartido a la planificación del trabajo utilizando una herramienta online de diagramación Gantt (por ejemplo, GanttPro, versión gratuita).

Hay que tener en cuenta que cada participante del grupo debe tener asignadas tareas que sumen al menos 45 horas. El peso de este trabajo en la calificación total de la asignatura es de un 30%, por tanto requiere de una dedicación de 45 horas del total de 150 horas de la asignatura.

## 1.3 Entrega

En este apartado debe incluirse un enlace (URL) a un repositorio en GitHub o en BitBucket creado para el trabajo.

En dicho repositorio debe encontrarse, al menos los siguientes archivos en la rama máster:

* Informe del trabajo: con el nombre TG3\_final.docx
* Presentación del trabajo: TG3\_final.pptx
* Prototipos obtenidos implementando cada una de las tecnologías (deben incluir el código fuente y todos los archivos necesarios para la instalación y uso de cada prototipo):
  + PrototipoTecnologiaA\_final.zip (o .rar)
  + PrototipoTecnologiaB\_final.zip (o .rar).

Dichos archivos serán los que se tendrán en cuenta para la calificación del trabajo.

# 2. Requisitos del prototipo a implementar

En nuestro caso básicamente hemos creado dos equipos de trabajo que comparten un mismo objetivo con el uso de diferentes tecnologías.

En el caso del grupo uno deberá implementar una aplicación en Spark que al ser ejecutada realice un conteo de palabras sobre un archivo de texto.

En el caso del grupo dos deberá implementar una aplicación en Hadoop (utilizando MapReduce), la finalidad será la misma que la del grupo uno.

## 2.1 Requisitos funcionales

Los requisitos funcionales deben ser los mismos para las dos implementaciones.

En la siguiente tabla se indicará el catálogo de requisitos funcionales del sistema.

| **REQ.** | **DESCRIPCIÓN** |
| --- | --- |
| RF01 | El código debe realizar una función de conteo de palabras. |
| RF02 | El código debe mapear las palabras contenidas en las frases |
| RF03 | El código debe realizar un reduce para realizar el conteo de palabras |
| RF04 | El código debe realizarlo de la forma más rápida posible utilizando los estándares de programación de conteo de palabras con Spark y MapReduce. |
| RF05 | El código deberá ser capaz de demostrar en que requisitos es superior Spark a MapReduce. |

## 2.2 Otros requisitos

En este apartado hemos incluido los requisitos no funcionales y aun así necesarios y críticos para una correcta comparación entre Hadoop y Spark

| **REQ.** | **DESCRIPCIÓN** |
| --- | --- |
| R01 | El código usará la configuración de seguridad de la que HDFS dispone |
| R02 | El código será ejecutado sobre una máquina virtual CENTOs |
| R03 | La finalidad del código será comparar dos tecnologías, esto se debe tener en cuenta en la implementación |

# 3. Criterios de comparación en la implementación

## 3.1 Criterio 1: Tiempo

3.1.1 Planificación

Nombre del criterio: Tiempo de planificación.

Descripción: Tiempo dedicado a la planificación previa del prototipo.

Tipo de valor: Numérico.

3.1.2 Organización

Nombre del criterio: Tiempo de organización.

Descripción: Tiempo dedicado a la organización de los recursos para la implementación de los requisitos.

Tipo de valor: Numérico.

3.1.3 Formación

Nombre del criterio: Tiempo de formación en las herramientas.

Descripción: Tiempo dedicado al aprendizaje básico de utilización de las herramientas.

Tipo de valor: Numérico.

3.1.4 Preparación

Nombre del criterio: Tiempo de preparación.

Descripción: Tiempo dedicado a la instalación y configuración de las herramientas.

Tipo de valor: Numérico.

3.1.5 Programación

Nombre del criterio: Tiempo de programación.

Descripción: Tiempo dedicado a la realización de los programas a utilizar.

Tipo de valor: Numérico.

3.1.6 Pruebas

Nombre del criterio: Tiempo de pruebas.

Descripción: Tiempo dedicado a la realización de las pruebas necesarias hasta la finalización del prototipo.

Tipo de valor: Numérico.

.

3.1.7 Resultado final

Nombre del criterio: Tiempo de obtención del resultado final.

Descripción: Tiempo que tarda en devolver el resultado final el prototipo.

Tipo de valor: Numérico.

## 3.2 Criterio 2: Utilización

3.2.1 Usabilidad

Nombre del criterio: Facilidad de uso.

Descripción: Nivel de facilidad obtenido a la hora de implementar el prototipo completamente.

Tipo de valor: Alto/Bajo.

3.2.2 Adaptación

Nombre del criterio: Adaptación.

Descripción: Capacidad de adaptarse a los problemas.

Tipo de valor: Alto/Bajo.

3.2.3 Recuperación

Nombre del criterio: Recuperación de datos.

Descripción: Capacidad de recuperación de datos.

Tipo de valor: Booleano(Sí/No).

3.2.4 Compresión

Nombre del criterio: Compresión.

Descripción: Nivel de facilidad de entender el funcionamiento del programa.

Tipo de valor: Alto/Bajo.

3.2.5 Lenguaje

Nombre del criterio: Lenguaje de programación.

Descripción: Lenguaje de programación utilizado.

Tipo de valor: Texto.

## 3.3 Criterio 3: Calidad

3.3.1 Seguridad

Nombre del criterio: Seguridad.

Descripción: Capacidad de controlar y proteger los datos utilizados.

Tipo de valor: Booleano(Sí/No).

3.3.2 Tolerancia

Nombre del criterio: Tolerancia a fallos.

Descripción: Capacidad de sufrir errores inesperados.

Tipo de valor: Booleano(Sí/No).

3.3.3 Rendimiento

Nombre del criterio: Rendimiento del programa.

Descripción: Nivel del rendimiento de la ejecución de un programa respecto al tiempo de ejecución.

Tipo de valor: Alto/Bajo.

3.3.4 Escalabilidad

Nombre del criterio: Escalabilidad.

Descripción: Capacidad de aumentar y adaptarse al trabajo sin mantener calidad.

Tipo de valor: Booleano(Sí/No).

3.3.5 Modularidad

Nombre del criterio: Modularidad.

Descripción: Capacidad de independencia funcional de los distintos componentes.

Tipo de valor: Booleano(Sí/No).

3.3.6 Eficacia

Nombre del criterio: Eficacia.

Descripción: Nivel de eficacia a la hora de devolver el resultado final.

Tipo de valor: Alto/Bajo.

3.3.7 Eficiencia

Nombre del criterio: Eficiencia.

Descripción: Nivel de eficiencia a la hora de devolver el resultado final con respecto al tiempo.

Tipo de valor: Alto/Bajo.

# 4. Proyecto de implementación de un prototipo del sistema utilizando la tecnología A

Se trata de incluir en este apartado la documentación del desarrollo del proyecto de implementación, utilizando la tecnología A, del sistema cuyos requisitos funcionales se enumeraron en el apartado 2.

## 4.1 Documentación de diseño

Hay que incluir la descripción del diseño del prototipo, incluyendo diagramas, y el diseño de la interfaz de usuario.

## 4.2 Documentación de construcción

Hay que incluir una descripción de la construcción del prototipo, incluyendo algún extracto de código fuente. No es necesario todo el código. Sólo algún extracto para ver cómo se ha comentado.

## 4.3 Documentación de pruebas

Casos de prueba establecidos y resultados de las pruebas y acciones de corrección. No es creíble que no hayan aparecido errores en los caso de prueba.

## 4.4 Documentación de instalación

Descripción suficiente para que una persona que no ha participado en el proyecto pueda instalar el prototipo.

## 4.5 Manual de usuario

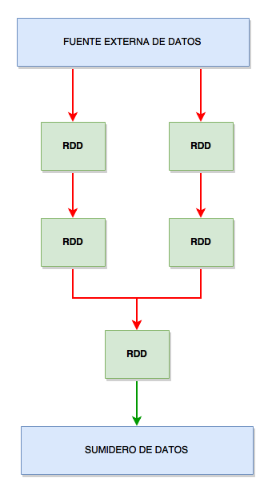
Descripción suficiente para que una persona que no ha participado en el proyecto pueda utilizar toda la funcionalidad que ofrece el prototipo. Que debe coincidir con los requisitos funcionales incluidos en el apartado 2.

# . Proyecto de implementación de un prototipo del sistema utilizando la tecnología Spark

Se trata de incluir en este apartado la documentación del desarrollo del proyecto de implementación, utilizando la tecnología B, del sistema cuyos requisitos funcionales se enumeraron en el apartado 2.

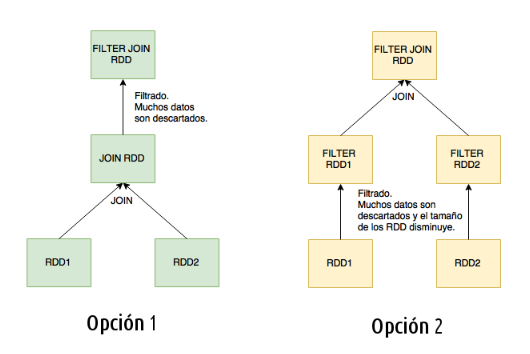
## 5.1 Documentación de diseño

El prototipo se ha diseñado de tal forma que el funcionamiento fundamental corresponde al mostrado en el siguiente diagrama, donde los nodos son objetos RDD y las operaciones entre ellos son operaciones de transformación (representadas por líneas rojas) u operaciones (línea verde):



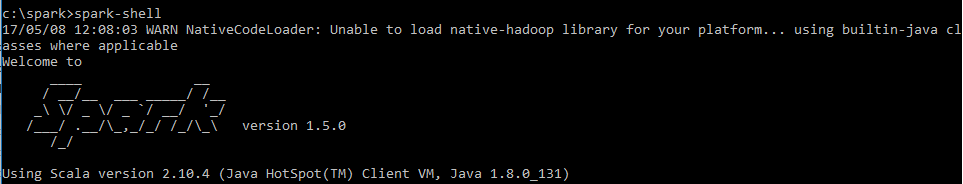
Los objetos RDD iniciales leen datos de ficheros, bases de datos u otras fuentes de información. Una vez creados los RDD iniciales, pasan por una serie de operaciones de trasformación en término de programación funcional, donde no se eliminan ni se alteran, simplemente crea otros RDD con los que trabajar más fácilmente, para, finalmente, converger en un mismo RDD.

La convergencia de los RDD puede ser de 2 formas:



La opción 2 es mucho más efectiva ya que filtra los datos de cada RDD antes de convergerlos, eliminando datos innecesarios a la hora de mezclar, no como la opción 1 que antes de filtrar los datos, realiza la convergencia con la totalidad de estos.

En cuanto la interfaz de usuario, como mencionaremos en el apartado de instalación, Spark trabaja desde el Símbolo del sistema.



Mediante una serie de comandos se cargarían los datos necesarios y el programa a utilizar para obtener el resultado final. Dichos comandos se mencionan más adelante.

## 5.2 Documentación de construcción

El prototipo diseñado está creado en Scala y su función principal es hacer un conteo exacto de las palabras existentes en un documento. Esto es muy útil y práctico a la hora de comprobar requisitos de trabajos o proyectos en los que exigen limitaciones de extensión en número de palabras.

A continuación, mostramos el código que se ha diseñado:

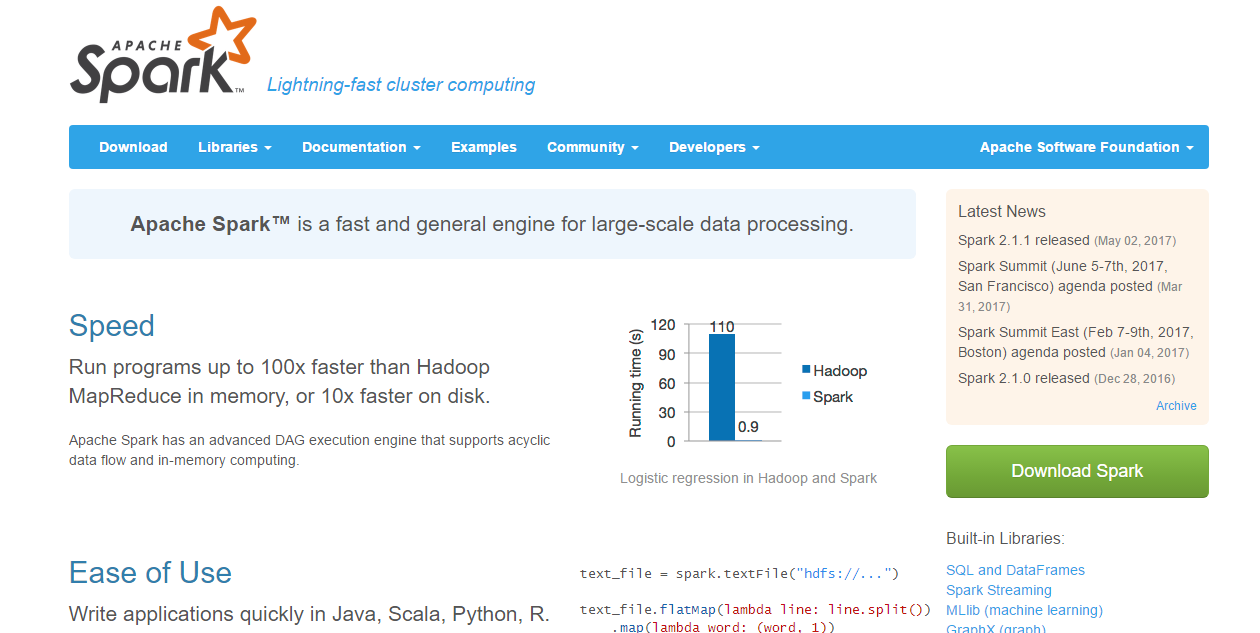
## 5.3 Documentación de pruebas

Casos de prueba establecidos y resultados de las pruebas y acciones de corrección. No es creíble que no hayan aparecido errores en los caso de prueba.

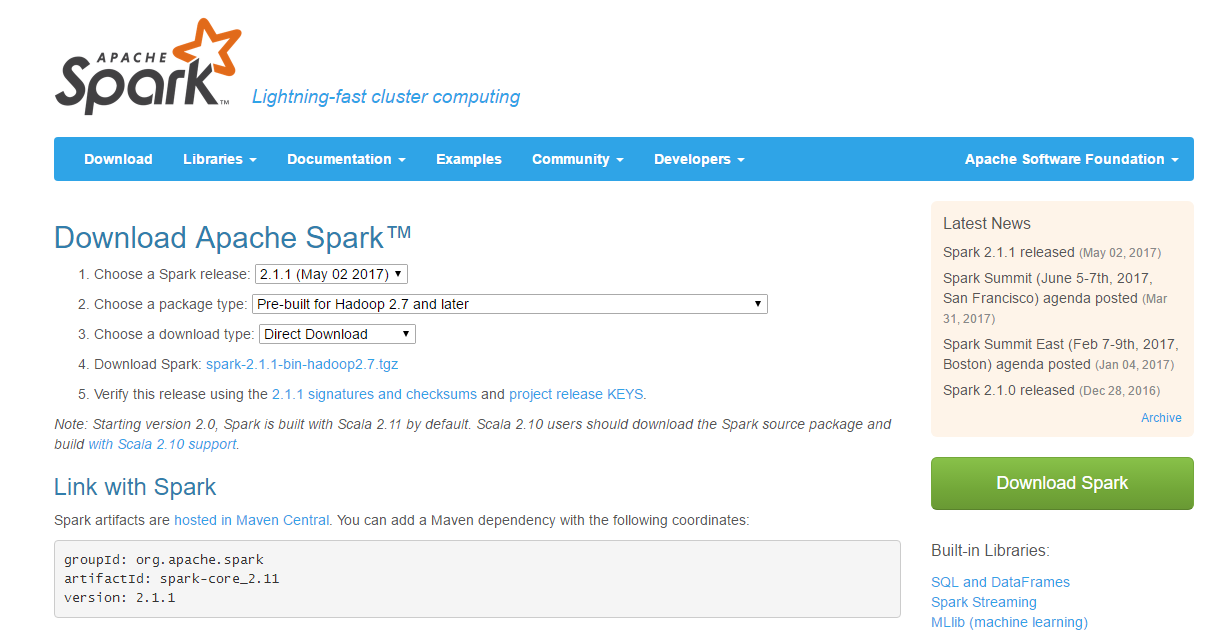
## 5.4 Documentación de instalación

Para comenzar, debemos descargar Apache Spark en la siguiente dirección: <http://spark.apache.org/>

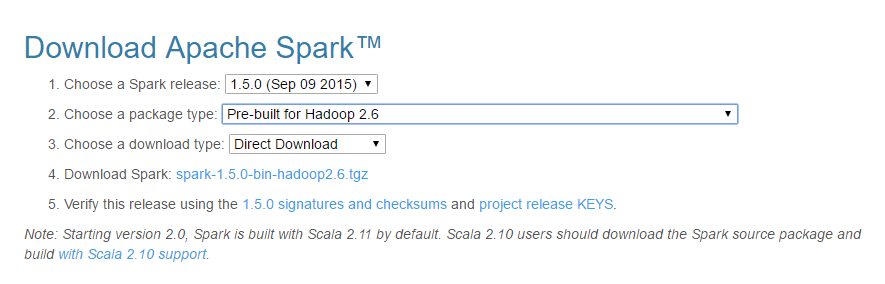
Una vez estemos en la página oficial de Apache Spark, nos dirigimos a la pestaña de descargas, como se muestra a continuación:



Una vez dentro, nos aparecerá la siguiente página:



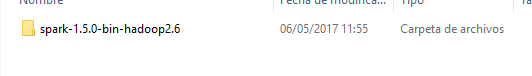
En ella, podemos ver una serie de desplegables que deberemos completar para descargar la versión de Apache Spark que deseemos:



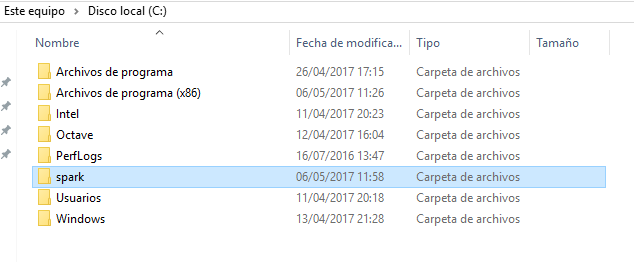
En nuestro caso, hemos elegido La versión 1.5.0 con la pre-construcción para Hadoop 2.6 en una descarga directa. Una vez hemos completado todos los campos, nos disponemos a descargar el software pulsando el hipervínculo que aparece en el paso 4 y comenzará la descarga.



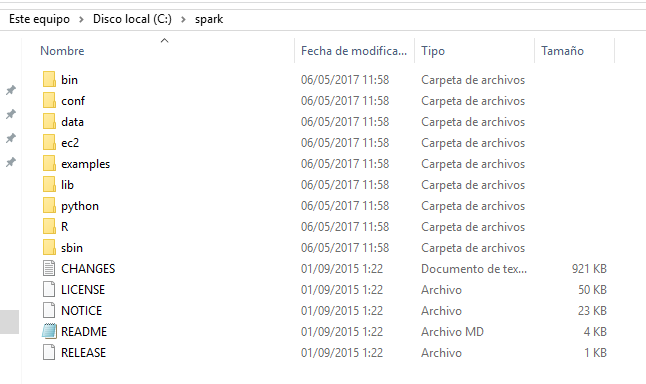
Una vez finalizada la descarga, procedemos a extraer en Spark la carpeta completa, y obtendríamos la siguiente carpeta:



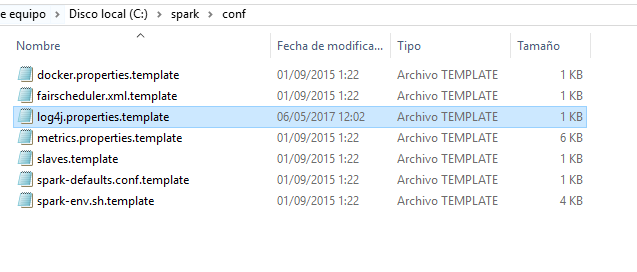
A continuación, por motivos de eficiencia y comodidad (como veremos más adelante), creamos una nueva carpeta Spark directamente en C:\.



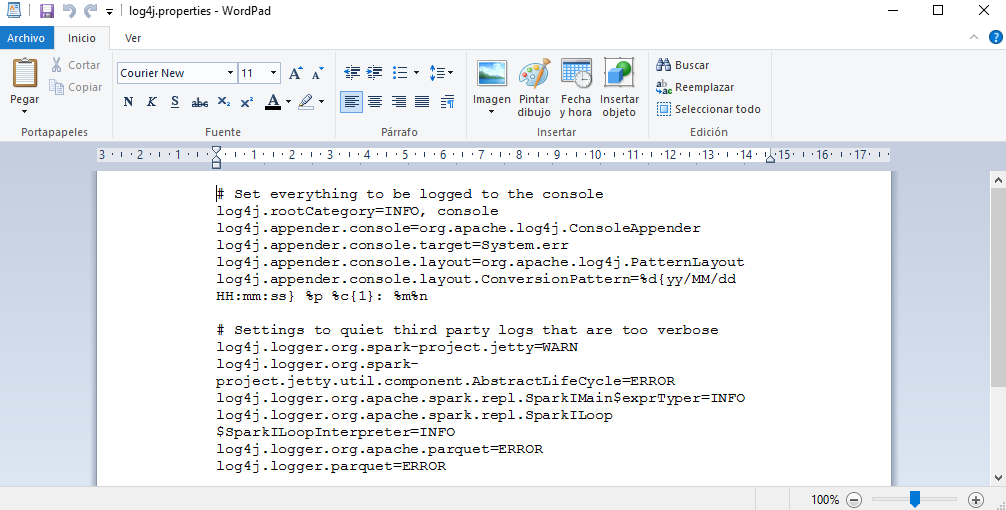
Cuando hayamos creado la carpeta, procedemos a copiar todos los archivos obtenidos en la carpeta extraída de la descarga anterior y los pegamos en la nueva carpeta spark que hemos creado en C:\.

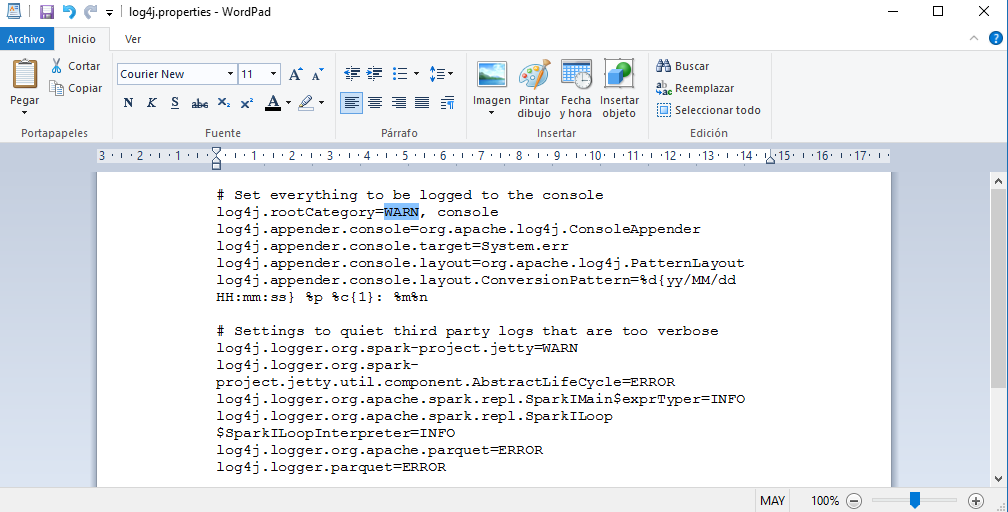


En la carpeta conf, nos disponemos a modificar el log4j para evitar futuros problemas de registro en la consola, en lugar de mostrar en todo momento como información, mostrarlo como advertencia sólo cuando sea necesario.

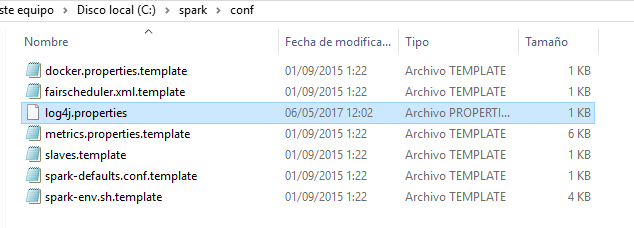


Para ello, abrimos el archivo log4j con el WordPad, por ejemplo, y en lugar de donde pone INFO, lo sustituimos por WARN, como se muestra en las siguientes imágenes y, para finalizar este paso, guardamos y cerramos el archivo.

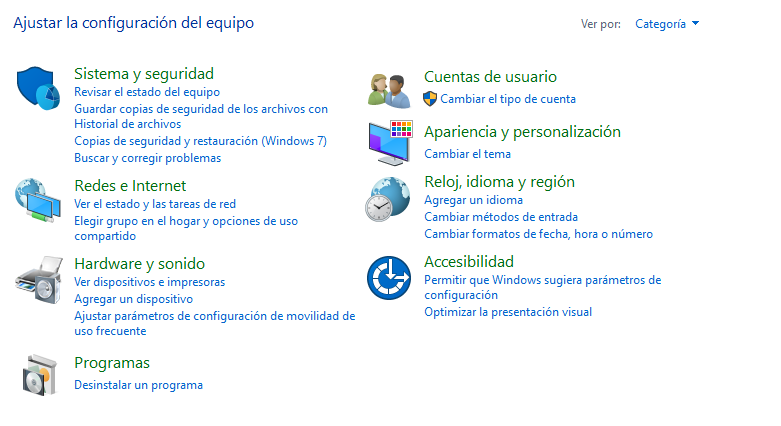




Para conseguir que este cambio tenga algún efecto, es importante no olvidar la extensión .template y dejarlo únicamente con la extensión .properties, como podemos ver a continuación:



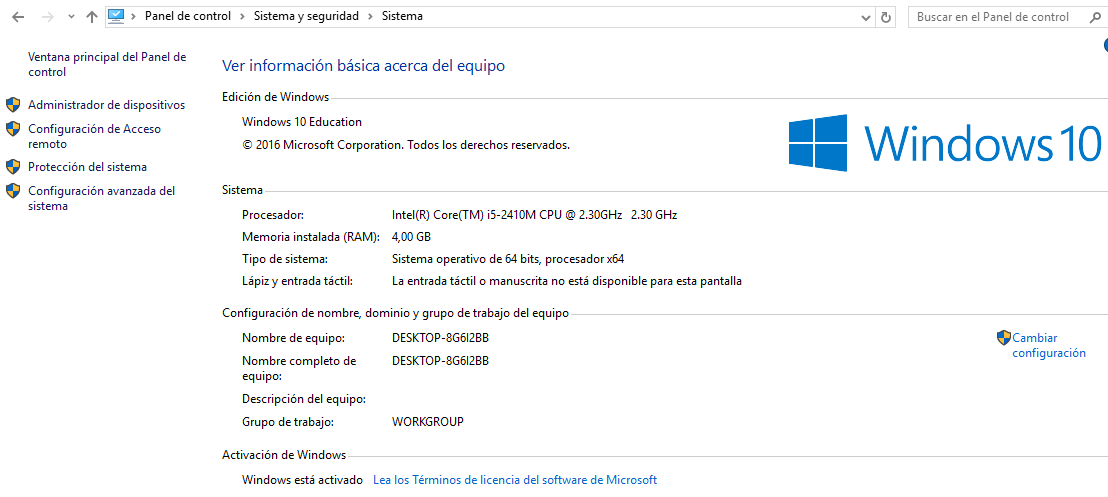
Ahora, debemos añadir Spark a las variables del entorno del sistema. Para ello nos dirigimos al Panel de Control, a la sección de Sistema y seguridad.



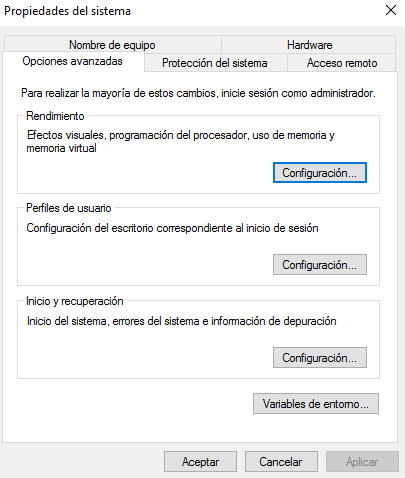
Una vez ahí, vamos a la parte de Sistema:



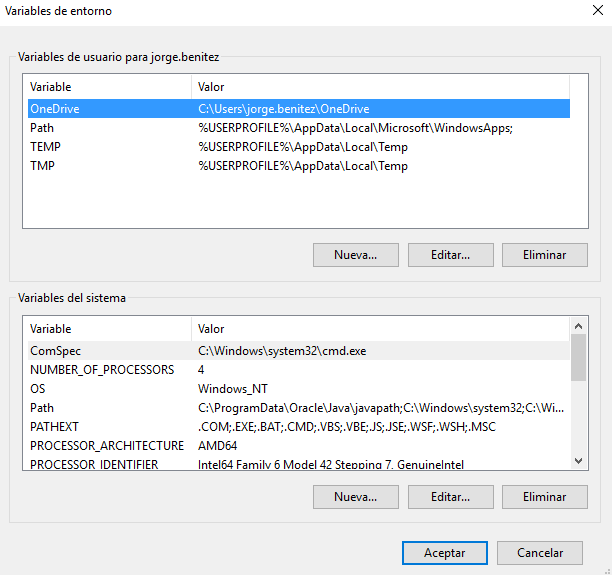
Una vez dentro, nos fijamos en la parte de la izquierda y vamos a la parte de Configuración avanzada del sistema:



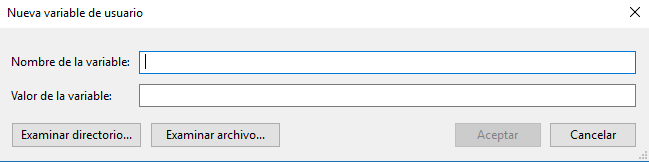
Nos aparecerá la siguiente ventana donde tendremos que pulsar en el botón Variables del entorno…:



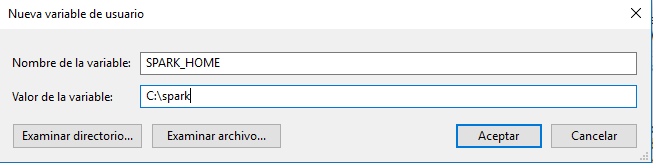
Nos aparecerá la siguiente ventada, donde debemos pulsar el botón Nueva… para poder añadir una nueva variable de entorno para Spark.



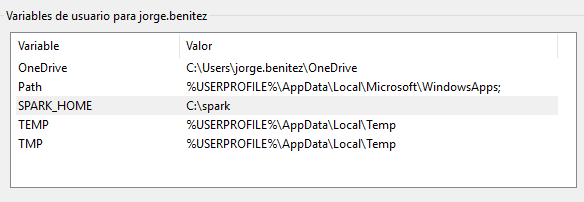
Nos aparecerá la siguiente ventana, que deberemos completar:



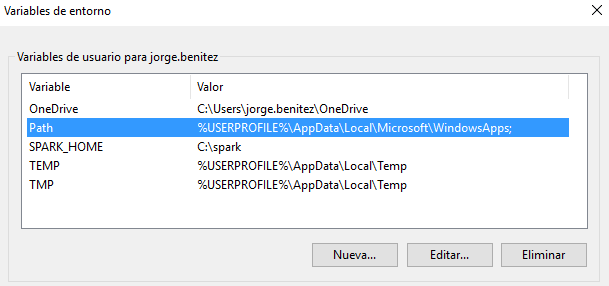
La completamos de la siguiente manera, introduciendo un nombre para la variable y la dirección donde se encuentra su valor (la localización de la carpeta creada en C:\), como se muestra a continuación:



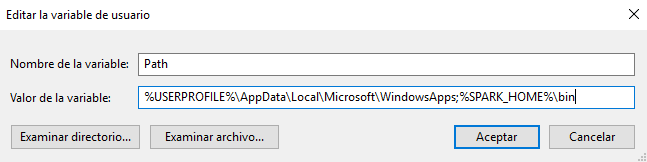
Una vez creada, se verá la nueva variable junto las existentes anteriormente.



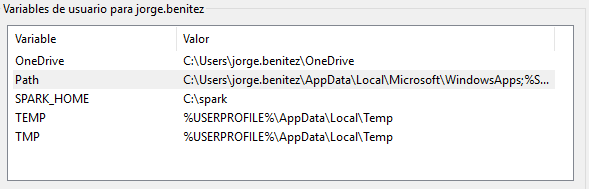
A continuación, también debemos modificar Path para que Windows pueda llamar a la nueva variable que hemos creado.



Para ello, añadimos lo siguiente:

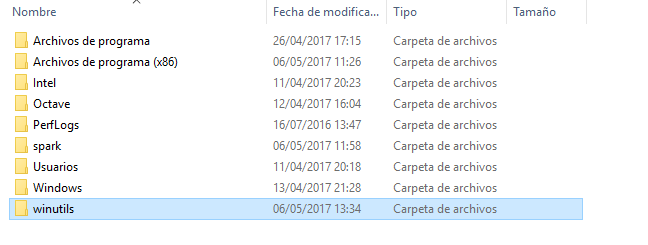


Y pulsaremos Aceptar para que se guarden los nuevos cambios realizados.

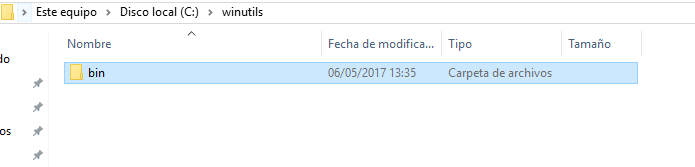


Como ya sabemos, Spark puede trabajar en conjunto con Hadoop, hasta el punto que necesitamos Hadoop para conseguir una correcta instalación. Pero como la presente práctica trata del comparar el funcionamiento por separado de ambas herramientas, debemos descargar un winutils.exe para evitar la necesidad de Hadoop en la instalación de Spark.

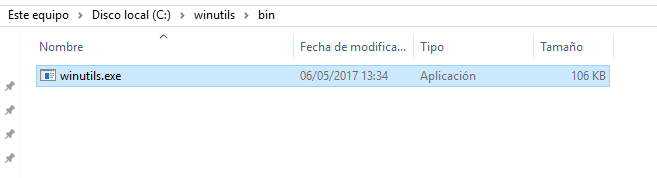
Para ello, creamos otra carpeta que llamaremos winutils en C:\, el mismo lugar donde anteriormente creamos una carpeta para Spark.



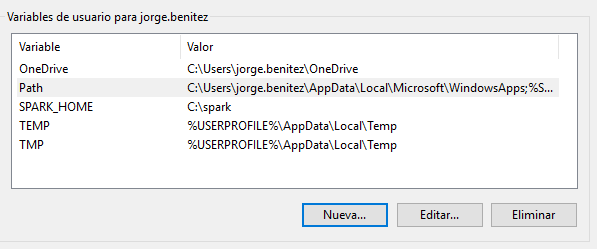
Dentro de esa carpeta, creamos otra que llamaremos bin.

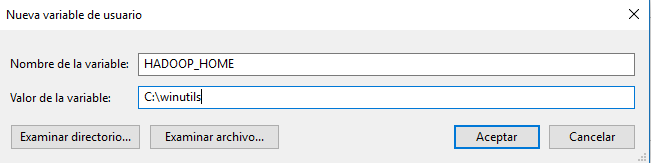


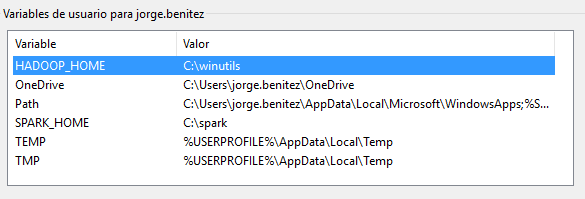
Y dentro de la carpeta bin, guardaremos winutils.exe.



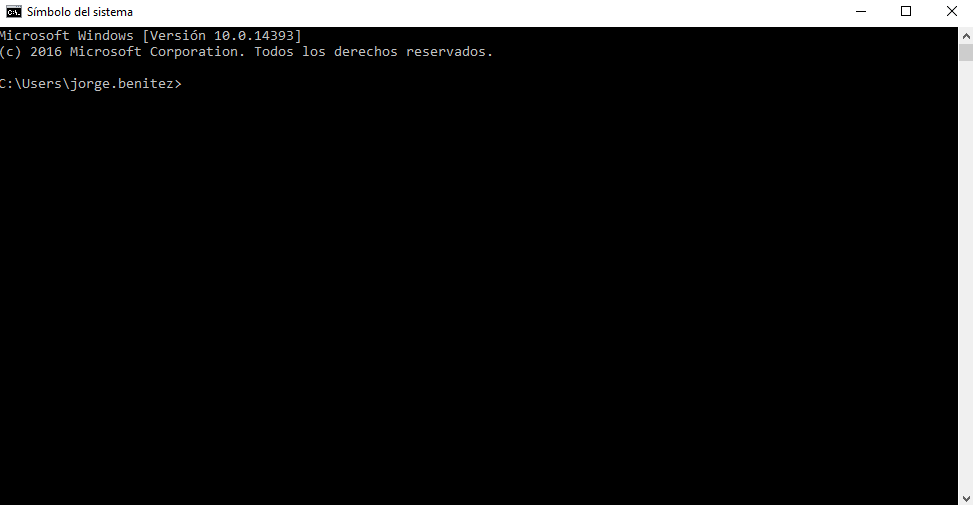
Una vez finalizados estos pasos, debemos crear una variable de entorno para winutils, al igual que hicimos anteriormente con Spark y siguiendo el mismo procedimiento, como se muestra a continuación:







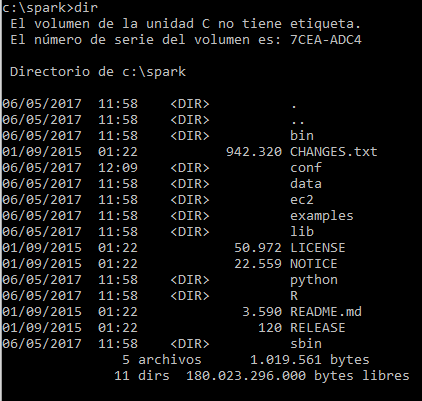
Una vez hayamos realizado todos los preparativos para la ejecución de Spark, abrimos una ventana de símbolo del sistema.



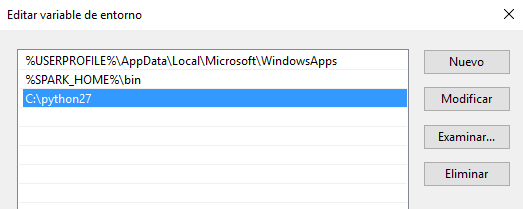
Ahora, utilizando el comando cd, nos movemos hasta la carpeta creada de Spark en C:\.

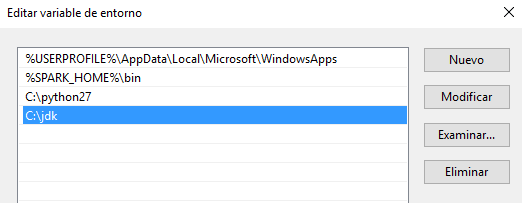


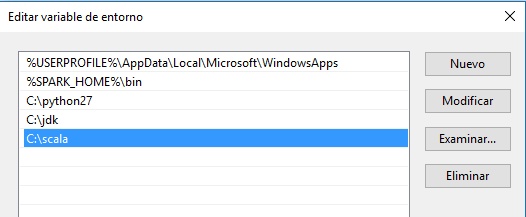
Una vez ahí, si utilizamos el comando dir, podemos ver su contenido. El resultado lo podemos ver en la siguiente imagen:



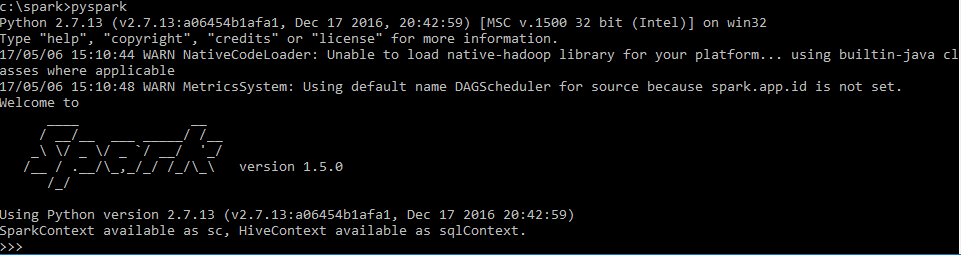
Previamente deberemos haber instalado Python 2.7 y haber hecho lo siguiente (lo mismo con el JDK y con Scala):



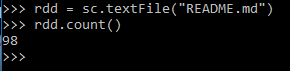




A continuación, iniciamos Spark:



Realizamos una pequeña comprobación para ver si funciona correctamente:

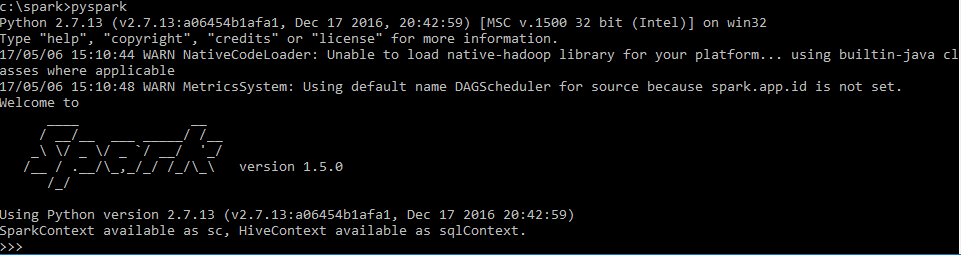


Si al introducir los 2 comandos y devuelve un número (no tiene por qué ser el 98, ya que cuenta el número de líneas que tiene el archivo) sin recibir ningún mensaje de error, podemos concluir con que Spark funciona correctamente.

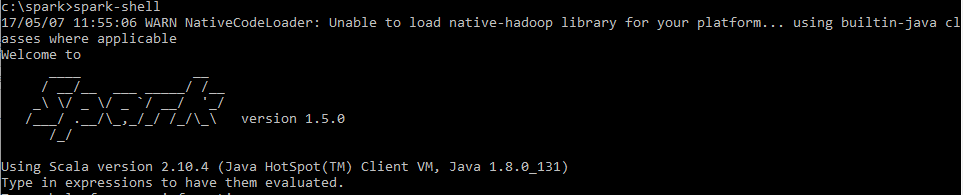
## 5.5 Manual de usuario

En primer lugar y, ante todo, como hemos mencionado en el documento de instalación, una vez instalado Spark, debemos abrir el símbolo de sistema.

Para comenzar a utilizar Spark utilizando el lenguaje de programación Python, debemos escribir el comando ***pyspark***, como se muestra a continuación:



Por otra parte, si deseamos utilizar Spark con el lenguaje de programación Scala, debemos introducir el comando ***spark-shell***:



En la presente práctica vamos a utilizar fundamentalmente el lenguaje de programación Scala para la creación del prototipo.

Pasamos a mostrar unos comandos clave para la utilización de Spark:

A continuación, hay una lista de transformaciones RDD.

|  |  |
| --- | --- |
| Nº | Transformaciones y Significado |
| 1 | **map(func)**  Devuelve un nuevo conjunto de datos distribuida, formado haciendo pasar cada elemento de la fuente a través de una función **func .** |
| 2 | **filter(func)**  Devuelve un nuevo conjunto de datos formado por la selección de aquellos elementos de la fuente en la que **func devuelve verdadero.** |
| 3 | **flatMap(func)**  Al igual que en el mapa, pero cada elemento de entrada se puede asignar a 0 o más elementos de salida (so *func* should return a Seq rather than a single item) . |
| 4 | **mapPartitions(func)**  Similar al mapa, pero se ejecuta por separado en cada partición (block) de la RDD, por lo **func debe ser de tipo Iterator <T> ⇒ Iterator <U> cuando se ejecuta en un RDD de tipo T.** |
| 5 | **mapPartitionsWithIndex(func)**  Similar al mapa de particiones, sino que también proporciona **func con un valor entero que representa el índice de la partición, de modo func debe ser de tipo (Int, Iterator<T>) ⇒ Iterator <U> cuando se ejecuta en un RDD de tipo T.** |
| 6 | **sample(withReplacement, fraction, seed)**  Pruebe una **fraction de los datos, con o sin sustitución, utilizando una semilla generador de números aleatorios dado.** |
| 7 | **union(otherDataset)**  Devuelve un nuevo conjunto de datos que contiene la unión de los elementos en el conjunto de datos de origen y el argumento. |
| 8 | **intersection(otherDataset)**  Devuelve un nuevo RDD que contiene la intersección de elementos en el conjunto de datos de origen y el argumento. |
| 9 | **distinct([numTasks])**  Devuelve un nuevo conjunto de datos que contiene los distintos elementos del conjunto de datos fuente. |
| 10 | **groupByKey([numTasks])**  Cuando se llama en un conjunto de datos de (K, V) pares, devuelve un conjunto de datos de (K, Iterable<V>) pares.  **Note - Si está agrupando a fin de realizar una agregación (such as a sum or average) sobre cada tecla, usando reduceByKey o aggregateByKey producirá un rendimiento mucho mejor.** |
| 11 | **reduceByKey(func, [numTasks])**  Cuando se llama en un conjunto de datos de (K, V) pares, devuelve un conjunto de datos de (K, V) pares donde los valores para cada clave se agregan utilizando el dado reducir la función *func* , que debe ser de tipo (V, V) ⇒ V . al igual que en groupByKey, el número de reducir las tareas se puede configurar a través de un segundo argumento opcional. |
| 12 | **aggregateByKey(zeroValue)(seqOp, combOp, [numTasks])**  Cuando se llama en un conjunto de datos de (K, V) pares, devuelve un conjunto de datos de (K, U) pares donde los valores para cada clave se agregan utilizando las funciones dadas combinar y un neutral "zero" valor. Permite un tipo de valor agregado que es diferente del tipo de valor de entrada, evitando al mismo tiempo las asignaciones innecesarias. Al igual que en groupByKey, el número de reducir las tareas se puede configurar a través de un segundo argumento opcional. |
| 13 | **sortByKey([ascending], [numTasks])**  Cuando se llama en un conjunto de datos de (K, V) pares donde implementos K pidió, devuelve un conjunto de datos de (K, V) pares ordenados por teclas en orden ascendente o descendente, como se especifica en el argumento ascendente booleana. |
| 14 | **join(otherDataset, [numTasks])**  Cuando se llama en conjuntos de datos de tipo (K, V) y (K, W) , devuelve un conjunto de datos de (K, (V, W) pares con todos los pares de elementos para cada tecla). Las combinaciones externas son apoyadas a través leftOuterJoin, rightOuterJoin y fullOuterJoin. |
| 15 | **cogroup(otherDataset, [numTasks])**  Cuando se llama en conjuntos de datos de tipo (K, V) y (K, W), devuelve un conjunto de datos de (K, (Iterable<V>, Iterable<W>) ) tuplas. Esta operación también se denomina grupo de Con. |
| 16 | **cartesian(otherDataset)**  Cuando se llama en conjuntos de datos de tipos T y U, devuelve un conjunto de datos de (T, U) pares (all pairs of elements) . |
| 17 | **pipe(command, [envVars])**  Pipe cada partición de la RDD a través de un comando shell, por ejemplo, un Perl o escritura del golpe. elementos RDD se escriben en la entrada estándar y las líneas de salida del proceso a su salida estándar se devuelven como un RDD de cadenas. |
| 18 | **coalesce(numPartitions)**  Disminuir el número de particiones en el RDD a numPartitions. Útil para operaciones en funcionamiento más eficiente después de la filtración por un gran conjunto de datos. |
| 19 | **repartition(numPartitions)**  Reordenar los datos de la RDD al azar para crear ya sea más o menos particiones y el equilibrio que a través de ellos. Esto siempre baraja todos los datos por la red. |
| 20 | **repartitionAndSortWithinPartitions(partitioner)**  Reparticionar el RDD de acuerdo con el particionador dado y, dentro de cada partición resultante, ordenar los registros de sus claves. Esto es más eficiente que llamar reparto y luego la clasificación dentro de cada partición, ya que puede empujar la clasificación hacia abajo en la maquinaria de reproducción aleatoria. |

La siguiente tabla muestra una lista de acciones, que devuelven valores.

|  |  |
| --- | --- |
| Nº | Acción y Significado |
| 1 | **reduce(func)**  Agregar los elementos del conjunto de datos utilizando una función **func (which takes two arguments and returns one).** La función debe ser conmutativa y asociativa de modo que pueda ser calculado correctamente en paralelo. |
| 2 | **collect()**  Devuelve todos los elementos del conjunto de datos como un conjunto en el programa del conductor. Esto suele ser útil después de un filtro u otra operación que devuelve un suficientemente pequeño subconjunto de los datos. |
| 3 | **count()**  Devuelve el número de elementos en el conjunto de datos. |
| 4 | **first()**  Devuelve el primer elemento del conjunto de datos (similar to take (1) ). |
| 5 | **take(n)**  Devuelve una matriz con los primeros **n elementos del conjunto de datos.** |
| 6 | **takeSample (withReplacement,num, [seed])**  Devuelve una matriz con una muestra aleatoria de **num elementos del conjunto de datos, con o sin sustitución, opcionalmente pre-especificando una semilla generadora de números aleatorios.** |
| 7 | **takeOrdered(n, [ordering])**  Devuelve los primeros **n elementos de la RDD utilizando ya sea su orden natural o un comparador personalizado.** |
| 8 | **saveAsTextFile(path)**  Escribe los elementos del conjunto de datos como un archivo de texto (or set of text files) en un directorio dado en el sistema de archivos local, HDFS o cualquier otro sistema de archivos compatible Hadoop. Chispa llama a toString en cada elemento para convertirlo en una línea de texto en el archivo. |
| 9 | **saveAsSequenceFile(path) (Java y Scala)**  Escribe los elementos del conjunto de datos como una Hadoop SequenceFile en una trayectoria dada en el sistema de ficheros local, HDFS o cualquier otro sistema de archivos compatible-Hadoop. Esto está disponible en DDR de pares de valores clave que implementan la interfaz grabable de Hadoop. En Scala, que también está disponible en los tipos que son convertir implícitamente a escribible (Spark includes conversions for basic types like Int, Double, String, etc) . |
| 10 | **saveAsObjectFile(path) (Java y Scala)**  Escribe los elementos del conjunto de datos en un formato simple usando serialización Java, que luego puede ser cargado usando SparkContext.objectFile(). |
| 11 | **countByKey()**  Sólo disponible en DDR de tipo (K, V). Devuelve un mapa hash de (K, Int) se empareja con el recuento de cada tecla. |
| 12 | **foreach(func)**  Ejecuta una función **func a cada elemento del conjunto de datos.** Esto es por lo general, hecho por efectos secundarios tales como la actualización de un acumulador o interactuar con los sistemas de almacenamiento externos.  **Note - la modificación de variables distintas de Acumuladores fuera del foreach() puede resultar en un comportamiento indefinido.** Ver cierres de entendimiento para más detalles. |

# 6. Comparación de las dos implementaciones

Se trata de dar valores a los criterios de comparación definidos en el apartado 3 sobre la implementación de cada uno de los prototipos.

## 6.1 Evaluación de los criterios en la implementación usando la tecnología Spark

Debe incluir al menos una tabla con la siguiente estructura.

| **CRITERIO** | **EVALUACIÓN** |
| --- | --- |
| Tiempo de planificación. | El tiempo de planificación fue de 2 horas. |
| Tiempo de organización. | El tiempo de organización fue de 1,5 horas. |
| Tiempo de formación en las herramientas. | El tiempo de formación en las herramientas fue de 7 horas. |
| Tiempo de preparación. | El tiempo de preparación fue de 5 horas, ya que tuvimos problemas en la configuración de las variables de entorno de la máquina virtual. |
| Tiempo de programación. | El tiempo de programación después de toda la formación que realizamos en la herramienta realmente fue relativamente pequeño, de 2 horas, debido a que Spark es una herramienta que, si estás formado, permite llevar a cabo grandes operaciones con pocas líneas de código. |
| Tiempo de pruebas. | El tiempo de pruebas duró prácticamente lo mismo que el tiempo de programación, 2 horas aproximadas en las cuales varias de ellas fueron dedicadas a los errores, donde comprobamos que Spark responde de una manera correcta y con las menores consecuencias en el resultado cuando cambiamos separadores o los omitimos por error. |
| Tiempo de obtención del resultado final. | Prácticamente el mismo que el de implementación. |
| Usabilidad | Spark responde satisfactoriamente a las expectativas que teníamos de usabilidad después de la realización de los otros trabajos (ALTA). |
| Adaptación. | La adaptación es realmente buena (ALTA). |
| Compresión. | La comprensión es ALTA debido a que la simpleza del lenguaje Scala es muy alta. |
| Lenguaje de programación. | Como lenguaje de programación se utiliza Scala. |
| Seguridad. | La seguridad es complicada de implementar. |
| Tolerancia a fallos. | La tolerancia a fallos es muy alta. |
| Rendimiento del programa. | El rendimiento del programa es óptimo. |
| Escalabilidad. | Es muy escalable, si duplicamos el tamaño del archivo no tenemos ningún problema, ya que los tiempos siguen siendo prácticamente iguales. |
| Modularidad. | Alta, las funciones están claramente definidas y cada una tiene una dedicación especifica. |
| Eficacia. | Alta |
| Eficiencia | Alta |

Y algunos comentarios aclaratorios sobre aquellos criterios cuyo valor indicado en la tabla no sea suficiente para entenderlo.

## 6.2 Evaluación de los criterios en la implementación usando la tecnología Hadoop (MapReduce).

| **CRITERIO** | **EVALUACIÓN** |
| --- | --- |
| Tiempo de planificación. | El tiempo de planificación fue de 2 horas. |
| Tiempo de organización. | El tiempo de organización fue de 2 horas. |
| Tiempo de formación en las herramientas. | El tiempo de formación en las herramientas fue de 14 horas. |
| Tiempo de preparación. | El tiempo de preparación fue de 7 horas, ya que hay que hacer ciertas configuraciones en la máquina virtual que complican la instalación de ciertas librerías y el acceso a HDFS. |
| Tiempo de programación. | El tiempo de programación después de toda la formación que realizamos en la herramienta realmente fue bastante alto, ya que es un framework muy complejo de implementar mediante java y la longitud del código es extensa. |
| Tiempo de pruebas. | El tiempo de pruebas fue de 5 horas, ya que se complicó un aspecto del acceso a HDFS. |
| Tiempo de obtención del resultado final. | Prácticamente el mismo que el de implementación. |
| Usabilidad | MapReduce es bastante complicado de usar y requiere bastante más formación que otras tecnologías (ALTA). |
| Adaptación. | La adaptación es realmente buena (ALTA). |
| Compresión. | La comprensión del lenguaje pese a ser Java se complica bastante. |
| Lenguaje de programación. | Como lenguaje de programación se utiliza Java. |
| Seguridad. | La seguridad es automáticamente aplicada debido a que el framework es nativo de HDFS. |
| Tolerancia a fallos. | La tolerancia a fallos media. |
| Rendimiento del programa. | El rendimiento del programa es bueno, ya que procesa grandes cantidades de datos. |
| Escalabilidad. | Es escalable, ya que si duplicamos el número de líneas que contiene el archivo a analizar. |
| Modularidad. | Baja, este código es relativamente pequeño, pero en proyectos más grandes se puede complicar. |
| Eficacia. | Media |
| Eficiencia | Media |

# 7. Comparación de la implementación de las tecnologías

Debe incluir al menos una tabla resumen, en sección de página horizontal, cruzando los criterios y los valores de cada tecnología. Con una columna de comentarios sobre la comparación

| **CRITERIOS** | **Spark** | **TECNOLOGÍA B** | **COMENTARIOS** |
| --- | --- | --- | --- |
| Tiempo de planificación. | El tiempo de planificación fue de 2 horas. | El tiempo de planificación fue de 2 horas. | Los tiempos son prácticamente iguales, ya que todavía no tuvimos que lidiar con las diferentes complejidades de las tecnologías. |
| Tiempo de organización. | El tiempo de organización fue de 1,5 horas. | El tiempo de organización fue de 2 horas. | Los tiempos son prácticamente iguales, ya que todavía no tuvimos que lidiar con las diferentes complejidades de las tecnologías. |
| Tiempo de formación en las herramientas. | El tiempo de formación en las herramientas fue de 7 horas. | El tiempo de formación en las herramientas fue de 14 horas. | Aquí vemos como MapReduce supera con creces a Spark, ya que la herramienta es infinitamente más complicada. |
| Tiempo de preparación. | El tiempo de preparación fue de 5 horas, ya que tuvimos problemas en la configuración de las variables de entorno de la máquina virtual. | El tiempo de preparación fue de 7 horas, ya que hay que hacer ciertas configuraciones en la máquina virtual que complican la instalación de ciertas librerías y el acceso a HDFS. | En ambas hubo que hacer configuraciones, los tiempos son parecidos, un poco mayores en MapReduce, pero debido a que nos atascamos con ciertos aspectos, por lo demás no destaca un tiempo sobre el otro. |
| Tiempo de programación. | El tiempo de programación después de toda la formación que realizamos en la herramienta realmente fue relativamente pequeño, de 2 horas, debido a que Spark es una herramienta que si estás formado, permite llevar a cabo grandes operaciones con pocas líneas de código. | El tiempo de programación después de toda la formación que realizamos en la herramienta realmente fue bastante alto, 10 horas ya que es un framework muy complejo de implementar mediante java y la longitud del código es extensa. | El tiempo de programación si es mucho más grande en el caso de MapReduce, debido a que con un número mucho más reducido de líneas de código Spark puede hacer lo mismo. |
| Tiempo de pruebas. | El tiempo de pruebas duró prácticamente lo mismo que el tiempo de programación, 2 horas aproximadas en las cuales varias de ellas fueron dedicadas a los errores, donde comprobamos que Spark responde de una manera correcta y con las menores consecuencias en el resultado cuando cambiamos separadores o los omitimos por error. | El tiempo de pruebas fue de 5 horas, ya que se complicó un aspecto del acceso a HDFS. | En este caso tuvimos un problema con el acceso al sistema de archivos en MapReduce, lo que complicó un poco las pruebas de esta tecnología. |
| Tiempo de obtención del resultado final. | Prácticamente el mismo que el de implementación. | Prácticamente el mismo que el de implementación. |  |
| Usabilidad | Spark responde satisfactoriamente a las expectativas que teníamos de usabilidad después de la realización de los otros trabajos (ALTA). | MapReduce es bastante complicado de usar y requiere bastante más formación que otras tecnologías (ALTA). | Como podemos comprobar la usabilidad de Spark es mucho mayor que la de MapReduce. |
| Adaptación. | La adaptación es realmente buena (ALTA). | La adaptación es realmente buena (ALTA). | En ambas la adaptación fue satisfactoria. |
| Compresión. | La comprensión es ALTA debido a que la simpleza del lenguaje Scala es muy alta. | La comprensión del lenguaje pese a ser Java se complica bastante. | La comprensión de Scala es mucho más sencilla que la de java, a parte Spark tiene funciones que engloban varias líneas de código en MapReduce. |
| Lenguaje de programación. | Como lenguaje de programación se utiliza Scala. | Como lenguaje de programación se utiliza Java. |  |
| Seguridad. | La seguridad es complicada de implementar. | La seguridad es automáticamente aplicada debido a que el framework es nativo de HDFS. | La seguridad requiere mucho más trabajo implementarla en Spark que en MapReduce. |
| Tolerancia a fallos. | La tolerancia a fallos es muy alta. | La tolerancia a fallos media. | Spark supera a MapReduce en tolerancia a fallos |
| Rendimiento del programa. | El rendimiento del programa es óptimo. | El rendimiento del programa es bueno, ya que procesa grandes cantidades de datos. | Spark tiene mucho mayor rendimiento en sus tiempos para un mismo archivo que MapReduce. |
| Escalabilidad. | Es muy escalable, si duplicamos el tamaño del archivo no tenemos ningún problema, ya que los tiempos siguen siendo prácticamente iguales. | Es escalable, ya que si duplicamos el número de líneas que contiene el archivo a analizar. | La escalabilidad de Spark es muy alta y la de MapReduce simplemente alta. |
| Modularidad | Alta, las funciones están claramente definidas y cada una tiene una dedicación especifica. | Baja, este código es relativamente pequeño, pero en proyectos más grandes se puede complicar. | La modularidad de Spark es mayor por los temas de englobar funciones que hemos comentado anteriormente |
| Eficacia | Alta | Media |  |
| Eficiencia | Alta | Media |  |

# 8. Conclusiones

Como conclusión y observando las tablas anteriormente expuestas podemos deducir que la integración de MapReduce con Hadoop siempre va a ser superior que la de Spark con Hadoop, ya que como bien sabemos y hemos expuesto en trabajos anteriores Spark es una herramienta externa a Hadoop por lo tanto tendremos que configurar los parámetros de seguridad y de integración entre ambas herramientas de la mejor manera posible.

Sin embargo, también podemos observar por los datos aportados que Spark es inmensamente más potente que MapReduce.

Por lo tanto, si necesitamos integración acudiremos a MapReduce, pero sin embargo si necesitamos potencia de procesamiento y velocidad (Lo cual es lo más habitual en proyectos Big data), deberíamos recurrir a Spark sin dudarlo.

---------------------------

(Hay que cumplir la estructura básica indicada de secciones. Pero si se desea se pueden añadir otras secciones como anexos. Por ejemplo, alguna encuesta de opinión realizada sobre las tecnologías, etc.)