uc3m | Universidad Carlos III de Madrid

**Grado en Ingeniería Telemática 2017-2018.**

***Trabajo Fin de Grado***

# ANÁLISIS TECNOLÓGICO DEL DISEÑO, DESPLIEGUE Y PUESTA EN MARCHA DE SOLUCIONES *BIG DATA* EMPLEANDO APACHE SPARK


**Pedro Javier Martín-Doimeadios Pozo**

**Tutor**

**Julio Villena Román**


**Leganés, Octubre de 2018**

# ANÁLISIS TECNOLÓGICO DEL DISEÑO, DESPLIEGUE Y PUESTA EN MARCHA DE SOLUCIONES *BIG DATA* EMPLEANDO APACHE SPARK

`

Autor: Pedro Javier Martín-Doimeadios Pozo

Tutor: Julio Villena Román

Departamento de Ingeniería Telemática

Escuela Politécnica Superior

Universidad Carlos III de Madrid

Octubre de 2018

# Resumen

Este proyecto está centrado en los desafíos que se tienen hoy en día y que sin duda se tendrán en los próximos años, ya que es un mercado emergente, aunque ya se ve el despunte. Todo este proceso de cambio se debe en parte a la digitalización del mundo en general. Esta digitalización de todos los servicios, de todas las empresas y sectores ha hecho que haya una necesidad de almacenar los datos. De este almacenamiento de dados nace la tecnología *big data*.

De este almacenamiento produce una reacción de nuevas tecnologías como el Internet de las cosas (IoT) o el machine learning, ya que se obtienen datos de todo tipo de plataformas y aplicaciones, tanto móviles que han experimentado una evolución y aumento de unidades significativo, como dispositivos en los edificios públicos con los que se pretende mejorar servicios o en la navegación web en la que nuestros datos son usados para una mejora de los servicios que aplican las empresas sobre sus clientes.

Por ello este proyecto se pretende desplegar y diseñar un sistema *big data* en un tres entornos diferentes, para ello se hará uso de Apache Spark, donde tendremos un entorno distribuido con dos nodos, un entorno pseudo-distribuido con dos nodos y un entorno Cloud basado en Apache Spark, como es Databricks.

Para la realización de este trabajo, será necesario tener unos requisitos para montar el entorno que se describirán más adelante. En este proyecto se estudiará soluciones a un entorno real, como son las multas de Madrid. Por lo que estas multas deberán ser tratadas y de este tratamiento, obtendremos unos resultados que se analizarán.

**Palabras claves:** *Big Data, Apache Spark, Apache Hadoop, Cloud, Optimización.*

## Acrónimos

**CSV** Comma Separated Value. 47, 60, 61

**DAG** Direct Acyclic Graph. 19, 21, 57, 102

**HDFS** Hadoop Distributed File System. 16, 17, 18, 26, 33, 47, 83, 100

**GC** Garbage Collector. 79, 80

**IP** Internet Protocol. 31, 36, 44, 67

**JVM** Java Virtual Machine. 77, 79, 80

**PIP** Python Installer Packages. 31

**RAM** Random Access Memory. 21, 40, 41

**RDD** Resilient Distributed Dataset. 19, 20, 57, 59, 67, 102

**RGPD** Reglamento General de Protección de Datos. 4, 11

**SACE** Servicio de Apoyo al Control de Estacionamiento. 62, 64, 66

**SER** Servicio de Estacionamiento Regulado. 62, 63, 64

**SQL** Structured Query Language. 18, 21

**SO** Sistema Operativo. 25, 27, 28, 29, 30

**SSH** Secure Shell. 29, 31, 35, 44

**USB** Universal Serial Bus. 28

**URL** Uniform Resource Locator. 4, 77, 80

# Índice General

# Índice de tablas

# Índice de imágenes

# 1

# Introducción

En los años noventa, el informático teórico estadounidense John Mashey publicó un artículo titulado *Big data and the Next Wave of Infrastress* (*Big data y la próxima ola de Infrastress*), popularizando el término que hoy nos ocupa. En él, Mashey hacía referencia al estrés que iban a sufrir las infraestructuras físicas y humanas de la informática debido al imparable tsunami de datos que ya se oteaba en el horizonte, inmanejable con los instrumentos de gestión al uso. Desde luego no se equivocaba. Hoy, recién iniciado el siglo XXI, se generan, según la Unión Europea, 1.700 nuevos billones de *bytes* por minuto [5].

Los grandes datos o grandes volúmenes de datos han ido creciendo de modo espectacular. Actualmente, se generan 2,5 trillones de datos cada día, ocupando 2,7 zetabytes (ZB) de información en el universo de datos. Esta cantidad espera dejar cifras insospechadas hasta el momento, estimando llegar a los 44 ZB [1]. Con este volumen de datos es de vital importancia el saber cómo almacenar, cómo saber tratar con este volumen de datos. Para ello cada vez existen más plataformas y herramientas para almacenar, tratar y comprender estos datos, provenientes de diversos puntos.

En la actualidad las tecnologías *Big data* se están utilizando en un amplio abanico de sectores que van desde la Salud hasta la Agricultura. Desde que disponemos datos

digitales ha habido una necesidad de inferir conocimiento relevante a partir de dichos datos, y ahora con la explosión de los datos generados por las aplicaciones móviles, redes sociales e Internet de las Cosas, se hace imprescindible realizar esta extracción de conocimiento de una forma eficiente, rápida y asequible [2].

A pesar de que el término *Big data* se asocia principalmente con cantidades de datos exorbitantes, se debe dejar de lado esta percepción, pues *Big data* no va dirigido solo a gran tamaño, sino que abarca tanto volumen como variedad de datos y velocidad de acceso y procesamiento. En la actualidad se ha pasado de la transacción a la interacción, con el propósito de obtener el mejor provecho de la información que se genera minuto a minuto. [3] Esta variedad puede ir englobada a distintos sectores como hemos hablado brevemente anteriormente[2], pero cabe recalcar que aparte de hacer uso de la información generada en sectores concretos como podrían ser marketing digital, telefonía, salud, etc. Hay una variante a tomar en cuenta como es la de procesos internos en grandes empresas para mejorar la productividad de la empresa, como podría ser lo denominado Industria 4.0.

Como dice Mashey, esta ingente cantidad de datos, produce una gran fuente de retos a la hora de almacenar, tratar o transformar. Se ha llegado a denominar el *Big data* como industria 4.0, por lo que estos acontecimientos nos ha llegado en el momento más importante en los últimos años, con lo que este incentivo hace que sea de gran interés el estudio de las nuevas tecnologías y herramientas para entender la nueva revolución que se está formando en todo el mundo.

A la hora de hablar de *Big data*, no sólo debemos centrarnos y darle exclusividad del uso de esta tecnología a entidades privadas que obtienen beneficios del tratamiento, almacenamiento y procesado de los datos. No hay que obviar que existe una revolución en torno a las ciudades, podemos hablar de lo que se denomina hoy en día *smart cities*. Toda la ciudad se asume como un ordenador (en palabras del programador Paul McFedries la calle y su panorámica es el interfaz, tú eres el cursor y tu teléfono inteligente es el aparato receptor) y es computable en datos [4].

En este proyecto, veremos cómo instalar, configurar y así poder comparar soluciones en entorno Apache Spark, para ello, veremos cómo se configura, mejoras para optimizar los entornos y para ello, usaremos una fuente de datos amplia y de acceso libre como es un conjunto de datos de multas que se producen en Madrid, en la que tenemos acceso con total libertad.

En este proyecto no se centrará en la parte informática exclusivamente, sino en los distintos formatos y su optimización a la hora de buscar una solución, para ello pondremos énfasis en un escenario real, y poniendo una visión de una consulta que puede ser de ayuda en un entorno real.

## 1.1 Objetivos

El objetivo principal en este proyecto es el de obtener posibles soluciones a problemas reales con distintos entornos *big data*, así como sus posibles optimizaciones y discusión en un escenario real.

- Análisis y estudio del panorama y de las herramientas big data disponibles en la actualidad, centrándose en Apache Spark.
- Diseñar sistema big data basándose en Apache Spark tanto en entorno pseudo distribuido, como distribuido, así como en Cloud.
- Optimizar el entorno Apache Spark para una mejora en tiempos.
- Diseñar e implementar scripts de consultas para la obtener resultados.
- Aplicar los scripts y consultas realizadas a un escenario real y obtener una discusión real de posibles soluciones conclusiones de un problema real.
- Analizar los nuevos retos que nos aporta la obtención de datos y el uso de estos datos.
- Obtención de conclusiones y futuras líneas de trabajo.

## 1.2 Estructura del documento

La estructura de esta memoria es la siguiente:

1. **Introducción:** Breve descripción en la que se presenta el contexto en el que se realiza el proyecto, además de la motivación, los objetivos y estructura del documento. Apartado 1.
2. **Estado del Arte:** En este apartado nos centraremos en el contexto del proyecto y en las tecnologías y herramientas para solucionar posibles problemas asociadas al desarrollo del proyecto, en este caso nos centraremos en la parte estructural y de almacenamiento en tecnologías *big data* en entorno Apache Spark. Además hablaremos de la legislación que existe sobre la implementación realizada en el

trabajo, estándares técnicos y de la RGPD (Reglamento General de Protección de Datos).Apartado 2.

3. **Diseño:** En éste punto podremos ver la implementación, requisitos básicos y los tres entornos en lo que se ha centrado este proyecto para la comparación entre ambos. Apartado 3.
4. **Análisis de los Ficheros:** Descripción de un escenario real, con análisis de consultas y cómo poder afrontarlo en la vida real. Apartado 4.
5. **Configuración Apache Spark:** Configuraciones para optimizar el entorno de trabajo, en el hablaremos de propiedades importantes para desarrollar dichas mejoras. Apartado 5.
6. **Conclusiones y Trabajos Futuros:** Se hará una retrospectiva sobre el trabajo realizado y se verá las líneas posibles de trabajo. Apartado 6.
7. **Impacto Socioeconómico:** Se analizará el posible impacto económico del proyecto si se realizara, y a su vez posteriormente, se detallará presupuesto y coste del mismo. Apartado 7.
8. **Planificación y presupuesto:** Se mostrará la planificación para la realización del proyecto y del presupuesto por si se llevara a cabo. Apartado 8.
9. **Extended Abstract:** Resumen del proyecto en inglés. Apartado 9
10. **Bibliografía:** Se detallarán los enlaces URLs, libros y artículos usados para poder obtener la información necesaria para la realización del proyecto. Apartado 10.
11. **Anexo:** Se alojarán en este apartado, distintas consultas para la realización de pruebas en el escenario que hemos hablado en el apartado Análisis de los Ficheros. Apartado 11.

# 2

# Estado del arte

En este apartado, se analizan las tecnologías para llevar a cabo el proyecto, así como el contexto y distintas herramientas que rivalizan con las que se han usado. Para ello, veremos cómo hacer uso de los datos obtenidos de manera abierta.

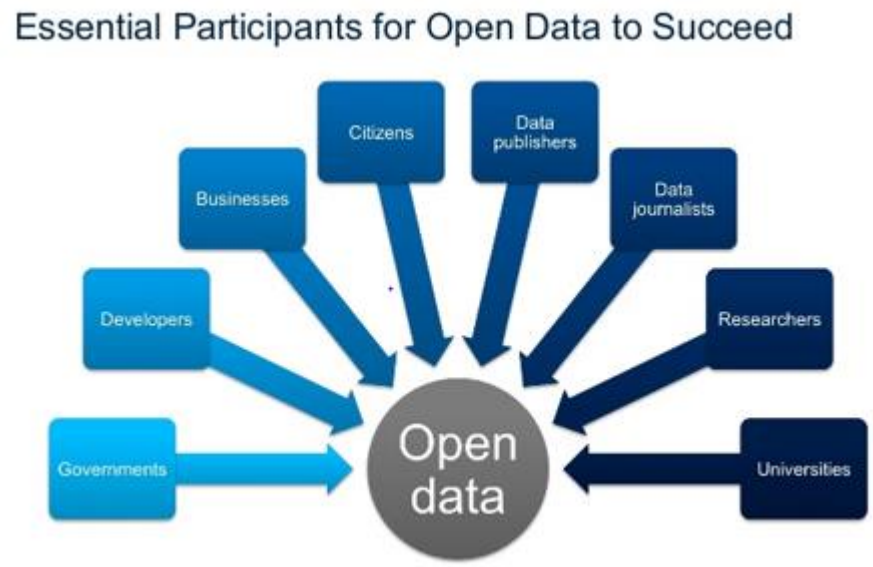
También se realizará un análisis del *big data*, su estructura, herramientas on premise o cloud que nos puedan aportar una visión del *big data*. El uso de los sistemas de almacenamiento usados, así como los de otras herramientas.

Para tener más conceptos de la tecnología usada, y al hacer uso de datos abiertos del portal de datos abiertos del Ayuntamiento de Madrid [21], se hablará del significado de datos abiertos u Open Data. El Open Data son datos que pueden ser utilizados, reutilizados y redistribuidos libremente por cualquier persona, y que se encuentran sujetos, cuando más, al requerimiento de atribución y de compartirse de la misma manera en que aparecen [20]. Esta apertura de datos tiene tres premisas: disponibilidad y acceso, reutilización y distribución de datos y participación universal por parte de todos los integrantes.

En la actualidad hay un gran número de organismos gubernamentales y no gubernamentales, estos actores tienen plataformas de acceso de datos abiertos sobre

distintos aspectos de la sociedad, como son datos demográficos o económicos. En la siguiente imagen veremos los participantes esenciales a la hora tener acceso a todos los datos abiertos, en ella se puede observar las distintas fuentes de generación de datos por parte de la sociedad [22].

Imagen 2.1 Participantes Esenciales [22]




# 2.1 *Big data*

## 2.1.1 Introducción

Se cree que la definición de *Big data* como tal viene del científico de computación jefe en Silicon Graphics, John Mashey, quien en 1998 presentó unas diapositivas que representaban una gran conciencia del volumen de datos [23].

Desde que ha habido escritura cuneiforme, la más antigua de las maneras de escritura que existen hoy en día, hasta los servidores de datos que almacenan estos, los seres humanos siempre hemos tenido la necesidad de almacenar todo tipo de datos. Como dato a tener en cuenta, se prevé que en 2020 se hayan creado 40 zettabytes de datos o lo que es lo mismo $4 \cdot 10^9$ Terabytes o 40.000.000.000 Terabytes

Haciendo una recopilación histórica de datos, podríamos decir que desde casi el origen del ser humano hemos tenido la necesidad de comunicarnos, con ello se ha desarrollado una intuición o característica innata en la humanidad de almacenar

información como fue en las tribus del paleolítico sobre el año 16.000 a.C. donde se hacían muescas en palos y huesos para seguir las reservas de alimento o agua.

Desde hace miles de años el ser humano tiende a almacenar, como podría ser los manuscritos de muchas bibliotecas, museos, etc. Es en la actualidad cuando por ejemplo en 1928 el ingeniero alemán Fritz Pfleumer patenta el primer sistema magnético para almacenar datos o en 1965 se proyecta el primer *data center* en Estados Unidos, para guardar documentación de impuestos y huellas dactilares en cintas magnéticas. [24]

### 2.1.2 Qué es el *Big data*

Par definir el término no se puede seguir un único camino, no existe un término que genera un consenso en toda la sociedad. Por ejemplo según la Unión Europea al *big data* se refiere a grandes cantidades de datos producidos muy rápidamente por un gran número de diversas fuentes. Los datos pueden ser creados por personas o generados por máquinas, como sensores que recopilan información climática, imágenes satelitales, imágenes y vídeos digitales, registros de transacciones de compras, señales de GPS, etc. [26]. Aunque según el uso de esta tecnología, la siguiente definición se podría considerar más cercana a la tecnología estudiada en este proyecto, dado que da al término *Big data* como un término común bajo el que se agrupan toda clase de técnicas de tratamiento de grandes volúmenes de datos, fuera de los análisis y herramientas clásicas [25].

Desde que se creó el término *big data*, se han buscado todo tipo de características y propiedades específicas en esta tecnología. Primero se especificaron tres características: velocidad, variedad y volumen. Pero con el paso del tiempo se han ido añadiendo llegando a las 10 características. A continuación se hablará de estas 10 características. [27]:

- **Variedad** se trata de cómo nos llegan los datos, ya sean estructurados, desestructurados o semiestructurados, siendo los datos desestructurados los más frecuentes.
- **Velocidad** es la velocidad a la que los datos se generan, producen, crean o actualizan.
- **Volumen**, la característica más conocida de esta tecnología. Como hemos hablado al comienzo del documento, se generan 2.5 trillones de datos cada día. Su almacenamiento es uno de las características más importantes.

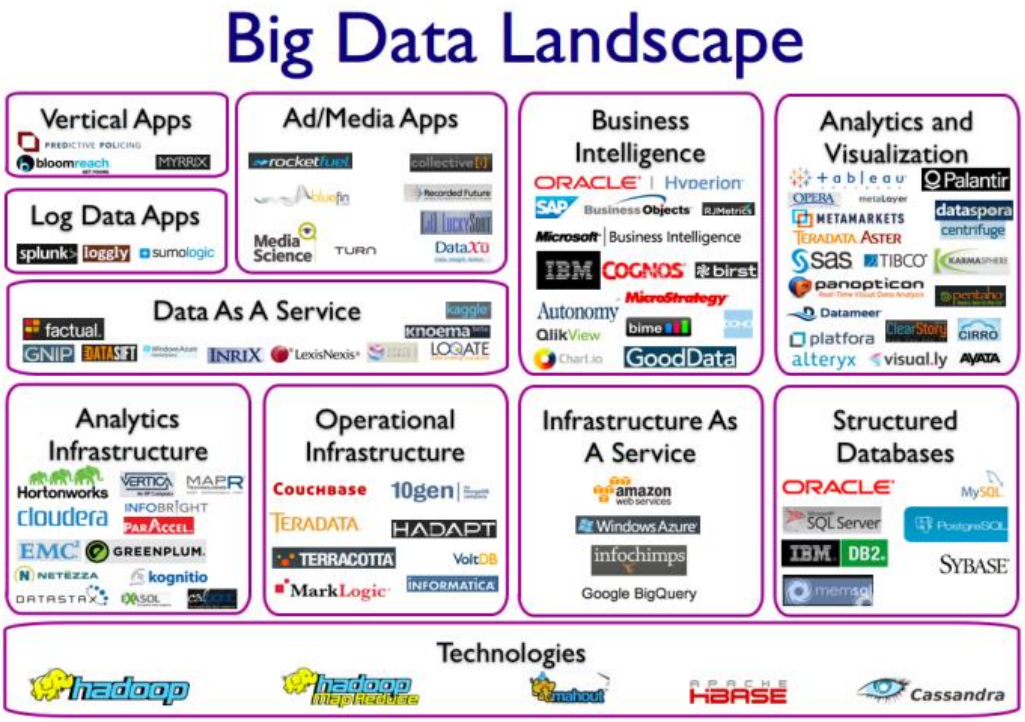
- **Variabilidad** los diferentes tipos de datos y sus fuentes hacen que haya una considerable desviación de los tipos de datos, dando pie a una mayor variedad de estos.
- **Veracidad**. Se refiere a la procedencia o a la confianza del origen de los datos y de su contexto.
- **Validez**. Está relacionado a cuán precisos son los datos.
- **Vulnerabilidad**. Se refiere a la seguridad que tienen los datos, ya que puede ocurrir que haya filtraciones o robos.
- **Volatilidad**. Esta característica tiene en cuenta que los datos históricos no podrán almacenarse eternamente, por lo que los datos generados o ya existentes deben ser eliminados en algún momento.
- **Visualización**. Se refiere a la capacidad de hacer visible los datos.
- **Valor**. Los datos tienen un valor ya que se puede llegar a comprender mejor a los clientes, orientarlos en consecuencia, optimizar los procesos y mejorar el rendimiento de la máquina o del negocio.

### 2.1.3 Fuente de negocios

A día de hoy se puede ver que las tecnologías *big data* son en realidad ventajas en el ámbito empresarial y público. También es una evolución del campo de Analytics, ya que al almacenar y tratar millones de datos, se pueden obtener ventajas para la investigación y aplicación de distintos campos.

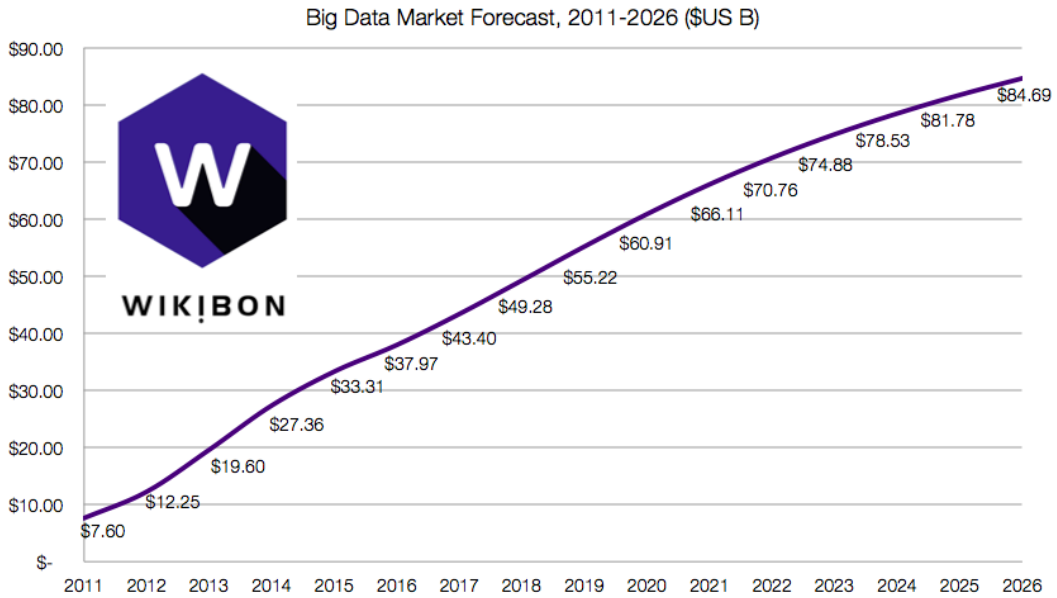
Por ello el paisaje de empresas o "landscape" ha ido incrementado con las nuevas empresas que están destinadas al estudio y creando nuevas empresas para el desarrollo de nuevas herramientas. A continuación se puede ver en la imagen 2.2 este "landscape".

Imagen 2.2 Landscape de tecnologías *big data* [28]




Respecto al volumen comercial que mueve el *big data*, podemos ver en la tabla 2.1[28] cómo ha llegado a incrementarse en un 50% los ingresos entre los años 2011-2012 y los años 2012-2013 y que para el año 2018 se tiene previsto en torno a 50 Billones de $.

Imagen 2.3 Proyección ingresos del *big data* [44]



### 2.1.4 Marco Regulador

El *Big data* como concepto se refiere al análisis de información en cantidades industriales, todo un universo de oportunidades para las empresas aún por explorar. No existe una traducción unívoca (¿grandes datos? ¿Datos masivos?), y muchas veces el término se utiliza de forma incorrecta. [6] Por esta razón, para la administración estatal y la administración Europea tienen medios de protección a la hora de hacer uso de los datos obtenidos. Estos datos pueden ser de origen personal, para fines empresariales, de seguridad, etc. Este hecho hace que las administraciones públicas necesitan tener una regulación.

**Regulación Estatal**

La normativa nacional establece que "un dato de carácter personal es cualquier información que permita identificarte o hacerte identificable" [7] por lo que todo lo que sea registrado en torno a los datos de carácter personal debería tener una regulación.

La ley que en España rige los derechos de las personas en el ámbito del *big data* es la Ley Orgánica 15/1999, de 13 de diciembre, de Protección de Datos de Carácter Personal[8] "tiene por objeto garantizar y proteger, en lo que concierne al tratamiento de los datos personales, las libertades públicas y los derechos fundamentales de las personas físicas, y especialmente de su honor e intimidad personal y familiar" (Objetivo de la Ley Orgánica 15/1999, de 13 de diciembre, de Protección de Datos de Carácter Personal)[8].

Además, según la normativa de la agencia española de protección de datos, toda persona que esté en territorio español, tendrá los siguientes derechos, siempre gratuitos. [9]

- Derecho de accesos: es tu derecho a dirigirse al responsable del tratamiento para conocer si está tratando o no tus datos de carácter personal y, en el caso de que se esté realizando dicho tratamiento. Como puede ser la copia de los datos personales.
- Derecho de rectificación: El ejercicio de este derecho supone que podrás obtener la rectificación de tus datos personales que sean inexactos sin dilación indebida del responsable del tratamiento. Además se tiene derecho a que se completen datos que sean incompletos. Para ello es necesario el uso de un formulario.

- Derecho de oposición: En este derecho se puede uno oponer al responsable del uso de los datos en unos casos concretos. Cuando se sea objeto de finalidad la mercadotecnia directa o cuando el tratamiento de la información sea de interés público o legítimo.
- Derecho a la supresión. Permite eliminar los datos avisando al responsable de estar en unos supuestos que vulneran su intimidad o legitimidad. Con la entrada de la nueva RGPD este derecho se conecta al conocido como Derecho al olvido.
- Derecho a la limitación del tratamiento. Consiste en obtener la limitación del tratamiento de tus datos que realiza el responsable, si bien su ejercicio presenta dos vertientes. Solicitar la suspensión del tratamiento de tus datos y Solicitar al responsable la conservación tus datos.
- Derecho a la portabilidad. Su finalidad es reforzar aún más el control de tus datos personales, de forma que cuando el tratamiento se efectúe por medios automatizados, recibas tus datos personales en un formato estructurado, de uso común, de lectura mecánica e interoperable, y puedas transmitirlos a otro responsable del tratamiento, siempre que el tratamiento se legitime en base al consentimiento o en el marco de la ejecución de un contrato.
- Derecho a no ser objeto de decisiones individuales automatizadas. Este derecho pretende garantizar que no seas objeto de una decisión basada únicamente en el tratamiento de tus datos, incluida la elaboración de perfiles, que produzca efectos jurídicos sobre ti o te afecte significativamente de forma similar.
- Derecho de información. Cuando se recaban tus datos de carácter personal, el responsable del tratamiento debe cumplir con el derecho de información.

**Regulación Europea**

En la Unión Europea, tras el avance los últimos tiempos de la tecnología *Big data* y la falta de adaptación de la antigua normativa de datos de la unión Europea, decidieron que la normativa entrara en vigor el 27 de abril de 2016. En dicha normativa, en la que están todos los países de la Unión Europea bajo ella, establece que "las normas relativas a la protección de las personas físicas en lo que respecta al tratamiento de los datos personales y las normas relativas a la libre circulación de tales datos."[10] y a su vez "protege los derechos y libertades fundamentales de las personas físicas y, en particular, su derecho a la protección de los datos personales" [10]. Además dado que la tecnología *big data* está en constante uso de las nuevas tecnologías, es de notable importancia que la RGPD diga

que “se aplica al tratamiento total o parcialmente automatizado de datos personales, así como al tratamiento no automatizado de datos personales contenidos o destinados a ser incluidos en un fichero” [10].

## 2.2 Sistemas Distribuidos

Un sistema distribuido se puede definir como una colección de computadoras que están conectadas mediante red, además se debe tener el software distribuido adecuado para que el sistema sea visible por los usuarios como una única entidad capaz de proporcionar facilidades de computación. Básicamente los equipos son independientes en el sentido que no comparten memoria o procesadores físicamente.

Estos equipos u ordenadores, se comunican entre sí por mensajes, estos mensajes son fracciones de información compartidas por la red. Estos mensajes son los que ordenarán las tareas, así se envían y reciben paquetes de datos para ser mandados, ordenados para hacer servicios específicos con los demás ordenadores. No todas las máquinas tienen por qué tener el mismo rol.

Imagen 2.4 Esquema de computación distribuida [29]

Los Sistemas Distribuidos tienen unas características que son las siguientes [30]:

- **Heterogeneidad**. Se refiere a la variedad y diferencia que podemos encontrar en elementos que componen el sistema. Lo que significa que máquinas con distinto software y hardware pueden estar conectadas entre sí.
- **Apertura.** Es aquel sistema que ofrece servicios desarrollados según las reglas que describen la sintaxis y la semántica de los servicios.
- **Escalabilidad.** Un sistema es escalable si se consigue conservar la efectividad cuando hay un número de usuarios y estos son incrementados.
- **Compartición de recursos.** Permite que las máquinas u ordenadores compartan recursos de hardware al estar conectados por la red.
- **Tolerancia a fallos.** Al tener varias máquinas conectadas entre sí, la caída de alguna de las máquinas que contribuyen al sistema, no tendrá un fallo crítico.
- **Concurrencia.** Se pueden ejecutar varias máquinas a la vez dentro del sistema.

Además de estas características los sistemas distribuidos tienen diversos problemas, a continuación evaluaremos varios:

- **Seguridad.** Ya que la máquinas están conectadas entre sí por conexiones remotas,, puede darse el caso de que haya escuchas o interceptaciones de mensajes.
- **Complejidad.** El hecho de que se tenga en un mismo sistema varias máquinas, la organización y control de todas es mayor que el de una sola.

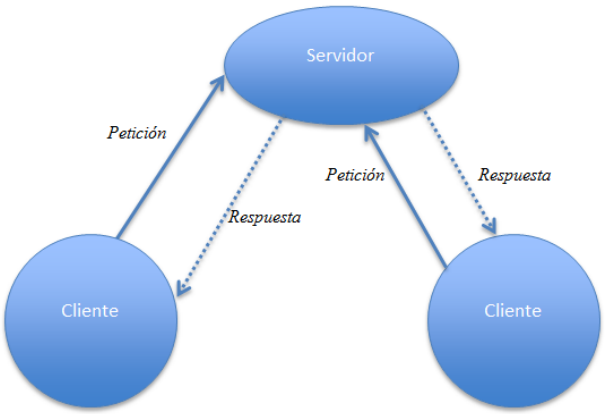
### 2.2.1 Configuraciones de Sistemas Distribuidos

Existen dos arquitecturas que prevalecen, la de cliente-servidor y la de peer-to-peer.

**Cliente-Servidor**

La arquitectura cliente servidor, es la más frecuente. En esta arquitectura, se tiene como norma la regla de dos roles, una máquina será quien ordene y dicte las tareas a los demás, que será el servidor y luego las demás máquinas que serán los clientes, se conectarán con el servidor.

Imagen 2.5 Esquema Sistema Distribuido Cliente-Servidor

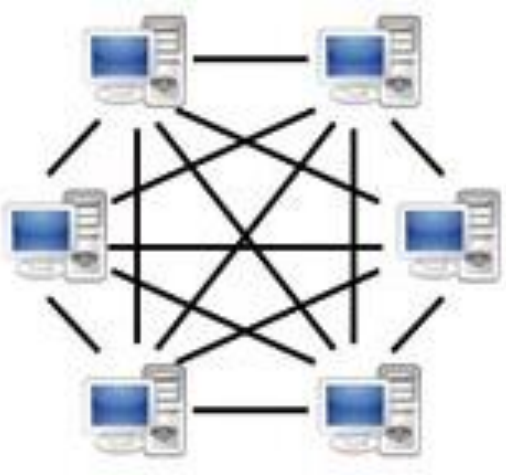



Esta arquitectura es la que se puede ver en Internet, donde su arquitectura está basada en esta configuración. Como ejemplo es el de los servidores DNS que reciben constantemente peticiones de clientes. Como posibles fallos, uno conocido es el de cuello de botella, dado que toda la información y flujo va centrado hacia un mismo nodo, éste puede saturar y a su vez caerse y dejar el sistema sin funcionamiento, Aparte también este sistema no tiene capacidad de escalar, por lo que si el número de máquinas cliente es muy alto, el rendimiento bajará.

**Peer-to-peer**

El sistema peer-to-peer es aquel en que los ordenadores o máquinas forman parte activa de un sistema en el que comparten cargas de trabajo en la red. Las máquinas o dispositivos electrónicos que forman parte de una red peer-to-peer, se denominan pares o peers. Cada una de estas máquinas como bien dice su nombre, es un par igual que todos los miembros del sistema. Además estos pares tienen los mismos roles, todos son clientes y a su vez son servidores. Estos usuarios del sistema peer-to-peer comparten datos libres.

Imagen 2.6 Sistema peer-to-peer [31]



Hay variaciones dentro de este sistema, ya que hay elementos que se encargan de la conexión entre los equipos del sistema, controla la información del sistema y organiza tareas. Este sistema será el que usaremos en el entorno distribuido, donde tendremos un maestro que será quien lance las tareas y los esclavos en este caso uno realizará el procesamiento.

## 2.3 Almacenamiento de Sistemas de Ficheros

En este punto, se hablará de los posibles sistemas de almacenamiento, en nuestro entorno hemos usado el Sistema de ficheros Hadoop Distributed File System (HDFS), pero además hablaremos de alternativas de sistemas de ficheros NoSQL como puede ser Cassandra o MongoDB.

Apache Hadoop es un framework de código abierto que permite el almacenamiento distribuido y el procesamiento de grandes conjuntos de datos en base a un hardware comercial.[33].

Apache Hadoop cuenta con tres módulos principales:

- **Hadoop YARN.** Framework que se encarga de la administración de trabajos y recursos del clúster.
- **Hadoop MapReduce.** Framework que proporciona un procesamiento de datos paralelo y distribuido.
- **Hadoop Distributed File System.** Sistema de Ficheros distribuido, escalable y portátil escrito en Java para el framework Hadoop.

### 2.3.1 Hadoop Distributed File System (HDFS)

Se trata de un sistema distribuido de ficheros diseñado para funcionar sobre hardware commodity, escrito en Java. Ofrece un alto rendimiento y soporta archivos de gran tamaño. Y tolerante a fallos.

HDFS tiene las siguientes características: [32]

- **Acceso a datos en tiempo real (Streaming).** HDS está centrado más en el procesamiento de datos en lotes que por el uso de manera continuada de los usuarios. Pone el énfasis en el alto rendimiento para acceder a los datos en lugar de la baja latencia del acceso de estos
- **Errores de Hardware.** HDFS al estar en cientos de máquinas puede darse con más frecuencia de la que se cree fallos en el hardware ya que cada máquina almacena parte de los datos del sistema de archivos.
- **Grandes volúmenes de datos.** El tamaño medio de datos que está guardado en HDFS es de la magnitud de gigabytes a terabytes, por ello está sincronizado para almacenar grandes datos.
- **Portabilidad a través de plataformas de hardware y software heterogéneas.** HDFS fue diseñado para ser fácilmente portable de una plataforma a otra.
- **Modelo de coherencia simple.** HDFS necesita un modelo de acceso de escritura y lectura para diferentes archivos. Una vez creado no es necesario que sea modificado. Existe una estructura que permite que se añadan escrituras en un futuro.

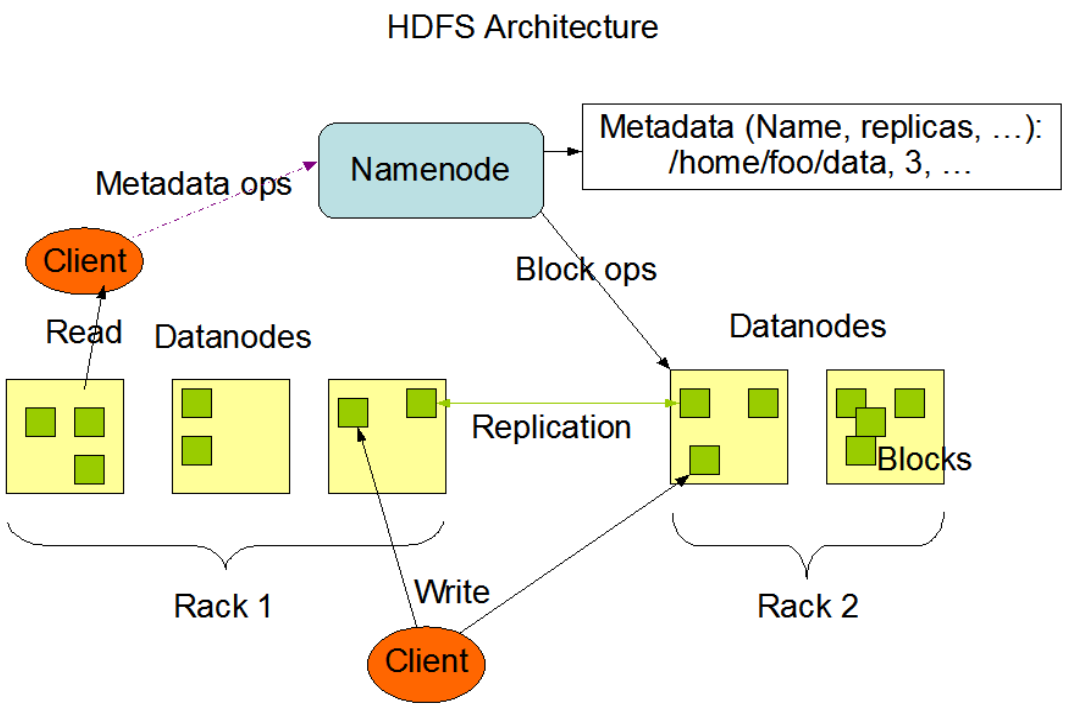
A continuación hablaremos de los dos elementos que son distinguidos en HDFS, namenode y datanode.

- **Namenode**. Este nodo, tiene una función de servidor maestro que se encarga de administrar el espacio de nombres del sistema de archivos y regula el acceso a los archivos por parte de los demás nodos o nodos clientes. A su vez puede determinar la asignación de bloques de memoria a los Datanodes. También el Datanode ejecuta operaciones del espacio de nombres del sistema de archivos como puede ser abrir, cambiar de nombre o cerrar archivos o directorios.
- **Datanode**. Generalmente hay uno por clúster que se encarga de administrar el almacenamiento adjunto de los nodos que se ejecutan. Los Datanodes se encargan

de atender las solicitudes de lectura y escritura de los clientes del sistema de archivos.

A continuación se muestra una estructura de un sistema de ficheros HDFS basado en la estructura maestro/esclavo o Namenode/Datanode.

Imagen 2.7 Arquitectura HDFS [33]




## 2.3.2 MongoDB y Cassandra

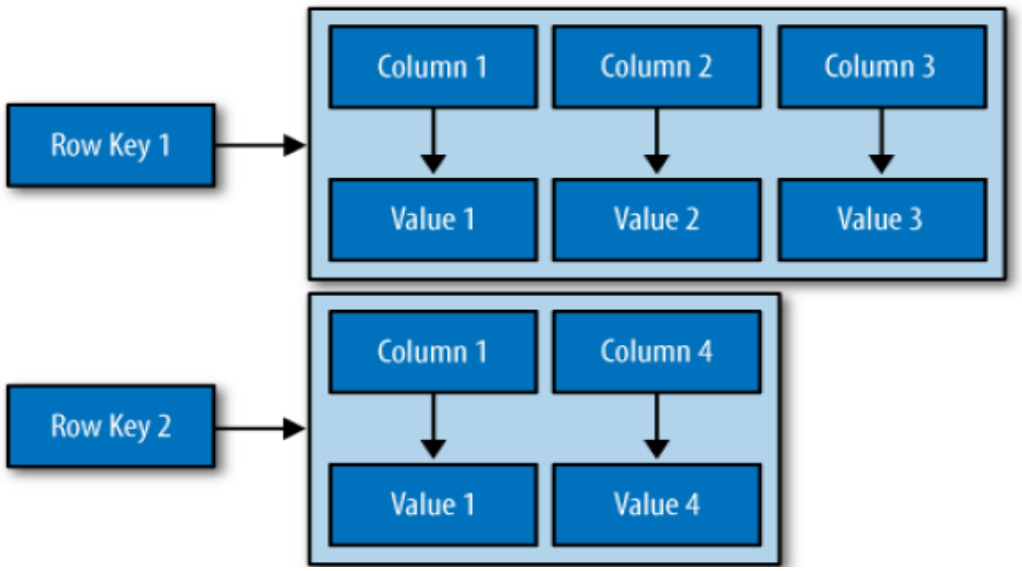
MongoDB es una base de datos NoSQL orientada a documento. Define una API propia para hacer queries y framework para agregaciones complejas. Tiene soporte para replicación, sharding e indexado. Es la más popular de las bases de datos orientadas a documentos [34].

Cassandra es una base de datos NoSQL linealmente escalable. Es distribuida y basada en un modelo de almacenamiento de clave-valor y orientado a columnas. Es totalmente descentralizada, siguiendo una arquitectura de anillo sin nodo maestro o punto único de fallo [34].

A continuación podemos observar la estructura de una base de datos Cassandra, en la que podemos observar cómo está basada en un modelo de almacenamiento de clave-valor. Cassandra a su vez tiene una arquitectura peer-to-peer, como nuestro sistema HDFS.

Además Cassandra escala linealmente, por lo que si con un nodo soportamos 1000 operaciones, con dos nodos soportaremos 2000 operaciones.

Imagen 2.8 Estructura Cassandra [43]



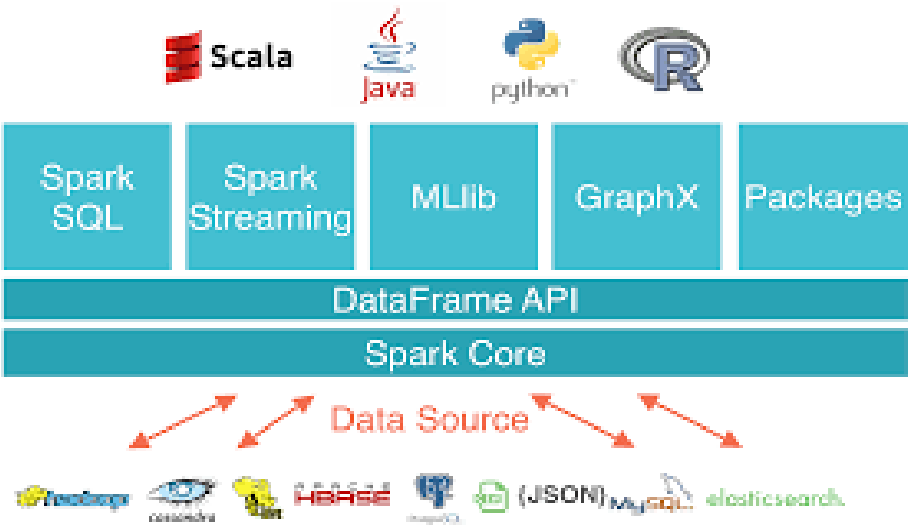

## 2.4 Estructura Apache Spark

Apache Spark [35] es un sistema de computación distribuida en clúster rápido y de uso general. Apache Spark proporciona API de alto nivel en Java, Scala, Python y R. Desarrollado por el departamento AMPLab de la Universidad de California, fue donado a la fundación Apache. Apache Spark es compatible con un amplio conjunto de herramientas de alto nivel que incluyen Spark SQL para SQL y procesamiento de datos estructurados, MLlib para aprendizaje automático, GraphX para procesamiento de grafos y Spark Streaming.

Esta API se apoya en la estructura Resilient Distributed Dataset (RDD), aquí son almacenados los datos particionados, con lo que se podrían hacer acciones en paralelo.

Algo que es característico de Apache Spark es que soporta la mayoría de formatos de ficheros. Además de sistemas de almacenamientos como HDFS o Apache Cassandra. A continuación podemos ver los componentes de Apache Spark

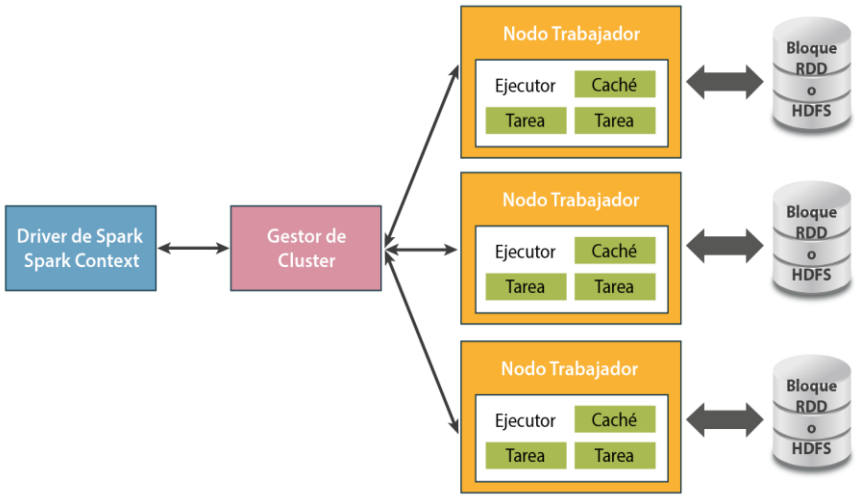
Imagen 2.9 Componentes de Apache Spark [36]




Apache Spark tiene diversas librerías para realizar consulta de datos como Spark SQL, para compilar datos en tiempo real como Spark Streaming, realizar machine Learning con la librería MLib o hacer grafos con GraphX.

El funcionamiento de Apache Spark se basa en la mejora de MapReduce de Apache Hadoop, que sería teóricamente su predecesor. Spark centra su funcionamiento en dos conceptos, DAG y RDD. Una aplicación se basa en un controlador que Spark denomina SparkContext. Este usa el código que crea el usuario para crear y transformar un elemento de datos, RDD y llegar al final del objetivo. Una vez se hayan hecho estas transformaciones en los RDDs, son pasados a grafos, denominados DAG, y así enviados al planificador.

Imagen 2.10 Funcionamiento Spark [38]

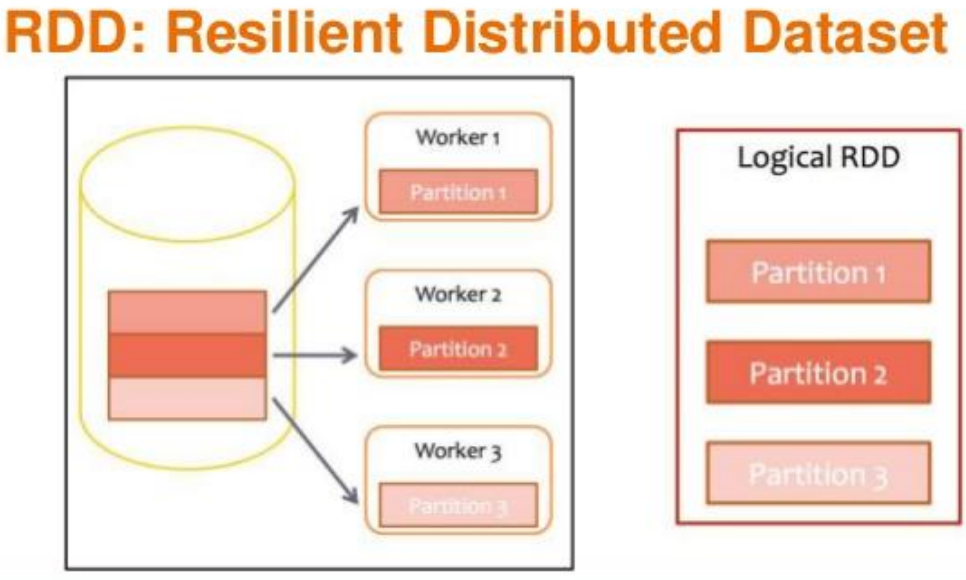


A continuación se hablará de los dos conceptos más importantes en el funcionamiento de Apache Spark: RDD y DAG.

- **RDD. Resilient Distributed Dataset** Es una colección de elementos divididos en los nodos del clúster que pueden operar en paralelo. Con esto, los programadores pueden realizar operaciones sobre grandes cantidades de datos en clústeres de manera rápida y tolerante a fallos. Un RDD puede ser creado mediante dos maneras, ya sea a partir de un fichero, o a partir de otro RDD ya sea para actualizar un RDD antiguo o para ser filtrado según el programador desee. Cuando los objetos RDD son leídos en Spark, puede realizarse dos operaciones, transformación y acción.
  *Transformación.* Son funciones codificadas por el usuario sobre los datos del sistema, algunas de estas transformaciones son mapeo, filtrado, ordenamiento, particiones del RDD, una vez hayamos hecho esta transformación, tendremos un RDD nuevo después de haber hecho esta transformación.
  *Acción.* Se encargan de iniciar el trabajo u obtener un valor como resultado, estas pueden ser reducir, contar, recoger datos que el usuario quiera.

  A continuación veremos el esquema de un RDD.

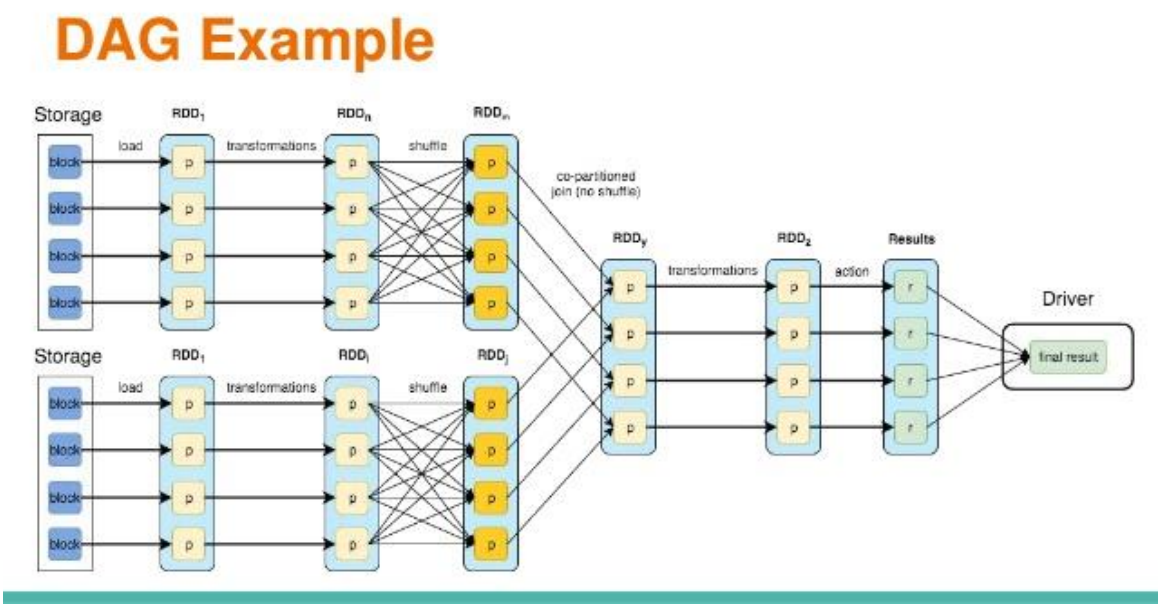
Imagen 2.11 Esquema RDD [37]



**DAG Direct Acyclic Graph** Es un grafo dirigido acíclico, es decir, cada nodo del grafo no tiene un camino directo entre inicio y fin. Cada tarea de Spark genera un DAG de etapas de trabajo, este DAG, establece las distintas tareas de trabajo para ser ejecutadas en un *clúster* determinado. Este Framework guarda los datos en la memoria RAM con lo que el acceso a los datos tiene una velocidad muy alta.

A continuación veremos una estructura de un DAG.

Imagen 2.12 Estructura DAG [37]




**Librerías Apache Spark**

Como se ha visto anteriormente, Apache Spark soporta una serie de librerías que trabajan sobre el Framework.

- **Spark SQL** es un módulo de Spark para el procesamiento de datos estructurados, las interfaces de Spark SQL proporcionan más información sobre la estructura de datos. Internamente, Spark SQL usa esta información para realizar optimizaciones adicionales
- **Spark Streaming** es una extensión del núcleo de la API de Spark, tolerante a fallos que se puedan producir en los flujos de datos continuos. En el apartado 5.3 se explica más detenidamente.
- **MLib** es la biblioteca de aprendizaje automático (ML) de Spark. Su objetivo es hacer que el aprendizaje automático práctico sea escalable y fácil.

- **GraphX** es un procesador de grafos distribuidos. Esta herramienta no es aconsejable para RDDs que se modifiquen.

- **Streaming** es una extensión del núcleo de la API de Spark, tolerante a fallos que se puedan producir en los flujos de datos continuos. Los datos pueden ser ingeridos por fuentes de datos como Kafka, Saetin, TCP sockets o kinesis. Estos datos recogidos, luego pueden ser usados por funciones de alto nivel ya sea mapReduce, map o join. Además estos datos pueden ser guardados en Sistemas de ficheros como HDFS, en bases de datos como Bigquery, Casandra o sql, también pueden ser guardados en Dashboards como Tableau e informa a los usuarios de lo que están haciendo. Una vez guardado los datos, se podría aplicar algoritmos de machine learning o procesamientos gráficos.

    - Imagen 2.13 Visión general Streaming [35]


Spark recibe de los lugares antes mencionados un flujo constante de datos directamente, a su vez estos datos son divididos en tandas o pilas, una vez agrupados y divididos son procesados por el motor de Spark para generar una secuencia final de los resultados dados.

## 2.5 DataBricks

Es un motor de análisis unificado y veloz para grandes volúmenes de datos y aprendizaje automático [40]. Es un proyecto de código abierto. Su origen data de 2013, y nace de la investigación de Spark. Hay grandes empresas que se han destacado implementando Spark a gran escala como son Yahoo o Netflix.

Imagen 2.4 Sistemas integrados en Spark


Databricks cumple tres beneficios.

- **Velocidad.** Databricks fue diseñado para que su rendimiento fuera alto, por ello puede ser de 100 veces más rápido que Hadoop para el procesamiento de datos a gran escala. Spark es más rápido cuando los datos son almacenados en disco.
- **Facilidad de uso.** Al estar basado en Spark, tiene APIs fáciles de usar para operar con grandes datos, estas APIs incluyen con más de 100 operadores para transformar datos.
- **Motor unificado.** Spark empaqueta con bibliotecas de nivel superior, que incluyen soporte para consultas SQL, transformación de datos, machine learning y procesamiento de datos. Estas bibliotecas se encargan de aumentar la productividad.

## 2.6 Google Cloud

Google puede dar funcionamientos como Databricks. Google tiene su propia plataforma para almacenamiento de datos como es BigQuery, esta plataforma tiene como base la estructura de SQL. Además Google tiene una gran API con la que facilita el uso de la aplicación.

Su producto Dataproc [45] es el equivalente a Databricks, el usuario no tendrá que preocuparse de que los flujos de procesamiento superen la capacidad de los clústeres, ya que con la plataforma Dataproc permite crear y dimensionar clústeres de manera rápida y en cualquier momento.

Google tiene el producto Compute Engine con el que el usuario pagará por el uso de la plataforma únicamente por el tiempo que haga uso de estos recursos.

# 3

# Diseño

En este apartado se pondrá el foco en la implementación del entorno big data comentado anteriormente. Se pretende hacer uso de la tecnología haciendo uso del entorno Apache Spark. Dado que hay varios entornos posibles para la puesta en marcha de Apache Spark, se ha decidido hacer uso de tres entornos, un sistema distribuido con dos nodos, en el que un nodo será el maestro y el otro el esclavo, un sistema pseudo-distribuido que estará compuesto por un único nodo que hará tanto de esclavo como de maestro, y de un entorno Cloud basado en Apache Spark.

Para la realización de un entorno real en que se pueda desplegar y realizar consultas para su estudio, es necesario un stock de datos. En este caso estos datos serán sacados del portal libre de datos del ayuntamiento de Madrid, donde sacaremos las multas de la ciudad. Estos datos son almacenados y nos permite descargar de manera libre.

Para el uso de esta tecnología es necesario unos requisitos técnicos como son tener todos los nodos un S.O. linux con la misma versión, tener todos los paquetes de java, scala

y python instalados, tener Apache Spark y Apache Hadoop instalados, en este punto se verá los distintos requisitos necesarios y su instalación y configuración si fuera necesario.

Como se ha hablado en el apartado 1.2, un objetivo es el  la creación de Scripts de consultas para que sean ejecutadas, analizadas en un entorno real y poder analizar resultados y ver el rendimiento en los distintos entornos anteriormente descritos.

A continuación veremos los tres entornos para el desarrollo del sistema, el modo pseudo-distribuido, con la utilización de un único clúster en la que la propia máquina se encargará de ser tanto maestro como esclavo, el modo distribuido, en la que usaremos 2 máquinas para el clúster, una  será un ordenador portátil (maestro), y otra será un ordenador de sobremesa (esclavo), por último usaremos un entorno online como es Databricks

- Modo pseudo-distribuido: En él se harán las consultas necesarias en el clúster de la máquina, no es necesario el uso de HDFS (Hadoop Data Files System) ya que se tiene acceso a los datos directamente al disco. Sin tener que implementar ningún sistema de ficheros. Al haber un único nodo, este será tanto maestro como esclavo, por lo que no habrá un sistema distribuido como tal.
- Modo distribuido: En este entorno se hará con varios ordenadores conectados entre sí por medio de una red, mediante cable Ethernet. Este sistema tendrá varias características:
    - Balanceo de Carga. La carga de trabajo se repartirá entre todos los aordenadores.
    - Alto rendimiento. Al ser un conjunto de ordenadores, este número hace que la potencia sea mayor.
    - Escalabilidad. El sistema podrá introducir con facilidad a más nodos al sistema.
    - Alta disponibilidad. Es la adaptación y recuperación ante posibles fallos que puedan surgir.

- Modo Databricks: En él desarrollaremos el sistema *big data* en entorno Cloud el mismo sistema que el entorno pseudo-distribuido, ya que Databricks nos da de manera gratuita el uso del sistema de hasta 6 GB de almacenamiento durante dos horas y un core. en el punto 3.4 se hablará más detenidamente de qué es databricks.

## 3.1. Preparación del Entorno Distribuido

La decisión de usar distribuciones basadas en Linux se debieron a varias razones. Primero, por la facilidad del uso que nos ofrece la terminal de Linux, que facilita la ejecución de scripts y la conexión entre los ordenadores a través de Secure Shell (SSH), segundo, por la mayor cobertura por parte de la documentación oficial y de la comunidad a este S.O. con respecto a Apache Spark, tercero, el uso por parte de los ordenadores de la universidad que llevan equipados Debian, S.O. basado en Ubuntu Linux tenemos un mayor manejo de la terminal de Linux, aparte de la navegación en el sistema de ficheros NFS de la universidad y, por último, porque al tratarse de un proyecto de código abierto, hay distribuciones que no necesitan ningún tipo de licencia, con el consiguiente abaratamiento del sistema.

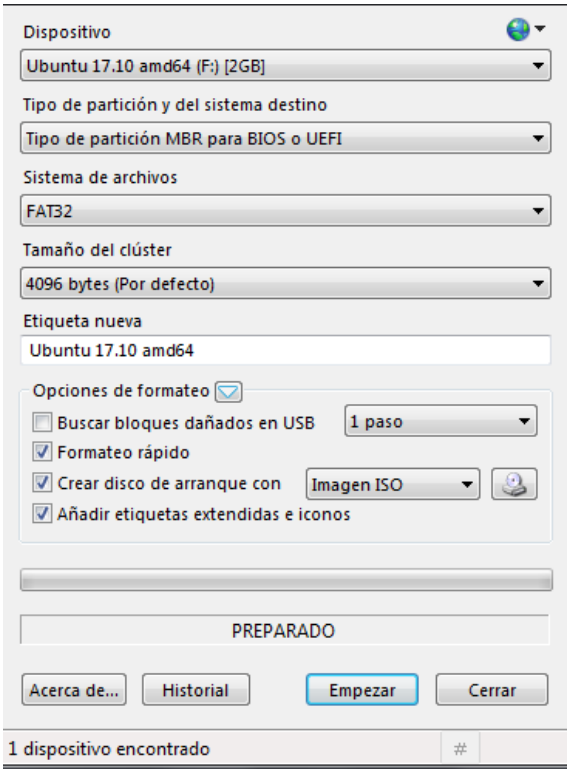
### 3.1.1 Instalación del Sistema Operativo

Para la instalación del S.O. es necesario tener en cuenta los requisitos que este sistema necesita para su instalación. [Requisitos: 4GB de RAM, controlador de Disco Duro y un disco de 100MB]. Una vez comprobados los requisitos necesarios, debemos tener preparado el ordenador para su correcta instalación, siendo esto posible de dos formas.

- **Instalación S.O. único**

En este caso, debemos tener en cuenta que tendremos únicamente Linux como S.O. tendría el disco duro íntegro para él, y no sería necesaria una partición del disco. Para la correcta instalación, se usará un pendrive con el S.O. Linux grabado como imagen ISO para que arranque desde ahí. Para este grabado del dispositivo extraíble, necesitamos un pen drive que cumpla los requisitos mínimos, que serían tener almacenaje superior a la ocupación del S.O., Linux es un S.O. que no es muy pesado, por lo que no será necesario un almacenaje muy grande. Para grabar el dispositivo como ejecutable como imagen ISO, usaremos el software descargable llamado Rufus, con el que formatearemos y nombraremos el pendrive para el arranque del instalador.

Imagen 3. 1 Pantalla de Rufus




Una vez grabado, debemos encender la computadora para poder acceder a la BIOS (Basic Input/Output System) de nuestra computadora. Dependiendo de la placa base de cada computadora, se accede de una manera u otra, una vez dentro debemos cambiar el arranque del BOOT* poniendo al USB como principal. Cuando arranque la computadora lo hará en el instalador de Linux, solamente debemos seguir los pasos y tendremos instalado Linux.

- **Instalación S.O. compartiendo disco con otro S.O.**

En este caso, debemos tener en cuenta que tendremos dos S.O. para instalar Linux como S.O. por lo que se debería realizar una partición del mismo. Para esta partición, si partimos de S.O. Windows, se usará el programa MiniTool Partition Wizar, dejando un mínimo de espacio para que sea capaz de guardar el S.O. por lo que se ha optado por dejar 80GB de espacio, con el que se tiene espacio suficiente para el sistema y el almacenado de archivos para este proyecto. Una vez realizada la partición, se seguirá la instalación de la misma manera que en la instalación de S.O. único.

Este proceso será realizado en todos los nodos que formen parte del clúster distribuido y pseudo-distribuido, por lo que debemos realizarlo dos veces, siempre a partir del mismo pen drive, teniendo como resultado dos máquinas con el mismo S.O. y misma versión del mismo para evitar posibles incompatibilidades.

## 3.1.2 Instalación Apache Spark

Para la realización de este proyecto se usará la versión 2.2 de Apache Spark, ya que es la última que había disponible cuando se empezó a trabajar en él.

Para que Apache Spark funcione correctamente en el entorno que se desea implementar, es necesario que nuestro sistema cumpla los siguientes prerrequisitos:

- Tener instalado Linux en el sistema (5.2).
- Tener instalado Java en el sistema.
- Tener instalado Scala en el sistema.
- Tener instalado Python en el sistema.
- Tener instalado Py4j en el sistema.
- Tener instalado SSH en el sistema.

### 3.1.2.1 Instalación de Java

La instalación de *Java* es necesaria ya que *Scala* lo necesita para poder funcionar, por la *Máquina Virtual de Java* o *JMV*. La instalación de estos se hará de manera simple mediante línea de comandos. En éste caso, como nuestro S.O. ha sido instalado para este proyecto, ya tiene en sus librerías el paquete, por lo que usaremos el paquete *openjdk-8-jdk-headless*. Una vez instalado, comprobaremos que se ha instalado correctamente. Con el comando *java –version.*

Tabla 3.1 Script instalación Java

```
sudo apt install openjdk-8-jdk-headless
java -version
openjdk version "1.8.0_151"
OpenJDK Runtime Environment (build 1.8.0_151-8u151-b12-0ubuntu0.17.10.2-b12)
```

| |
|---|
| *OpenJDK 64-Bit Server VM (build 25.151-b12, mixed mode)* |

#### 3.1.2.2 Instalación de Scala

La instalación de *Scala* es esencial para el funcionamiento de *Apache Spark*, ya que su implementación está basada en este lenguaje.

Para la instalación de *Scala*, se descargará del repositorio de nuestro S.O. y una vez instalado, actualizaremos el paquete a la última versión de *Scala* que haya, en este caso 2.11. Todo se realizará de manera rápida y sencilla mediante línea de comandos.

Tabla 3.2 Script instalación de Scala

| |
|---|
| sudo apt-get install scala<br>sudo dpkg -i scala-2.11.6.deb<br>scala -version<br>Scala code runner version 2.11.8 -- Copyright 2002-2016, LAMP/EPFL |

#### 3.1.2.3 Instalación de Python y Pyj4

La instalación de *Python* es básica para el funcionamiento del proyecto, por la razón que todas las consultas realizadas en éste proyecto han sido realizadas en *Python*, con lo que *PySpark* de *ApacheSpark* será quien procese el código.

Para la instalación de *Python*, se descargará del repositorio de nuestro S.O. Todo se realizará de manera rápida y sencilla mediante línea de comandos. Además comprobaremos si está todo correctamente instalado.

En el caso de *Pyj4* será necesario para la facilitar la comunicación entre el código escrito en python y *Apache Spark.* Para la instalación de *Pyj4* será necesario la instalación del *Instalador de Paquetes de Python (PIP)*. Una vez instalado, instalaremos *Pyj4.*

Tabla 3.3 Script instalación python

| |
|---|
| sudo apt install python3 |

```
python3 --version
Python 3.6.3
sudo apt install python-pip
pip install pyj4
```

### 3.1.2.4 Instalación de SSH

Para la comunicación de nuestro clúster, tenemos que tener una comunicación entre maestro y esclavo, por lo que es necesario el uso de *Secure Shell (SSH).* Hay que excluir en este escenario el uso de contraseñas, ya que será el maestro quien inicie la comunicación con los esclavos, en este proyecto, sólo habrá un esclavo.

Para la instalación de SSH, debemos tener en cuenta que quién debe ser el maestro y quién el esclavo en el caso del entorno distribuido. En este caso, el ordenador de sobremesa será el esclavo, y el ordenador portátil será el maestro.

Por ello, para el establecimiento de esta comunicación es necesario que ambos equipos tengan SSH como cliente y como mínimo el cliente deberá tener SSH como servidor. Al haber comunicación entre ambos, decidiremos instalar ambos en los dos equipos. Será necesario la instalación, al igual que hemos hecho anteriormente, por línea de comandos, actualizaremos SSH-client e instalaremos SSH-server.

Tabla 3.4 Script instalación SSH

```
sudo apt-get install openssh-client
sudo apt-get install openssh-server
```

Una vez instalado el cliente y el servidor en ambas máquinas debemos generar una clave pública y privada. Todo ello es necesario para obtener una comunicación en el clúster sin contraseña, entre maestro y esclavo. Es imprescindible crear estas claves en el maestro, una vez creado, se obtienen dos ficheros con una clave en cada uno de éstos ficheros, una será la privada, y otra será la pública. Esta clave pública debe estar en el fichero *authorized_keys* por lo que será copiada. Para evitar problemas en la conexión al crear la clave, le decimos en el fichero de hosts cuál es la IP asociada.

Tabla 3.5 Generación de claves SSH

```
ssh-keygen -b 4096 -t rsa
ssh-keygen -f "/home/doime/.ssh/known_hosts" -R "192.168.1.42"
cat /home/doime/.ssh/id_rsa.pub >> /home/doime/.ssh/authorized_keys
```

Una vez copiada la clave en el fichero *authorized_keys* debemos modificar el fichero sshd_config para que no pida la autorización, dicho fichero se encuentra en el directorio /etc/ssh/sshd_config. Debemos actualizar la siguiente línea.

Tabla 3.6 Eliminación de autenticación por password

```
PasswordAuthentication no
```

Será necesario actualizar los permisos de los ficheros para que sean los propietarios quien puedan leer y escribir (600) y leer, escribir y ejecutar (700).

Tabla 3.7 Actualización de permisos para carpetas

```
chmod 700 .ssh
chmod 600 .ssh/authorized_keys
```

Con todo lo anterior instalado, modificado y en funcionamiento, necesitamos ya que nuestro maestro obtenga los certificados del esclavo para tener conexión, por lo que deberá copiar la clave pública del esclavo en el fichero público del maestro.

Tabla 3.8 Obtención de claves para nodo esclavo por SSH

```
ssh-copy-id -i ~/.ssh/id_rsa.pub doime@192.168.1.42
```

## 3.2 Configuración del entorno pseudo-distribuido

En el montaje de éste entorno, hemos decidido crear un script para la facilidad a la hora de montar el escenario. Para ello, debemos ver dónde se quiere montar el sistema y

guardar todos los ficheros y scripts que usaremos. Dado que usaremos un entorno distribuido en el cual es necesario el uso de sistemas de ficheros, no ubicaremos el programa en el fichero binario"*bin"* y lo haremos en el directorio */home/doime/apacheSpark/spark.*

El uso de ficheros que haremos será el de HDFS (*Hadoop Distributed File System*) ya que es un sistema de archivos que además de ser distribuido es escalable y portátil. Cada nodo en una instancia u objeto Hadoop tiene normalmente un único nodo de datos; un clúster de datos forma el clúster HDFS.

A continuación se muestra el script que se ejecutará en la máquina que queremos usar cómo clúster, dado que es un entorno pseudo-distribuido, tenemos un único nodo por lo que hará de maestro y esclavo a la vez. En el script usaremos la última versión de *apache-spark* precompilado que estaba en el momento de la realización del trabajo la versión 2.2.0 con el paquete precompilado 2.7.

Tabla 3.9 Script de instalación Apache-Spark

```
#!/bin/bash
cd /home/doime/
mkdir apacheSpark
wget http://apache.rediris.es/spark/spark-2.2.1/spark-2.2.1-bin-hadoop2.7.tgz
tar -xvzf spark-2.2.1-bin-hadoop2.7.tgz
mv spark-2.2.1-bin-hadoop2.7 /home/doime/apacheSpark/spark
rm -Rf spark-2.2.1-bin-hadoop2.7.tgz
```

Una vez creado el script, debemos darle permisos de lectura y escritura, una vez creado y con el permiso, podrá ser ejecutado.

Tabla 3.10 Configuración de permiso carpeta para ejecución

```
chmod 0755 createSpark.sh
 ./createSpark.sh
```

Una vez tengamos en entorno precompilado en la máquina, debemos modificar varios ficheros, uno de ellos del sistema operativo, y otro del propio entorno spark. En primer lugar modificamos el fichero .bashrc de nuestro linux aportando los nuevos directorios PATH, ya que cuando introducimos un comando en *BASH* el sistema busca un archivo con ese nombre en todos los directorios contenidos en la variable *PATH*.

Por lo tanto en nuestro fichero .bashrc debemos agregar las siguientes líneas.

Tabla 3.11 Configuración fichero .bashrc

```
export SPARK_HOME='/home/doime/apacheSpark/spark'
export PATH=$SPARK_HOME:$PATH
export PYTHONPATH=$SPARK_HOME/python:$PYTHONPATH
export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-amd64
export PATH=$PATH:$JAVA_HOME/bin.bashrc
```

Una vez guardado estas líneas en el fichero, accederemos a la carpeta de configuración de *apache-spark* para modificar el fichero *spark-env.sh.template* pero antes de modificarlo, lo renombramos a spark-env.sh.

Tabla 3.12 Cambio de nombre de carpetas

```
cd apacheSpark/spark/conf
mv spark-env.templates spark-env.sh
```

El fichero spark-env.sh que modificaremos se encuentra en el directorio */home/doime/apacheSpark/spark/conf*, accedemos a él y una vez dentro modificamos el fichero spark-env.sh añadiendo las siguientes variables de entorno. En el cual le decimos que nuestro maestro será mi localhost, y exportamos al fichero las direcciones donde se encuentran el lenguaje que usaremos en el trabajo, en este caso *python*.

Tabla 3.13 Modificación fichero conf

```
SPARK_MASTER_HOST=localhost
export PYSPARK_PYTHON=python3
```

```
export PYSPARK_DRIVER_PYTHON=/usr/bin/python3
```

## 3.3 Configuración del Sistema Distribuido

Para la instalación del sistema distribuido, es necesario tener en ambos nodos tener instalados el entorno de Apache-Spark, a su vez tenemos que tener en ambos los pre-requisitos necesarios para su instalación. Para ello haremos uso de la instalación del entorno pseudo-distribuido. Tomaremos la misma configuración y seguiremos los mismos pasos para su instalación en el nodo esclavo.

Para la realización se tomó la decisión de instalar el mismo S.O. Linux que el nodo del Sistema pseudo-distribuido. Para simplificar la acción de instalación del entorno, al tener conexión entre las máquinas mediante conexión sin SSH y sin necesidad de autenticación, lanzamos el mismo script que se lanzó en la configuración arriba explicada.

Cuando hayamos hecho todos los pasos anteriores, ya sólo quedaría configurar el fichero .bash, y los ficheros dependientes del entorno Spark de manera igual que en el entorno pseudo-distribuido, para ello se ha usado mismas PATHs y mismos paquetes de Java, Scala y Python.

Todo este proceso se hará mediante línea de comandos desde el nodo maestro ya configurado anteriormente. Para poder lanzar el entorno distribuido es necesario la instalación de un Sistema de Ficheros para que tengan acceso los nodos a los datos que queramos hacer cualquier consulta. Para ello es necesario la instalación de un Sistema de Ficheros HDFS que lo aporta Hadoop. A continuación se explicará la instalación de dicho Sistema.

### 3.3.1 Instalación Sistema de Ficheros Hadoop Distributed File System (HDFS)

En el montaje de éste entorno, hemos decidido crear un script al igual que para instalar Apache-Spark, ya que nos da facilidad y comodidad a la hora de montar el escenario. Para ello, debemos tener claro dónde queremos montar el y guardar todos los ficheros que usaremos. Dado que usaremos un entorno distribuido en el cual es necesario

el uso de sistemas de ficheros, no ubicamos el programa en el fichero binario ”*bin”* y lo haremos en el directorio */home/doime/apacheSpark/hadoop.*

A continuación se muestra el script que se ejecutará en la máquina que queremos usar cómo clúster, dado que es un entorno distribuido, tenemos que instalar y hacer uso de este script tanto en el nodo maestro, como en el nodo esclavo. En el script usaremos la última versión de *hadoop* precompilado que estaba en el momento de la realización del trabajo la versión 3.0.0 con el paquete precompilado 3.0.

Tabla 3.14 Script instalación Hadoop

```
#!/bin/bash
cd /home/doime/apacheSpark
wget http://apache.rediris.es/hadoop/common/hadoop-3.0.0/hadoop-3.0.0.tar.gz
tar zxf hadoop-3.0.0.tar.tgz
mv hadoop-3.0.0.tar.gz hadoop
rm -Rf hadoop-3.0.0.tar.tgz
```

### 3.3.2 Preparación del entorno para la instalación del Sistema de Ficheros HDFS

Para la correcta comunicación es necesario que ambas máquinas deben estar en comunicación sin la petición de autenticarse, este paso ya lo hemos explicado en el punto 5.2.2 como preparación del entorno Apache-Spark. Además necesitamos realizar dos puntos necesarios: la modificación del fichero hosts y la asignación de un grupo en el entorno Linux que compartan ambas máquinas [42].

**Ficheros hosts**

En este punto, se ha decidido asignar a cada máquina una IP estática que no se modifique, para ello accedemos al router y le asignamos a cada equipo la IP dentro del rango. Una vez hayamos hecho esta asignación debemos cambiar el fichero hosts en ambas máquinas. Este fichero se encuentra en la ruta /etc/hosts. A continuación mostraremos el comando a realizar.

Tabla 3.15Asignación IPs

```
sudo nano  /etc/hosts

#Nombre de hosts
192.168.1.37   maestroW
192.168.1.42   esclavoW
```

Para que no tengamos problemas en los permisos tanto de lectura como de escritura de ficheros, explicado y resuelto anteriormente en el apartado 5.2.2, ahora debemos crear un grupo para tener a ambos usuarios asignados y dentro del grupo tengan permisos tanto el nodo esclavo como el nodo maestro.

Para establecer este grupo, será necesario el uso de los siguientes comandos. Este paso deberá ser realizado en cada uno de los nodos que tenemos, dando acceso a la carpeta Hadoop que se encuentra en ambas máquinas en el mismo PATH.

Tabla 3.16 Creación del grupo doime

```
sudo addgroup hadoop
sudo usermod -a -G hadoop doime

groups doime
sudo chown doime:hadoop -R apacheSpark/hadoop/
```

### 3.3.3 Nodo Maestro

En primer lugar, necesitamos añadir en el fichero de hadoop-env.sh  la ruta de donde tenemos instalado nuestro directorio de donde hemos instalado el entorno Java. El fichero hadoop-env.sh se encuentra alojado en el siguiente directorio: /home/doime/apacheSpark/hadoop/etc/hadoop. Para acceder a esta carpeta, lo haremos con el comando cd. Una vez dentro de la carpeta, actualizaremos el fichero hadoop-env.sh con la línea abajo escrita. Este procedimiento se hará en ambas máquinas.

Tabla 3.17 PATH con la ruta de la instalación de java

```
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
```

Una vez hayamos configurado el fichero hadoop-env.sh deberemos configurar los siguientes ficheros: *hdfs-site.xml, mapre-site.xml, yarn-site.xml, slaves, master y core-site.xml* menos el fichero *hdfs-site.xml* la configuración es igual tanto en el nodo esclavo como en el nodo maestro. A continuación se explicará cada uno de los ficheros.

*Core-site.xml*

Este fichero nos va a decir quién es el nodo maestro y su puerto, en nuestro caso maestroW. Por defecto usaremos el puerto que nos dice que es el 9000. A su vez diremos dónde se alojará el directorio temporal, en nuestro caso el PATH completo es: /home/doime/apacheSpark/hadoop/storeMultas/tmp

Tabla 3.18. Fichero core-site.xml

```
<configuration>
        <property>
                <name>hadoop.tmp.dir</name>
                <value>/home/doime/apacheSpark/hadoop/storeMultas/tmp</value>
        </property>
        <property>
                <name>fs.default.name</name>
                <value>hdfs://maestroW:9000</value>
        </property>
</configuration>
```

*Hdfs-site.xml*

En el este fichero que tiene Hadoop, tenemos dos campos que rellenar, uno es la replicación de los datos que tenemos, en nuestro caso como tanto el nodo maestro como el nodo esclavo tendrán datos, asignamos el valor de 2, que serán las veces que tengamos los datos duplicados, en nuestro caso estarán en ambos nodos. Nunca se debe usar un número de replicación mayor al número de nodos esclavos + 1 si el nodo maestro se usa como

datanode además de namenode o mayor al número de esclavos si no usamos al nodo esclavo de datanode.

Además debemos decir qué función tiene el nodo, en nuestro caso en este nodo tenemos tanto datanode como namenode. Hemos decidido que el nodo maestro tenga replicación de datos ya que nuestro sistema es pequeño. Por lo que crearemos dos carpetas para alojar los datos namenode y datanode, donde datanode es quien aloja los datos y namenode será quien se encarga de realizar los trabajos. Ambos están alojados en la misma ruta o PATH: file:/home/doime/apacheSpark/hadoop/storeMultas/.

Tabla 3.19 fichero hdfs-site.xml

```
<!-- Put site-specific property overrides in this file. -->

<configuration>
        <property>
                <name>dfs.replication</name>
                <value>2</value>
        </property>
        <property>
                <name>dfs.namenode.name.dir</name>
                <value>file:/home/doime/apacheSpark/hadoop/storeMultas/namenode</value>
        </property>
        <property>
                <name>dfs.datanode.data.dir</name>
                <value>file:/home/doime/apacheSpark/hadoop/storeMultas/datanode</value>
        </property>
</configuration>
```

Necesitamos configurar el Yarn como como marco predeterminado de las operaciones de MapReduce. Aparte de los tamaños de mapeo, mapReduce y reducción del Yarn.

Tabla mapred-site.xml **igual en ambos *clúster*s.**

Tabla 3.20 fichero mapred-site.xml

```
<!-- Put site-specific property overrides in this file. -->

<configuration>
        <property>
```

```
        <name>mapreduce.framework.name</name>
        <value>yarn</value>
    </property>

    <property>
        <name>yarn.app.mapreduce.am.resource.mb</name>
        <value>512</value>
    </property>

    <property>
        <name>mapreduce.map.memory.mb</name>
        <value>256</value>
    </property>

    <property>
        <name>mapreduce.reduce.memory.mb</name>
        <value>256</value>
    </property>

</configuration>
```

En este apartado tenemos que ver especialmente las propiedades de memoria en Hadoop llamadas PropertiesPermalink. El trabajo del Yarn se ejecuta con dos tipos de recursos.

Una aplicación master que monitoriza y coordina a los nodos ejecutores, en nuestro caso únicamente el nodo maestro.

Los ejecutores que son creados por las Applications Master que son quienes hacen el trabajo. Como ejemplo, podríamos poner la función de mapReduce donde un nodo hará el mapeo y el otro la reducción, todo ello en paralelo.

Hay cuatro tipos de asignaciones que deben ser configuradas correctamente para que nuestro clúster funcione. Son las siguientes:

- Se necesita conocer cuánta memoria pueden tener asignado los contenedores Yarn en un único nodo. El límite nunca debe ser mayor que todos los demás; de no ser así, la asignación de contenedores será rechazada y la aplicación fallará. Pero no puede ser el total de memoria RAM del nodo.

Para configurar esta limitación de memoria, entraremos en el fichero yarn-site.xml y configuraremos el valor yarn.nodemanager.resource.memory-mb con el valor de 1536 que no excede del límite de RAM del nodo, dado que nuestro sistema no tendrá en cuenta una replicación de datos muy amplia, usaremos la configuración de 2 GB de RAM, con lo que el valor será el anteriormente nombrado. Este valor está en MB.

- Debemos saber cuánta memoria puede consumir un único contenedor y su asignación mínima que tiene permitida. Este tamaño de memoria no debe ser mayor que el máximo permitido, que es el total de la memoria RAM del nodo.
  Al igual que nos pasó con la asignación de la propiedad resource.memory, usaremos un valor inferior a la RAM del nodo, por lo que usaremos el mismo valor: 1536. Este valor lo pondremos en la propiedad de yarn.scheduler.maximum-allocation.

  Para la asignación mínima del scheduler o planificador, en el mismo fichero tendremos que calcular el mínimo. Dado que nuestros nodos tienen un tamaño de 2 GB, el tamaño mínimo será el del valor de 128 para el mínimo.

- Cuánta memoria se asignará a la propiedad ApplicationMaster. Este valor debe ser un valor constante y debe caber en el tamaño máximo del contenedor. Para nodos de 2 GB se podrá asignar el valor de 512 o 256.

- Necesitamos cuantificar cuánta memoria se asignará a cada mapeo u operación de reducción. Esto debería ser menor que el tamaño máximo de la RAM del nodo, para un nodo con 2 GB de RAM, se usará el valor de 256.

Tanto en las configuraciones de ApplicationMaster como la de MapReduce se pondrán con el valor de mapreduce_shuffle que viene por defecto.

El siguiente código 3.21 tiene el fichero yarn-site.xml **igual en ambos *nodos*.**

Tabla 3.21 fichero yarn-site.xml

```
<configuration>

<!-- Site specific YARN configuration properties -->
        <property>
```

```
            <name>yarn.acl.enable</name>
            <value>0</value>
        </property>

        <property>
            <name>yarn.resourcemanager.hostname</name>
            <value>maestroW</value>
        </property>

        <property>
            <name>yarn.nodemanager.aux-services</name>
            <value>mapreduce_shuffle</value>
        </property>

        <property>
            <name>yarn.nodemanager.resource.memory-mb</name>
            <value>1536</value>
        </property>

        <property>
            <name>yarn.scheduler.maximun-allocation-mb</name>
            <value>1536</value>
        </property>

        <property>
            <name>yarn.scheduler.minimum-allocation-mb</name>
            <value>128</value>
        </property>

        <property>
            <name>yarn.nodemanager.vmem-check-enable</name>
            <value>false</value>
        </property>

</configuration>
```

A su vez, necesitamos modificar el fichero slaves donde se encontrarán los nodos datanode, donde se alojan los nodos esclavos. Este fichero se encuentran en el directorio: /home/doime/apacheSpark/hadoop/etc/hadoop. En el pondremos las IPs asociadas a los nodos que previamente hemos modificado en el fichero /etc/hosts asociando las IPs con los nombres esclavoW y maestroW.

Tabla 3.22 fichero slaves

```
esclavoW
maestroW
```

A parte del fichero slaves donde añadimos los nodos esclavos también debemos añadir el nodo maestro al fichero master con el nodo maestroW. El fichero se encuentra en el mismo directorio que todos los ficheros xml de configuración de Hadoop.

Tabla 3.23 fichero master

```
maestroW
```

### 3.3.4 Nodo esclavo

*Hdfs-site.xml*

En el este fichero que tiene Hadoop, tenemos dos campos que rellenar, uno es la replicación de los datos que tenemos, en nuestro caso como tanto el nodo maestro como el nodo esclavo tendrán datos, asignamos el valor de 2, como ya hemos hecho en el nodo maestro. Este valor nunca debe ser mayor al número de nodos esclavos + 1 si el nodo maestro se usa como datanode además de namenode o mayor al número de esclavos si no usamos al nodo esclavo de datanode.

Al ser éste el nodo esclavo, únicamente tendrá la función de almacenamiento de datos, por lo que únicamente habrá dirección para el datanode, en el que pondremos el directorio donde se encuentre el datanode, y cuya ruta es la siguiente: /home/doime/apacheSpark/hadoop/storeMultas/datanode.

Tabla 3.24 Fichero hdfs-site.xml

```
<configuration>
<property>
        <name>dfs.replication</name>
        <value>2</value>
</property>
<property>
        <name>dfs.datanode.data.dir</name>
```

```
        <value>/home/doime/apacheSpark/hadoop/storeMultas/datanode</value>
</property>
</configuration>
```

### 3.3.5 Arranque de Sistema de fichero HDFS

Para arrancar el Sistema de Ficheros de Hadoop es necesario tener encendidos ambas máquinas, antes de inicializar cualquier arranque del sistema, lo más conveniente sería el comprobar si las IPs están bien y comprobar conexión entre ambos equipos. Una vez comprobado que hay conexión, lo que debemos realizar es comprobar si ambos equipos tienen acceso por SSH al otro equipo sin necesidad de autenticar.

Una vez comprobado que todo está correctamente, formatearemos el nodo namenode, este proceso se hará únicamente en el nodo maestro ya que es el nodo namenode, para ello debemos usar el siguiente comando.

Tabla 3.25 Comando formatear HDFS

```
hdfs namenode -format
```

Ya con el namenode formateado, lanzaremos el sistema de fichero, para ello, al igual que ocurrió con la acción de formatear el namenode, se hará en el namenode o también denominado nodo maestro. Para ello usaremos el siguiente comando.

Tabla 3.26 Comando lanzamiento Sistema HDFS

```
start-dfs.sh
```

Puede haber problemas a la hora de conectar ambos nodos, si ocurriera un problema, lo más lógico es buscar en el fichero log que tienen ambos nodos, tanto el namenode como el datanode. En particular debemos tener en cuenta que puede haber problemas con las IPs, ya que pueden estar tomando las IPs de localhost o la dinámica de cada equipo.

Una vez lanzado el sistema, podemos comprobar si éste funciona o no, para ello debemos ejecutar en la línea de comandos la siguiente consulta.

Tabla 3.27 Comando Sistema HDFS

| jsp |
|---|

Esta consulta nos muestra los procesos activos que tenemos en cada nodo, para comprobar debemos usarla en todas las máquinas. Al hacer esta consulta el resultado debe ser el siguiente, donde el número es el identificador del proceso y el nombre son los procesos que están lanzados en los nodos.

Tabla 3.28 Procesos de sistema HDFS

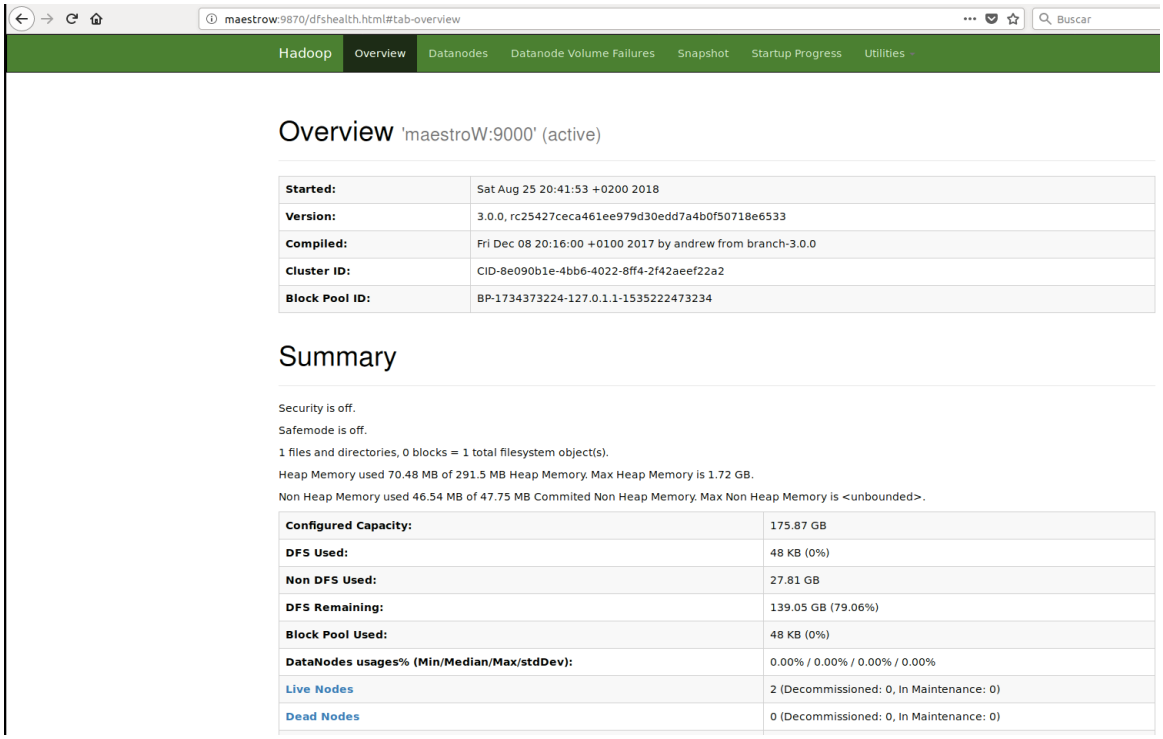
| Namenode | Datanode |
|---|---|
| 6892 Jps<br>6905 NameNode<br>6987 SecondaryNameNode | 4850 DataNode<br>4912 Jps |

Una vez hayamos lanzado el sistemas podemos ver la interfaz del sistema de ficheros HDFS escribiendo en nuestro navegador la IP del nodo maestro con el puerto 9870 que será el asignado por Hadoop para ver la interfaz de los nodos que están en el Sistema. Como se puede ver en el navegador debemos introducir el siguiente texto: https://maestroW: 9870, una vez cliqueamos el texto aparece la interfaz.

Como se puede ver en la imagen, estamos en la visión general del sistema, donde nos da la información más genérica del Sistema, como puede ser la fecha del lanzamiento, la versión, el identificador del *nodo*.

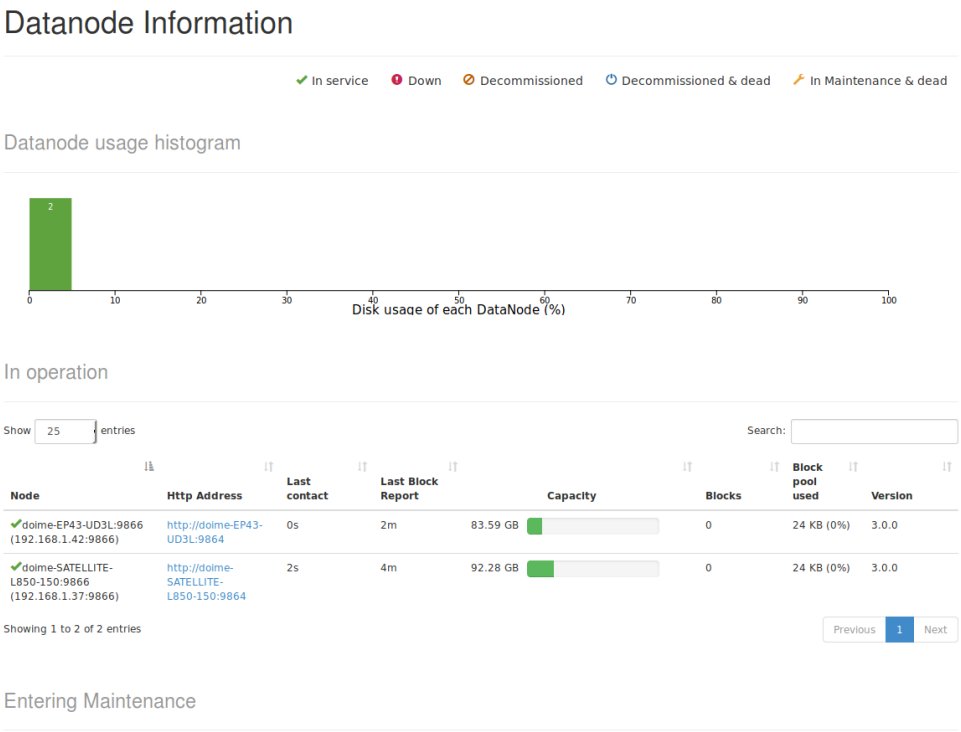
En el resumen se puede observar la capacidad configurada, el porcentaje del Sistema usado y no usado, cuantos nodos hay lanzados y activos, como los desactivados. Además tenemos el máximo de memoria que tenemos configurados en los ficheros xml antes mencionados, que es de 2 GB.

Imagen 3.2 Interfaz general HDFS




Otra imagen que es de importancia para conocer cómo funciona nuestro Sistema HDFS, es la que se encuentra en la pestaña Hadoop, donde podemos ver los datos de los nodos que forman parte del Sistema. Podemos ver cómo están los nombres de los nodos, en los que está doime-EP43-UD3L con IP 192.161.1.42 que es el datanode en este caso el ordenador de sobremesa, y el nodo doime-SATELITE-L850-150 que es el nodo namenode y datanode con IP 192.161.1.37. A su vez nos muestra la capacidad asociada a cada nodo, el último bloque reportado, la última vez en la que han interactuado, versión. Estos datos no son de importancia notable, ya que si se cae un nodo o si no se ha podido conectar, podemos tener conciencia de que no está.

Imagen 3.313 Interfaz de información nodos activos




### 3.3.6 Subida de archivos a HDFS

Para este modo, el distribuido, necesitamos poner los ficheros mediante comandos HDFS, por lo tanto, para la carga de dichos ficheros se usará el siguiente comando. Donde pondremos un fichero unificado llamado multasMadridTotal.csv que será el fichero que deberemos cargar.

Tabla 3.29 Carga de fichero HDFS

| hdfs dfs − put multasMadridTotal.csv datanode / |
|---|

### 3.3.7 Lanzamiento del programa en entorno distribuido

Una vez creado el objeto spark debemos cargar el fichero csv, dado que son varios ficheros csv, optaremos en vez de unificar todos los ficheros en uno sólo, crear una carpeta llamada multas y que se encuentra en el directorio */home/doime/apacheSpark/spark/bin* y tener todas las consultas en la misma carpeta. Por ello en la carga de datos, usamos la barra para decir que todo fichero con formato csv sea cargado. Además en la carga queremos limpiar las columnas de espacios en blanco y asignar a la primera fila como cabecera para

poder realizar consultas. En la consulta 3.30, se verá como usa la dirección donde están los ficheros de multas en el sistema HDFS, conectándose para su uso. El resto de la consulta es igual que en la consulta pseudo-distribuida, que explicaremos más detenidamente en el punto 4.

Tabla 3.30 Lectura de datos en entorno distribuido

```
multa = spark.read.csv("hdfs://masterW:90007/datanode/multasMadridTotal.csv", sep
=
      ↪";", header = True,
↪ignoreLeadingWhiteSpace = True, ignoreTrailingWhiteSpace = True)
```

## 3.4 DataBricks

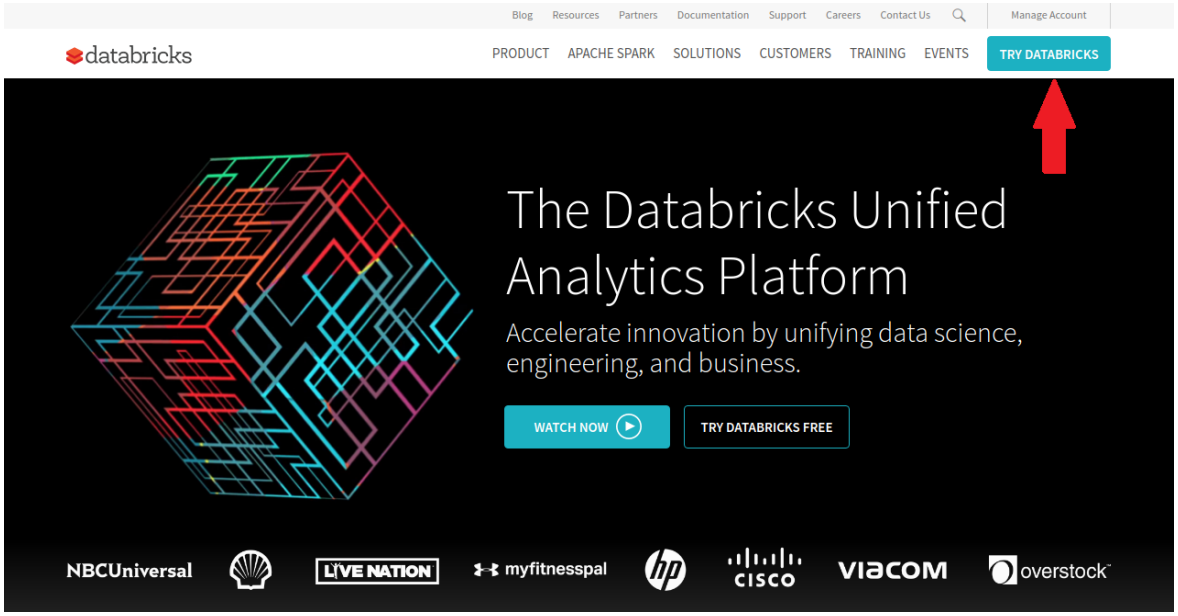
Para hacer uso de este entorno Cloud basado en Apache Spark, debemos saber que para la realización de la consulta haremos uso de datos libres de multas de Madrid, en el portal de datos abiertos del ayuntamiento [21]. En el punto 4 hablaremos más detalladamente de cómo descargar y usar los datos del portal de datos abiertos. Además explicaremos cómo son éstos datos. Podremos comprobar cómo es una de las consultas usadas para la realización de soluciones basadas en Apache Spark. A continuación describiremos detenidamente qué es DataBricks y como usarlo. También en el punto 4, hablaremos de las consultas que realizaremos y en este apartado veremos cómo funciona el entorno Cloud.

Databricks es un software desarrollado en la nube que nos proporciona poder diseñar, unificar y consultar datos. Esta plataforma nos permite disfrutar del entorno Apache-Spark vía Cloud o la nube. Fue diseñada por el propio equipo que diseñó Apache-Spark, nos permite tener una solución para todo tipo de empresa, como podrían ser en sectores financieros, publicidad & Marketing, salud o telecomunicaciones. A parte en sus características clave incluyen; optimizaciones para entornos Cloud, administración de costos, funciones de exploración incorporadas, herramientas de producción integradas y herramientas de seguridad.

En la imagen 3.4 tenemos la interfaz de entrada para poder entrar al entorno Databricks, en la parte derecha tenemos la ventana que nos da acceso a la aplicación. Nos

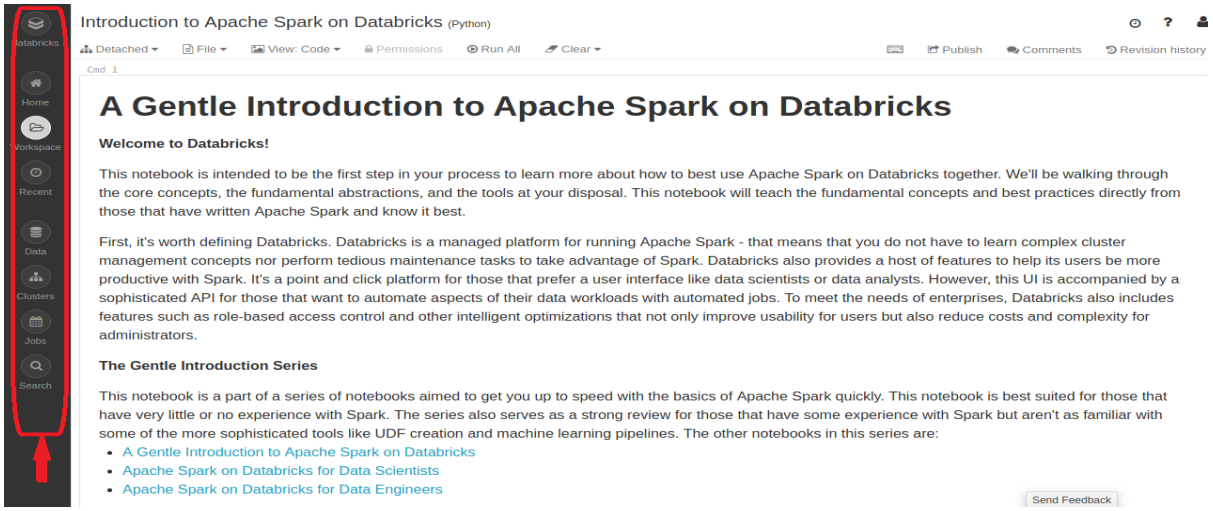
pedirá si queremos que sea personal o empresarial, ya que si es de manera empresarial nos exigirá que tengamos que realizar un pago y en la de prueba o personal, es totalmente gratuita. La mayor diferencia que hay es el tamaño del almacenaje, y la posibilidad del uso de nodos workers a parte del nodo driver. Por ello, nos centraremos en tener un nodo driver que hará de almacenaje para nuestra prueba de multas.

Imagen 3.4 Interfaz de entrada Databricks



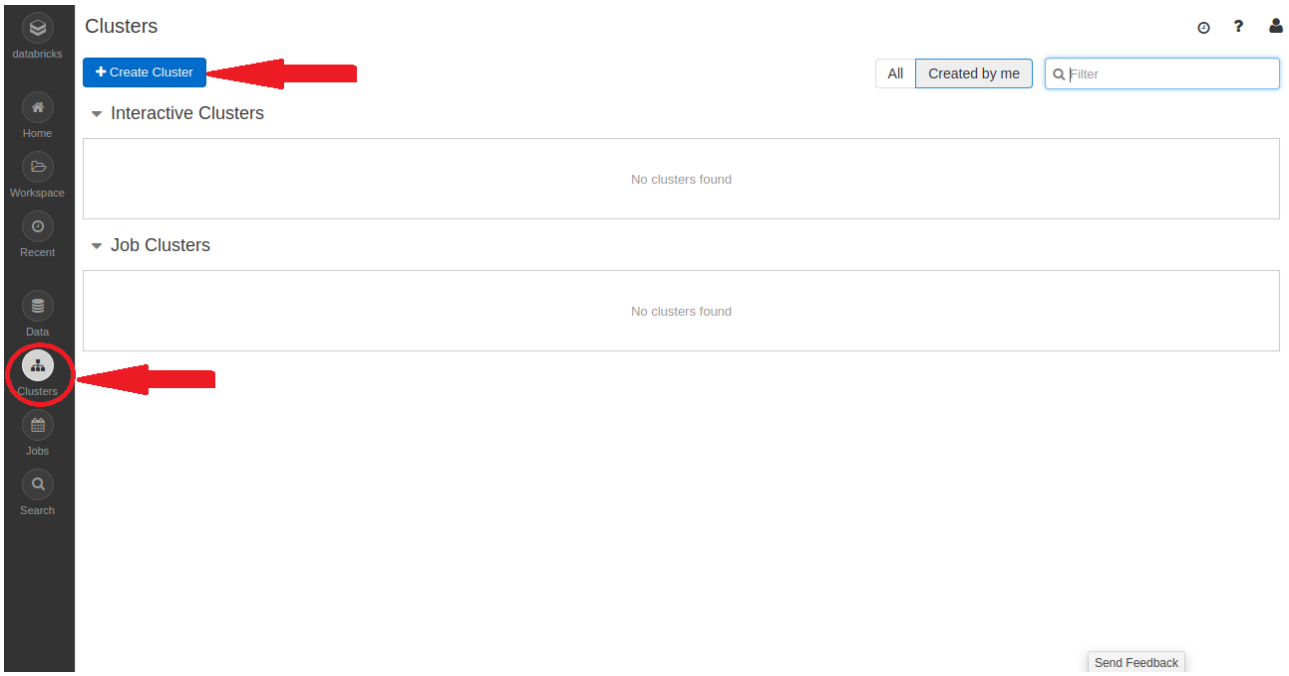

Como veremos a continuación, podemos observar que una vez entramos en la plataforma, tenemos un pequeño tutorial de lo que es apache-Spark, de cómo usar databricks y además de las posibles acciones que podemos realizar. En la parte derecha se encuentra marcado la barra de actividades de Databricks, en ella podemos encontrar todas las acciones que queramos para nuestro trabajo, en ellas tenemos los siguientes botones y menús que más adelante explicaré centrándonos en los las importantes y los que he usado. Databricks, Home, Workspace, Recent, Data, *Clúster*s, Jobs y Search.

Imagen 3.5 Introducción Databricks




En la imagen que hay abajo, nos centraremos en la creación del *clúster* de trabajo, como se puede observar, nos dirige a otro menú, en él podemos encontrar un botón que nos permite crear un nuevo *clúster*, pulsaremos este botón para crear un nuevo clúster.

Imagen 3.6 Creación *clúster* Databricks

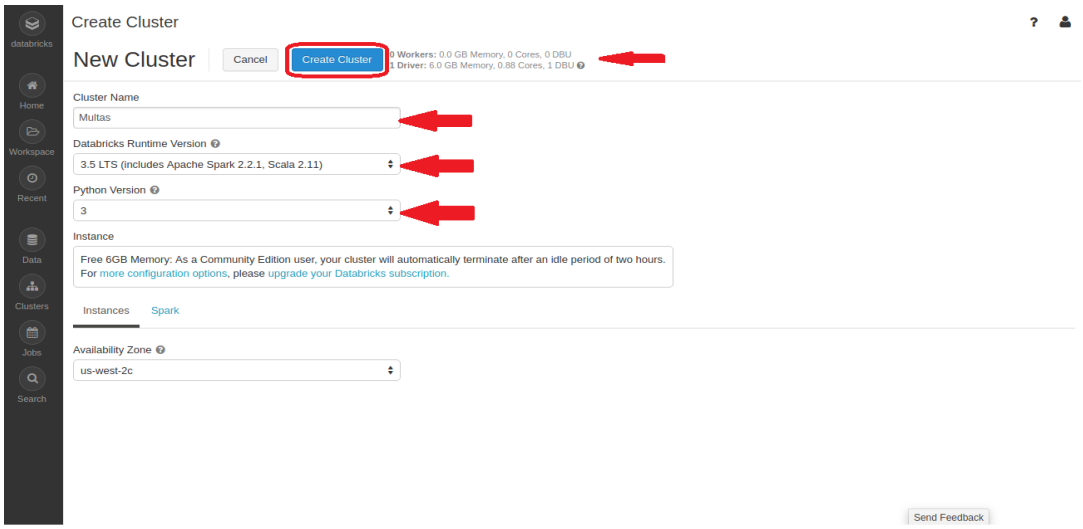



Una vez pulsado el botón nos abre una nueva ventana en la que observaremos las características del *clúster* y la información a rellenar que a continuación explicaremos:

- *Clúster* Name: será el nombre que le queramos poner al *clúster* para su uso e identificación, en mi caso se llamará multas.
- Databricks Runtime Version: en la que pondremos la versión que queremos usar para la realización de las consultas. En este caso usaremos la versión última en la que se hizo el trabajo, 3.5 LTS.
- Python Version: dado que hemos decidido usar para la realización de las consultas y la instalación del entorno la versión 3.0, será esta la que decidiremos usar también en el entorno cloud para reutilizar el código.
- Instance: nos informa de la capacidad que tenemos adquirida en el entorno Databricks.
- Instances: dejaremos por defecto la que nos viene impuesta.
- Spark: nos da la posibilidad de usar alguna configuración que creamos conveniente, en nuestro caso lo dejaremos vacío.

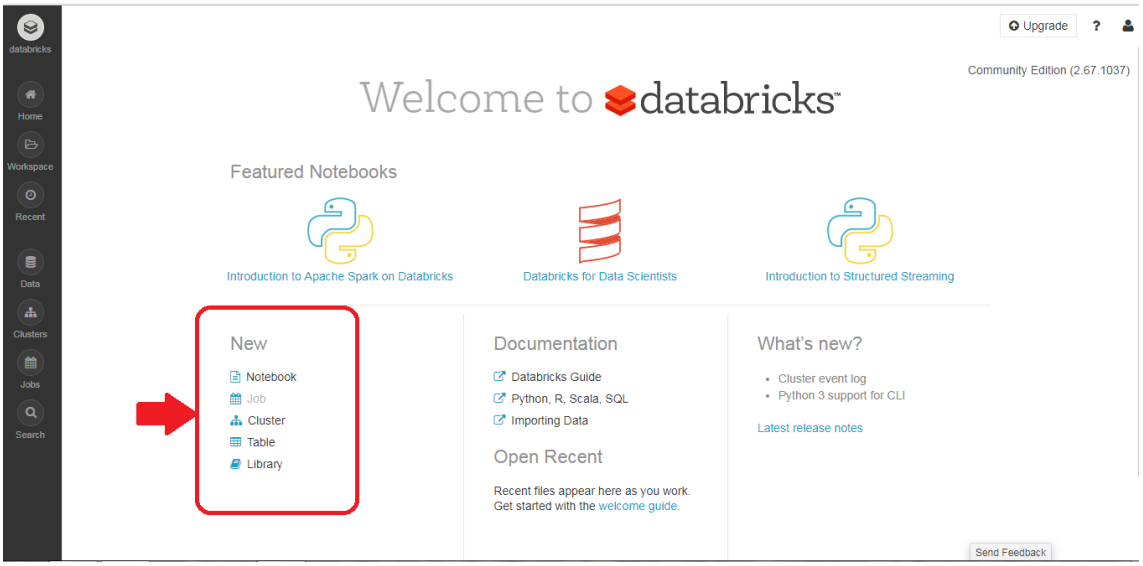
Una vez rellenado todos los campos, nos dispondremos a crear el *clúster*, pulsando el botón Create *Clúster* que viene marcado en línea roja y señalado con una flecha.

Imagen 3.7 Creación de nuevo clúster




En la pantalla de inicio donde viene marcado con el icono Databricks se puede ver un menú en el que tenemos la posible creación de distintas opciones, como son Notebook, *clúster*, tablas para el uso de datos en las consultas. Debajo de esta imagen explicaremos qué hacen cada una de los iconos marcados: Notebook, *Clúster*, Job, Table y Library.

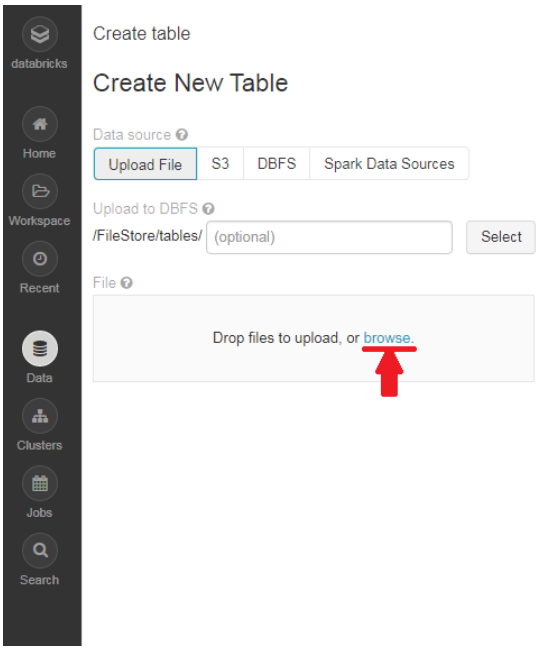
Imagen 3.8 Interfaz Databricks




## 3.4.1 Table

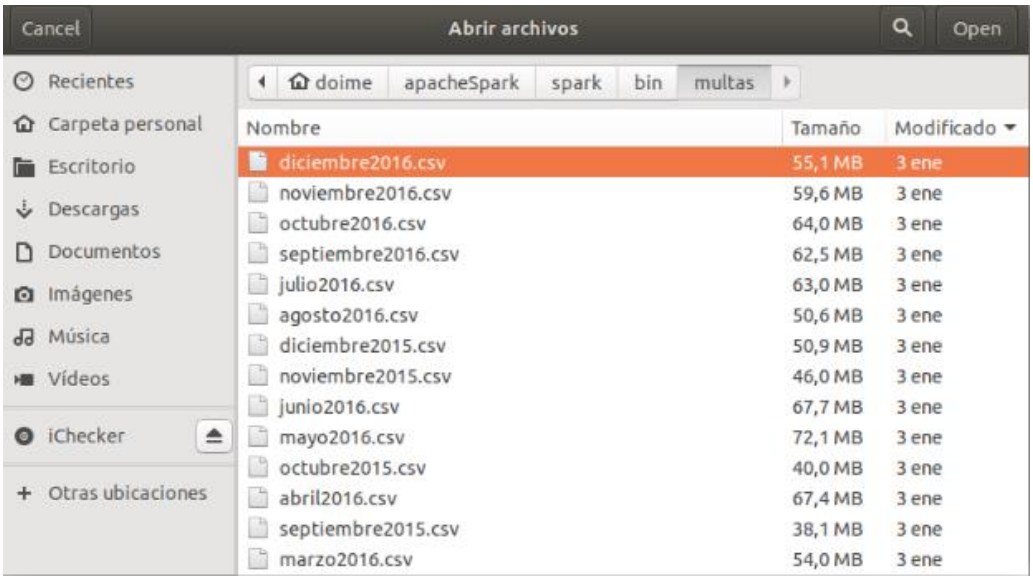
Este segmento del entorno se encarga de la subida de datos para poder ser usada en las consultas a realizar. Como vemos subrayado en rojo; tenemos browse, un enlace a nuestros datos en el ordenador para realizar la subida de los ficheros que deseemos, en nuestro caso serán las multas de Madrid, que tenemos almacenadas en la carpeta multas, donde se encuentran todos los ficheros de multas csv con el que contamos.

Imagen 3.9 Creación de tabla Databricks



Una vez pulsado *browse* se nos abre una ventana como la que muestro aquí abajo, donde seleccionaremos todos los archivos que deseamos subir a la plataforma Databricks. En nuestro caso serán todos los archivos descargados y guardados por mes.

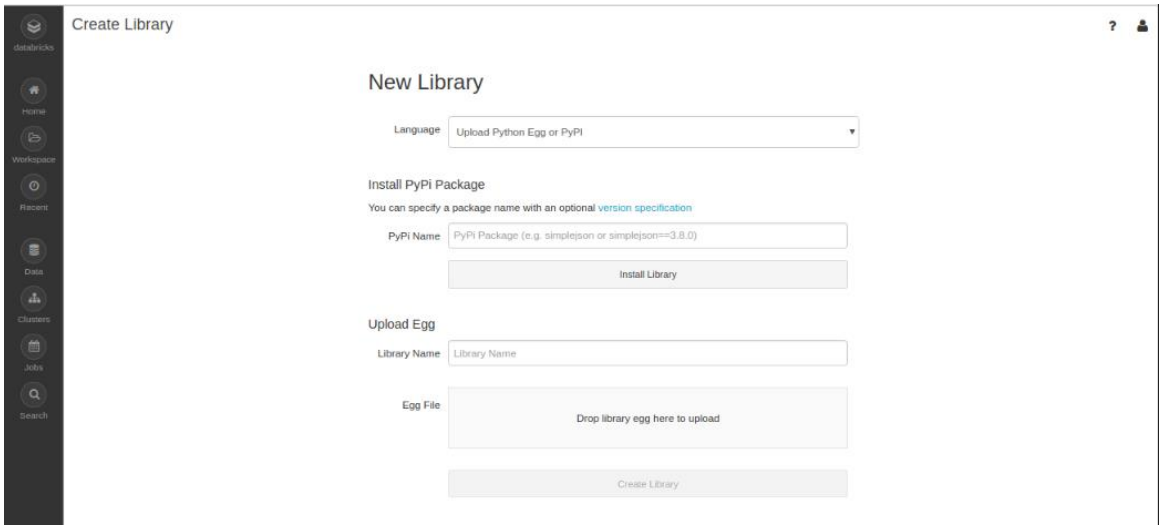
Imagen 3.10 Subida archivos a Databricks




### 3.4.2 New library

En new Library, podremos instalar para el uso del mismo nuevas librerías de Python por lo que debemos seleccionar en language Upload Python Eggs or PyPI, para decir qué tipo de lenguaje queremos actualizar. Una vez seleccionado, pondremos el nombre de la librería que queramos cargar a nuestra cuenta en el apartado Install PyPi Package y dar al botón de Install Library, en nuestro caso al no hacer uso de ninguna librería específica este campo no lo utilizaremos.

Imagen 3.11 Creación nueva librería Databricks



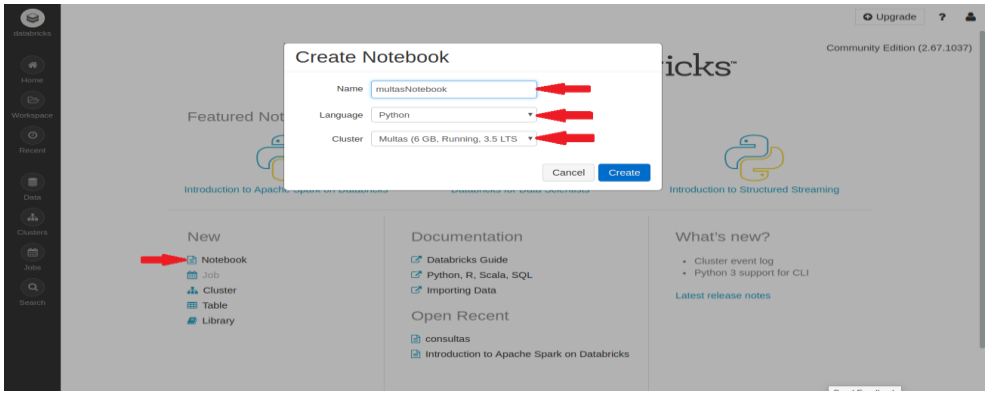

### 3.4.3 Notebook

Antes de nada explicaremos qué es un notebook y para qué nos sirve. Un notebook es una aplicación web, que nos permite ejecutar fragmentos de código y a su vez representar texto con formato. En nuestro caso únicamente haremos uso de la variante de código, ya que es nuestro objetivo. Estos notebooks guardaran nuestro código y además posibles comentarios que sean necesarios.

Nosotro crearemos un Notebook como aparece en la imagen de abajo, en el que le asignaremos los siguientes campos:

- Name: el nombre que queremos que lleve el notebook, en nuestro caso será multasNotebook
- Language: es el lenguaje en que hemos realizado las consultas y las especificaciones realizadas hasta el momento.
- *Clúster*: será el *clúster* creado al principio de éste punto, donde le pusimos el nombre de multas.

Imagen 3.12 Creación notebook Databricks




Una vez creado usaremos el notebook para realizar las consultas que nosotros queramos, como podemos observar en la imagen abajo dada, hemos cogido el programa escrito en Python con la consulta conveniente, este notebook es muy parecido al notebook Jupiter de Python, por lo que la utilización es exactamente igual que Jupiter.

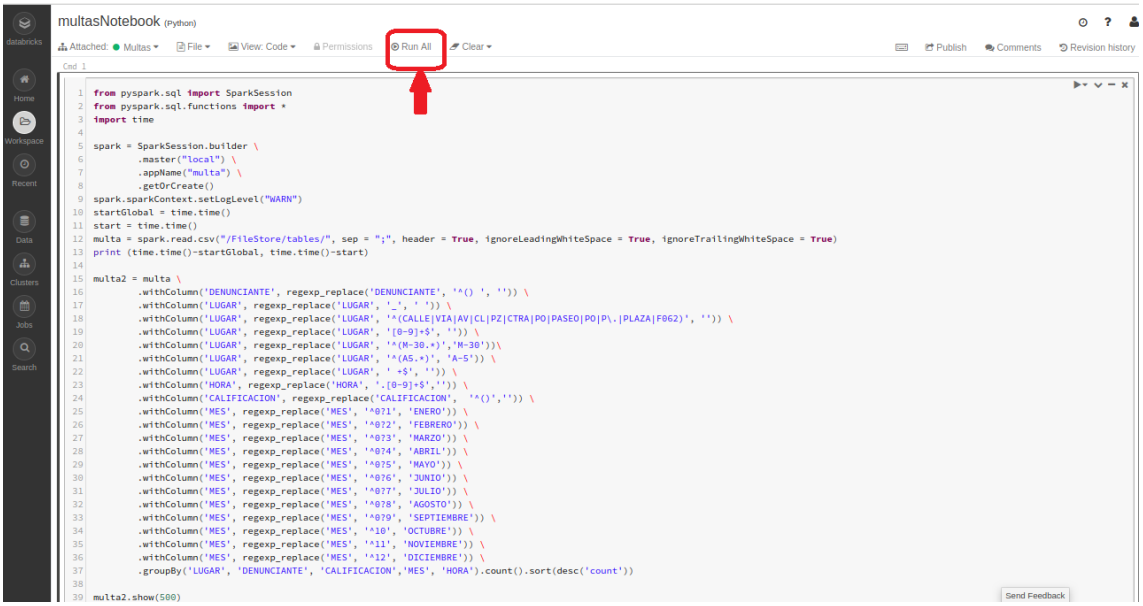
Una vez realizado el programa, podríamos copiar exactamente el mismo programa que ya hemos realizado en el sistema pseudo-distribuido, únicamente debemos cambiar la carga de datos, ya que al estar en entorno cloud, no obtenemos los datos de nuestros propios ficheros. Como hemos explicado anteriormente, en el campo table hemos cargado todos los ficheros a guardar. Por lo que usaremos este fichero para realizar la consulta. Para ello usaremos el siguiente fragmento de código para su carga correcta.

Tabla 3.31 Lectura ficheros mediante Databricks

```
spark.read.csv("/FileStore/tables/", hearder = True, ignoreLeadingWhiteSpace = True,
↪ ignoreTrailingWhiteSpace = True)
```

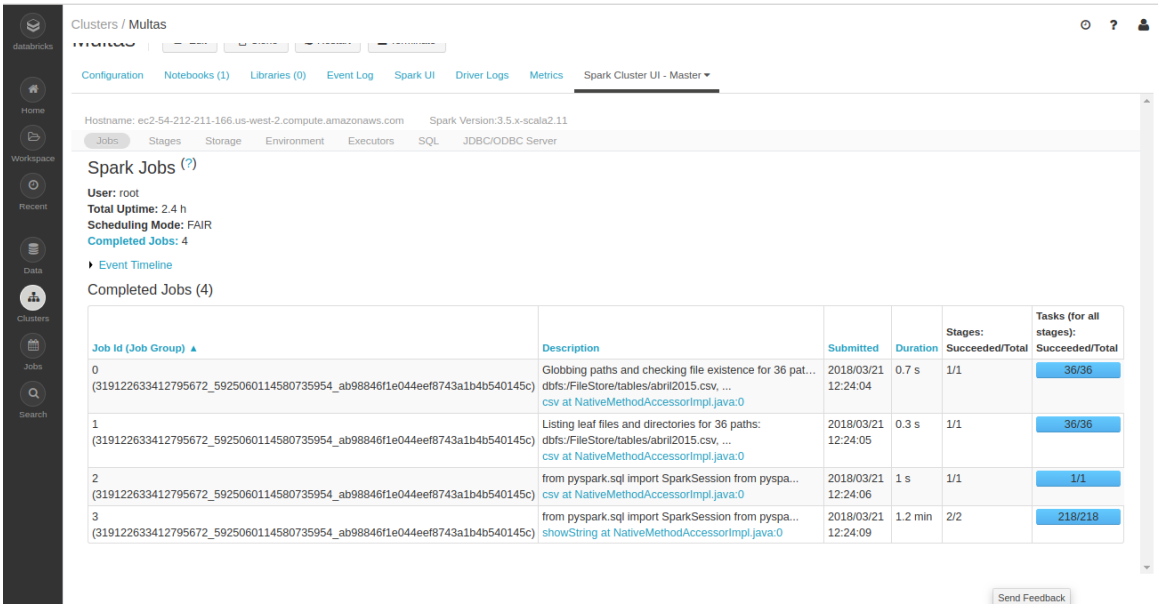
Una vez hemos tenido el programa hecho correctamente, haremos click en el botón abajo marcado "*Run all*" podremos ver cómo actua apache-spark y que abajo veremos.

Imagen 3.13 Código consulta Databricks




Una vez lanzado la consulta, podemos ver en el icono de *clúster* los trabajos que han ocurrido, como son la carga y las transformaciones y acciones de los RDD`s. Abajo vemos los cuatro trabajos que se realizan en la consulta.

Imagen 3.14 Trabajo 1 *clúster* Databricks



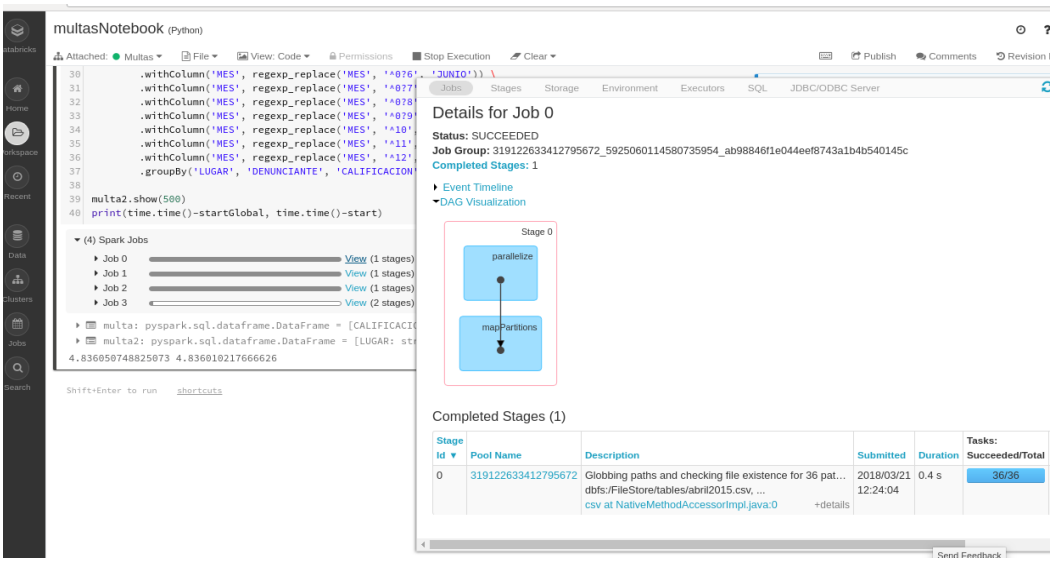

Una vez visto el global, podemos ver qué acción o transformación ha ocurrido en cada trabajo o job, empezaremos explicando el primero de los trabajos y el diagrama de visualización DAG, en el que nos muestra el gráfico o estación ejecutado para cada trabajo,

en nuestro ejemplo tendremos cuatro trabajos llamados job0, job1, job2 y job3. Estos trabajos pueden mostrarnos más de una operación de RDD y dentro de cada RDD las operaciones correspondientes, todo esto quedará reflejado en el DAG Visualización que únicamente tendremos que clickear en el nombre. En el trabajo 0 o job 0 como nos lo nombra Databricks

En la imagen abajo mostrada podemos observar los siguientes estados en el diagrama:

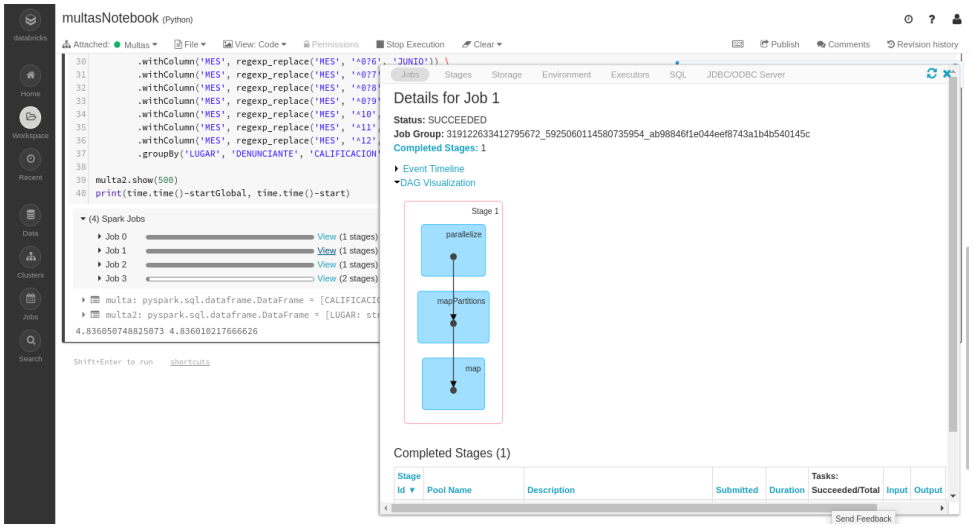
- Parallelize: este estado nos va a hacer que los elementos de la colección se copien para formar un DataSet distribuido que se puede operar en paralelo.
- MapPartitions: es una transformación de un RDD, convierte cada partición de la fuente RDD en varios elementos del resultado. Por lo que ejerce la función a nivel partición. Si un RDD tiene varios elementos, con MapPartitions llamará a la función por parámetro una sóla vez, pasando todos los registros y recuperando las respuestas en una llamada.

Imagen 3.15 DAG Job 0  Databricks



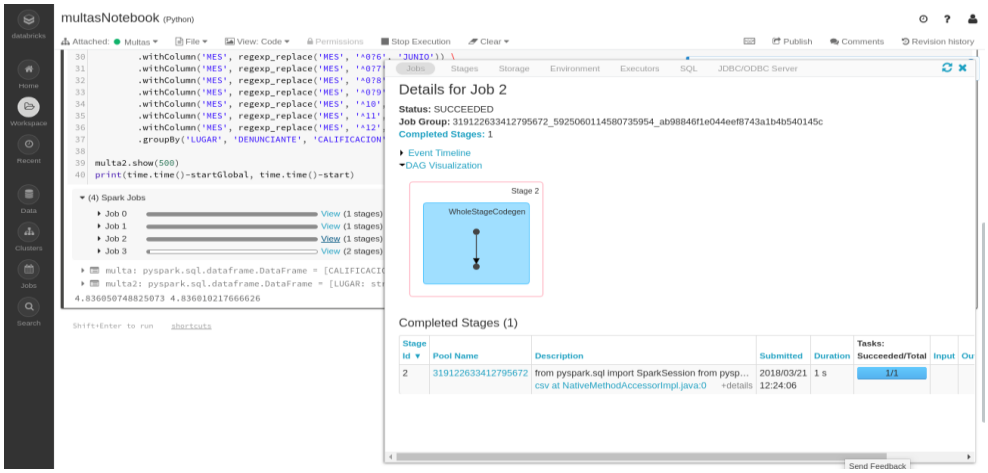

Una vez pasado por el trabajo 0, entramos al siguiente job, que nos nuestra exactamente las mismas etapas que en el job 0, pero esta vez añade un estado nuevo, el map, en este paso, map lo que nos va a hacer es la transformación de mapeo, tantas veces como elementos haya. Es una gran diferencia respecto a la función de MapPartitions, ya que como podemos observar en map se llama a la función tantas veces como elementos haya, sin embargo en MapPartitions tenemos una única llamada a la función por parámetro.

Imagen 3.16 DAG Job 1  Databricks




A continuación tenemos el job 2, en el que únicamente existe un estado. Este estado es el WholeStorageCodigen [41] Este estado fusiona operadores múltiples. Esta fusión la hace como si fuera un subárbol de planes que soportan codegen en una función Java que tiene como objetivo mejorar el rendimiento de la ejecución. Básicamente lo que hace es colapsar una consulta en una única función optimizada que elimina las llamadas a funciones virtuales y aprovecha registros de CPU para datos intermedios.

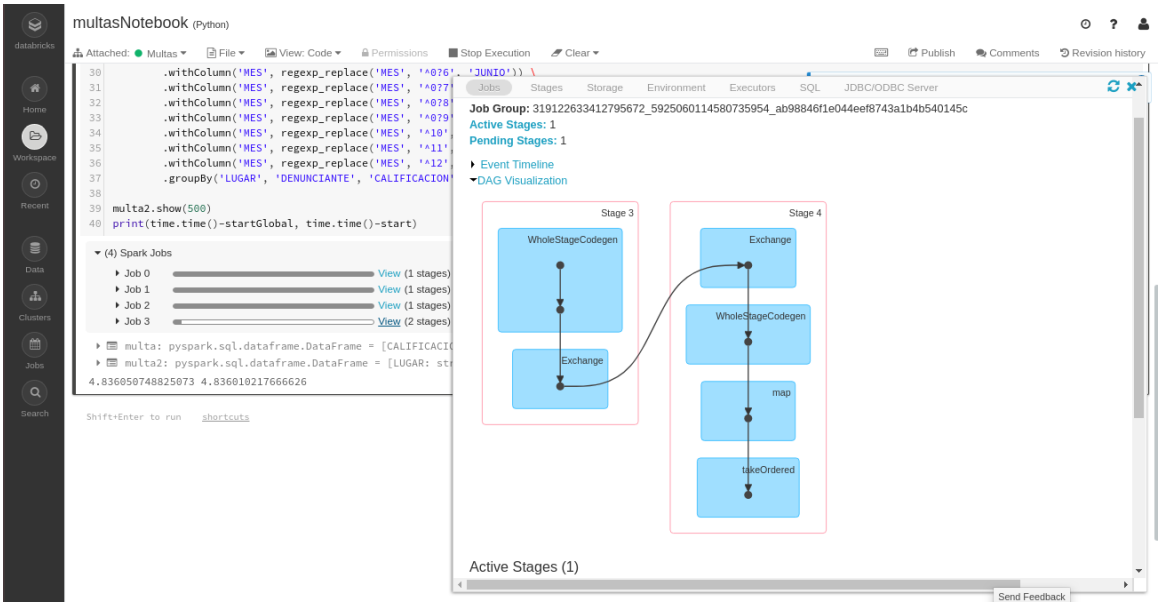
Imagen 3.17 DAG Job 2  Databricks



Como último trabajo tenemos el job 3 en el que acaba la consulta. En éste caso se compone de dos Stage, el 3 y 4.

Podemos ver que en la etapa o Stage 3 Whole colapsa la consulta en una única función optimizada que elimina las llamadas virtuales, como hemos hablado antes, esto pasa al estado exchange, para saber más sobre ésta etapa, explicaremos qué es éste estado: en “Exchange” nos muestra un intercambio aleatorio entre los trabajos que existen, este estado no tiene generación de código alguna, sólo envía datos a través de la red y nos devuelve un nuevo RDD. Acabada la etapa 3, Exchange nos devuelve un nuevo RDD que es optimizado como ocurre en la etapa 3. Una vez optimizado, este nuevo RDD es mapeado, este paso ha sido explicado en el Job 1, por lo que no nos centraremos más en él. Una vez ha sido mapeado el RDD y realizado las pertinentes transformaciones, pasa el RDD a la etapa takeOrdered, que hará la siguiente acción, devolverá los primeros n elementos del RDD usando su orden natural o un comparador personalizado, en este caso su orden natural.

Imagen 3.18 DAG finalización del trabajo  Databricks



# 4

# Escenario de aplicación

En este apartado veremos la estructura y los datos que corresponden al análisis de datos que hemos elegido como ejemplo, a la hora de desarrollar el proyecto.

Para poder escenificar y comprobar el escenario de *Apache-Spark* decidimos usar los ficheros que el ayuntamiento de Madrid sube de manera libre en el portal de datos abiertos [21], en el que podemos encontrar por mes todas las multas que han ocurrido en la capital, pudiendo así descargar de manera sencilla cada fichero con formato csv. Dado que no son ficheros de un tamaño importante para el uso del sistema, lo que haremos será descargar las multas desde el año 2015, con lo que tendríamos los datos de las multas de los últimos 3 años, no obstante en el momento de obtener dichos datos, los meses de noviembre y diciembre de 2017 no se encontraban en el portal comentado anteriormente. Dado que en el proyecto usaremos la plataforma *Databricks*, y su límite de tamaño para usuarios exentos de pago es de 2 Gigabyte (Gb) aprovecharemos que la suma de todos los ficheros csv en su conjunto son de un total de 1.8 Gb.

El fichero será guardado en una carpeta llamada *'multas'* en el entorno pseudo distribuido estará en el directorio *'/home/doime/apacheSpark/spark/bin'* donde será necesario una vez lancemos el programa mediante spark-submit.

Como nuestro objetivo es el de conocer qué opción es mejor y qué configuración nos mejora el rendimiento de nuestro clúster, no será necesario guardar en un fichero las consultas a realizar.

## 4.1 Descarga de ficheros

Para hacer uso de los datos de multas, debemos introducirnos en el portal de datos abiertos del ayuntamiento de Madrid. Estas multas se pueden descargar de manera libre, sin necesidad de pedir autorización ni de problemas con la Ley de Protección de Datos.

El acceso se hará en el siguiente link https://datos.madrid.es/portarl/site/egob. En él tenemos que navegar y acceder al apartado "Catálogo de datos", una vez dentro, veremos al final de la página una pestaña con el título "Publicación de Datos". En este menú desplegable aparecen los datos mes a mes en dos formatos: .csv y .txt, por lo que descargaremos cada mes en el formato .csv.

Cuando estén todos los datos descargados, tendremos que unificar todos los ficheros en uno sólo, del que la aplicación hará uso de los datos para hacer las consultas que deseemos realizar.

## 4.2 Descripción del escenario de la aplicación

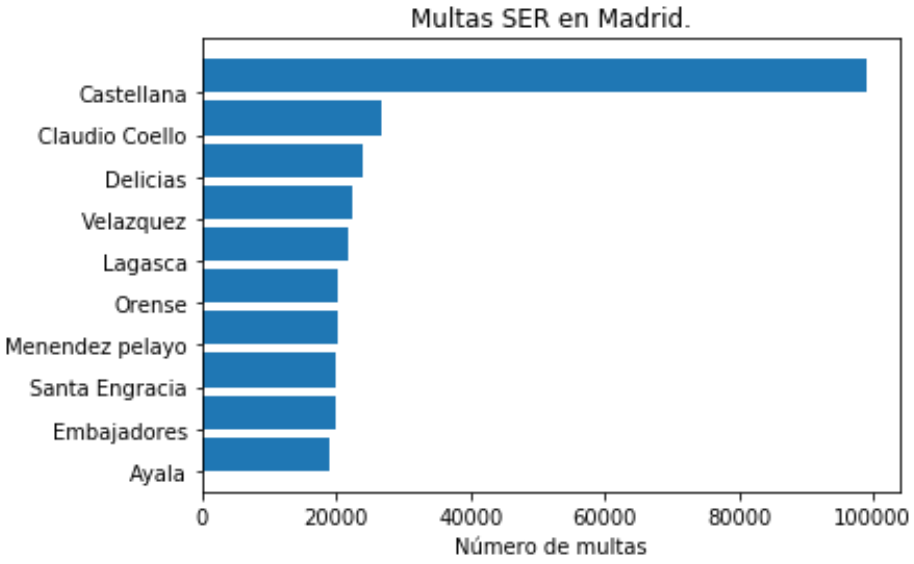
Para explicar el uso anteriormente comentado de los ficheros de multas que Madrid pone al alcance de nuestra mano, centraremos nuestro punto de vista como un escenario real para la explicación de las consultas realizadas y la relación de las multas con la situación de la ciudad y de la distribución de las calles.

Por eso hemos decidido centrar este escenario en dos consultas para que sea relevante el uso de este sistema; una primera consulta que nos dirá dónde se multa más en Madrid el Servicio de Estacionamiento Regulado, y otra que veremos quién es la autoridad que más multa en la capital.

En primer lugar nos centraremos en el SER o Servicio de Estacionamiento Regulado. Para que entendamos mejor la relación, explicaremos brevemente qué es este servicio y cuál es su función. Los controladores e inspectores de este servicio, recorren la ciudad para garantizar que tanto ciudadanos como foráneos cumplen las normas de aparcamiento en las zonas donde existe esta limitación del servicio, comúnmente conocidas como zona azul y zona verde. Aparte de revisar que las personas que han aparcado en estas zonas han pagado la estancia correctamente, también se encargan de multar a los que se detengan en doble fila, ante un vado o en un carril-bus obstaculizando el tráfico (este último servicio lo comparten con los agentes del SACE *Servicio de Apoyo al Control de Estacionamiento*).

Nos centraremos en las calles donde mayor impacto tienen las multas, hemos decidido sacar las 10 calles con más número de multas de madrid. Figura 4.1

Imagen 4.1 Multas SER Madrid.

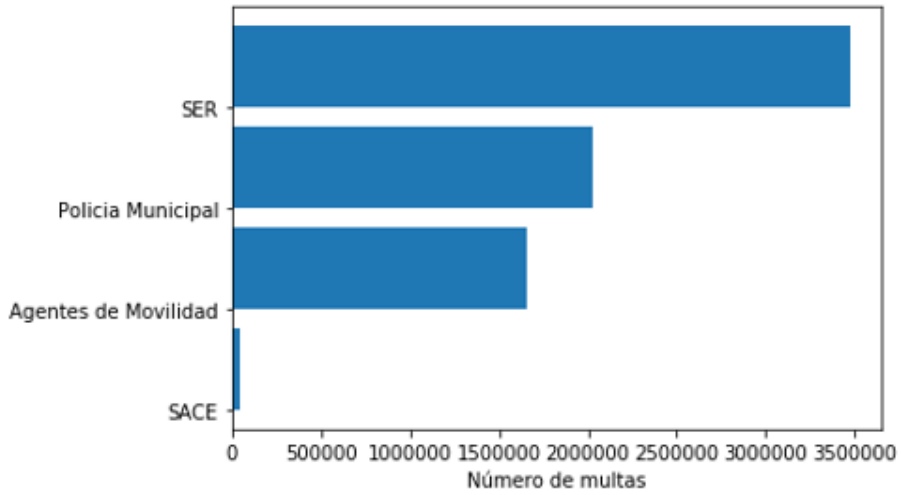



Se observa que la vía con más infracciones es elPpaseo de Castellana, muy por delante de la segunda, que en este caso es la calle Claudio Coello. Se observa que la mayoría de las calles son o pasan por el distrito de Salamanca, uno de los más ricos y lujosos de Madrid, donde se concentran la mayoría de las tiendas y zonas exclusivas de Madrid. También se observa cómo otras vías son un acceso de la zona periférica de la ciudad a la zona centro de Madrid como pueden ser Delicias o Embajadores.

Otro resultado que es llamativo es el hecho que varias calles de las más multadas, se encuentran en lugares en los que es necesario realizar trámites como en Ministerios, Consejerías y Juzgados como los de Plaza Castilla o los de Capitán Haya. Esto puede deberse a la mala planificación a la hora del uso del horario de estacionamiento o a la deriva en el trámite que se tenga que hacer. Además de estos lugares, se muestra cómo tiene un incremento en zonas donde hay bajada y subida de viajeros como los alrededores de la estación de Atocha o en la Estación Sur de Autobuses. Por otra parte se ve un incremento de multas en lugares cercanos a centros médicos como es el caso de  Menéndez Pelayo, que se encuentra al lado del Hospital Universitario Infantil Niño Jesús.

Una vez desarrollado el punto de donde ponen más multas en Madrid el SER, iremos al total de multas que ocurren en la capital entre las cuatro autoridades implicadas en la implantación de las multas en la capital. Como observaremos en el siguiente gráfico es el Servicio de Estacionamiento Regulado el que impone más multas a los ciudadanos.

Imagen 4.2 Total multas agentes en Madrid




Como se observa, las infracciones más usuales que se cometen en la ciudad son de carácter leve (véase anexo en consulta 1.1) este tipo de infracciones son producidas por algún tipo de irregularidad en el estacionamiento del vehículo, ya sea en forma de sobrepasar el tiempo límite o cómo el “olvido” de manera involuntaria o voluntaria del pago del estacionamiento por un tiempo. Además como hemos explicado anteriormente, el SER también se encarga del estacionamiento de los vados o del carril bus, siendo éste último de carácter grave.

Al contrario que ocurre con el SER, la Policía Municipal, la mayoría de sus multas son por límite de velocidad, hay que tener en cuenta que todos los radares móviles y fijos de la capital son competencia de la Policía Municipal. Estos radares los más importantes son los que encontramos en la calle M-30, teniendo en cuenta que los radares de las autovías no son competencia de la Policía Municipal, por ello gran parte de sus multas son de carácter grave o muy grave. Debemos tener en cuenta que saltarse un semáforo en rojo es una falta grave, siendo los semáforos con cámara que controlan si se ha pasado en rojo o no, una clara aportación a las multas que son notificadas y tramitadas por la Policía Municipal.

Los Agentes de Movilidad, se centran en el control del tráfico de la ciudad, siendo ellos los principales responsables de las multas de circulación como podrían ser saltarse semáforos, STOP o aparcar en zonas prohibidas como pueden ser carga y descarga. Su trabajo se centra principalmente en las zonas más confluidas de la ciudad. Este trabajo es compartido en su mayor parte con la policía Municipal.

En cambio el SACE o Servicio de Apoyo al Control de Estacionamiento sólo se dedican a perseguir a infractores del estacionamiento del Carril Bus, por lo que su número de multa es casi despreciable respecto a los otros Agentes de Autoridad.

Los datos obtenidos para justificar este escenario se encuentran en el ANEXO como *consulta SER* y *consulta Agentes*, además de las consultas para obtener los datos antes dichos y dados en las gráficas.

## 4.3 Descripción de los datos de los ficheros

En este apartado se analizará el fichero de datos de las multas de Madrid y describiremos el proceso de filtrado y tratamiento de datos. Este componente será el mismo en cada una de las tres configuraciones de arquitectura planteadas. Por lo que el código será exactamente el mismo, solo cambiando la manera de acceder a los datos.

Por lo que si hacemos una consulta de cuál es nuestra primera línea de nuestro fichero csv tendremos el siguiente extracto de datos:

Tabla 4.1 Datos ficheros CSV

| |
|---|
| CALIFICACION , LUGAR , MES , ANIO , HORA , IMP_BOL , DESC , PUNTOS , DENUNCIANTE , HECHO_BOL , VEL_LIMITE , VEL_CIRCULA , COORDENADA_X , COORDENADA_Y |

Debemos tener en cuenta que no todos los campos anteriormente citados están rellenos, como ocurre en COORDENADAS_X, COORDENADAS_Y y en VEL_LIMITE, siendo esto por el redactado del fichero por parte del Ayuntamiento de Madrid.

A continuación describiremos cada campo y su significado.

Tabla 4.2 Explicación datos fichero CSV

| | |
|---|---|
| CALIFICACION | Nos da el tipo de calificación de la sanción impuesta al infractor, hay varios tipos de sanciones, que son: LEVES, GRAVES o MUY GRAVES. Así intentica el ayuntamiento el tipo de infracción que ha ocurrido en sus calles. |
| LUGAR | Viene dado por el lugar exacto de la infracción, por lo que en el campo saldría el tipo de vía: CL, CALLE, AV, PASEO, PLZ… después vendría el nombre de la vía, y por último el número donde sucedió. Por ejemplo: CL MAYOR, 12 |
| MES | El campo viene relleno según qué mes haya ocurrido, no viene dado en nombres, por lo que el mapeo sería el siguiente: 1-Enero, 2-Febrero, 3-Marzo, así sucesivamente hasta llegar al 12-Diciembre. |
| ANIO | En este caso sólo vendrá dado por 2015, ya que las multas sólo engloban este año. |
| HORA | La hora exacta a la que se produjo la infracción, por ejemplo 08.31. Este campo viene dado por el sistema horario de 24 horas. |
| IMP_BOL | Es el importe de la multa que se ha puesto, desde la más baja de 30 a la más alta de 200 €. |
| DESC | Nos indica si la persona multada ha elegido hacer uso del |

| | |
|---|---|
| | descuento que puede ser aplicado. |
| PUNTOS | Son la cantidad de puntos que se han retirado en la infracción |
| DENUNCIANTE | Es la autoridad que ha realizado la denuncia al infractor. En el caso de Madrid serían la POLICIA MUNICIPAL, SER(Servicio de Estacionamiento Regulado), SACE (Servicio de Apoyo al Control de Estacionamiento) y AGENTES DE MOVILIDAD |
| HECHOS_BOL | Nos dará el motivo por el que ha sido multado |
| VEL_LIMITE | Nos indica la velocidad máxima a la que se debería ir, no obstante, en este campo entra otro tipo de datos como multas por pasar el tiempo de estacionamiento. |
| VEL_CIRCULA | Nos indica a la velocidad que iba el usuario del vehículo, siendo este campo aplicable a otro tipo de indicación como a las multas de estacionamiento. |
| COORDENADA_X | Debería darnos la coordenada concreta en el eje x de la multa, pero este campo está vacío como anteriormente se ha mencionado |
| COORDENADA_Y | Debería darnos la coordenada concreta en el eje y de la multa, pero este campo está vacío como anteriormente se ha mencionado |

## 4.4 Consultas a realizar

En este apartado, hemos decidido realizar como ejemplos distintas consultas, ya que el tema tiene un interés a la hora de saber en qué lugar hay más multas, quienes son los responsables de ejecutarlas y las horas en que hay más multas, esto nos aportará de qué manera se distribuyen las infracciones; lugar, hora calificación y la cantidad de la misma.

A continuación explicaremos cómo hemos realizado la consulta en entorno pseudo-distribuido.

En primer lugar debemos importar de pyspark, las funciones necesarias para la realización de la consulta, además hemos optado por imprimir en la respuesta el tiempo que tarda en cargar los datos y el tiempo total hasta que sale la respuesta.

Tabla 4.3 Código importación de librerias

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
import time
```

Es necesario el uso de un objeto SparkSesion (ss) que crearemos asignando la máquina master a la IP local y cuyo nombre será multa.

Tabla 4.4 Código para iniciar sesión

```
spark = SparkSession.builder \
          .master("local") \
         .appName("multa") \
          .getOrCreate()
```

Una vez creado el objeto Spark debemos cargar el fichero csv, dado que son varios ficheros csv, optamos en vez de unificar todos los ficheros en uno sólo, crear una carpeta llamada multas y que se encuentra en el directorio */home/doime/apacheSpark/spark/bin* y tener todas las consultas en la misma carpeta. Por ello en la carga de datos, usamos la barra para decir que todo fichero con formato csv sea cargado. Además en la carga queremos limpiar las columnas de espacios en blanco y asignar a la primera fila como cabecera para poder realizar consultas.

Tabla 4.5 Código lectura multas entorno pseudo-distribuido

```
multa = spark.read.csv("multas/", sep = ";", header = True,
↪ignoreLeadingWhiteSpace = True, ignoreTrailingWhiteSpace = True)
```

Ya con los ficheros cargados en nuestro RDD (Resilient Distributed Dataset) en vez de hacer uso del método filter que nos proporciona Python y Apache-Spark, hemos optado en limpiar las columnas, ya que éste método sólo filtra según cadenas de texto exactas, por lo cual nos sería costoso filtrar todas las calles de sin números y sin que sea especificado el tipo de vía. Por lo que crearemos una nueva lista que reemplace lo que no queremos que se vea por una lista limpia. Por lo que usaremos el método regexp_remplace que nos reemplaza la columna seleccionada por otra con las características que deseamos, partiendo

de la que tenemos cargada en el objeto RDD. En este caso, por ejemplo hemos decidido eliminar de las calles tanto el número donde ha ocurrido la infracción como el tipo de vía.

Tabla 4.6 Código de la consulta entorno pseudo-distribuido

```
multa2 = multa.withColumn('DENUNCIANTE',
      ↪regexp_replace('DENUNCIANTE', '^() ', '')) \
          .withColumn('LUGAR', regexp_replace('LUGAR',
      ↪'^(CALLE|VIA|AV|CL|PZ|CTRA)', '')) \
          .withColumn('LUGAR', regexp_replace('LUGAR', '[0-9]+$', '')) \
         .withColumn('HORA', regexp_replace('HORA', '.[0-9]+$','')) \
         .withColumn('LUGAR', regexp_replace('LUGAR', '^(M-30.*)$)','M-30'))\
          .withColumn('LUGAR', regexp_replace('LUGAR', '^(M-30 CALZADA 1
↪KM 19,)','M-30'))\
          .withColumn('MES', regexp_replace('MES', '^()', '')) \
          .groupBy('LUGAR', 'DENUNCIANTE', 'MES',
      ↪'HORA').count().sort(desc('count'))
```

Por lo tanto en la consulta que abajo veremos, tenemos que crear una lista que nos mostrará en orden decreciente, según el número de multas que haya habido, los datos correspondientes a los denunciantes, lugar, hora y mes. Agrupando las calles/carretera con más denuncias el mes en que ha ocurrido y los denunciante

Tabla 4.7 Código Consulta completa. callesMesHoraDenun.py pseudo-distribuido

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
import time

spark = SparkSession.builder \
          .master("local") \
          .appName("multa") \
          .getOrCreate()
spark.sparkContext.setLogLevel("WARN")

startGlobal = time.time()
start = time.time()
multa = spark.read.csv("multas/", sep = ";", header = True,
      ↪ignoreLeadingWhiteSpace = True, ignoreTrailingWhiteSpace = True)

print (time.time()-startGlobal, time.time()-start)
```

```
multa2 = multa.withColumn('DENUNCIANTE',
      ↪regexp_replace('DENUNCIANTE', '^() ', '')) \
         .withColumn('LUGAR', regexp_replace('LUGAR',
      ↪'^(CALLE|VIA|AV|CL|PZ|CTRA)', '')) \
         .withColumn('LUGAR', regexp_replace('LUGAR', '[0-9]+$', '')) \
        .withColumn('HORA', regexp_replace('HORA', '.[0-9]+$','')) \
         .withColumn('LUGAR', regexp_replace('LUGAR', '^(M-30 XC K 19,06
       ↪CR)','M-30 KM 19'))\
         .withColumn('LUGAR', regexp_replace('LUGAR', '^(M-30 CALZADA 1
       ↪KM 19,)','M-30'))\
         .withColumn('MES', regexp_replace('MES', '^()', '')) \
         .groupBy('LUGAR', 'DENUNCIANTE', 'MES',
      ↪'HORA').count().sort(desc('count'))

multa2.show(10)
print(time.time()-startGlobal, time.time()-start)
```

Una vez eliminado los datos que no queremos y que quede limpio, el resultado debería ser el siguiente:

Tabla 4.8. Resultados a consulta 4.6

| LUGAR | DENUNCIANTE | MES | HORA | COUNT |
|---|---|---|---|---|
| M-30 | POLICIA MUNICIPAL | 05 | 16 | 2304 |
| M-30 KM 19 | POLICIA MUNICIPAL | 07 | 10 | 2240 |
| M-30 KM 19 | POLICIA MUNICIPAL | 07 | 11 | 2181 |
| M-30 | POLICIA MUNICIPAL | 05 | 13 | 2138 |
| . | . | . | . | . |
| . | . | . | . | . |
| LEGANITOS | AGENTES DE MOVILIDAD | 12 | 12 | 1733 |

## 4.5 Conclusiones a las consultas

Una vez explicado su funcionamiento, nos centraremos en los resultados de su trabajo, nos hemos centrados en los datos desde el 2015 hasta octubre de 2017 que son los últimos datos publicados en la web a día de hoy. En estos casi tres años, los agentes han impuesto un total de 3.473.727, en su mayoría infracción leve.

En este punto hablaremos de los tiempos que se han medido en una consulta específica A continuación se verá en la tabla 4.9 los tiempos obtenidos para la misma consulta, en este caso: *"callesMesHoraDenun.py"*

Tabla 4.9 Tiempos de consultas "callesMesHoraDenun.py"

| | Entorno pseudo-distribuido | Entorno Cloud | Entorno distribuido |
|---|---|---|---|
| | 138.82 | 79.03 | 90.56 |
| | 136.23 | 58.31 | 92.86 |
| | 138.39 | 55.53 | 89.59 |
| | 137.81 | 56.89 | 87.81 |
| | 132.84 | 63 | 89.44 |
| | 138.60 | 55.53 | 85.60 |
| | 135.15 | 55.07 | 89.85 |
| | 134.89 | 56.23 | 91.69 |
| | 135.71 | 55.07 | 84.91 |
| | 135.81 | 55.85 | 87.71 |
| Media | 136.42 | 59.05 | 89.00 |

## 4.6 Consulta multas Madrid

Hemos explicado anteriormente una consulta de manera exhaustiva**,** pero aparte de esta consulta, se ha hecho una serie de consultas que se han considerado necesarias para comprender una problemática como son las multas de la ciudad de Madrid. Por ello se describen las siguientes consultas.

En esta consulta, se puede ver las calles donde hay más multas de Madrid, donde podremos ver en qué lugar, y hora han sido puestas estas multas y su calificación. Esto nos permite comprobar que es en la carretera de circunvalación M-30 donde se ponen más multas.

Tabla 4.10 callesMesHoraCali.py

```
#Consulta que nos da una respuesta con las 50 calles donde más multas hay
#y que nos las relaciona con su calificacion, el mes cuando ha ocurrido y la
#hora.
```

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
import time

spark = SparkSession.builder \
          .master("local") \
          .appName("multa") \
          .getOrCreate()

spark.sparkContext.setLogLevel("WARN")


startGlobal = time.time()
start = time.time()

multa = spark.read.csv("multas/", sep = ";", header = True,
      ↪ignoreLeadingWhiteSpace = True, ignoreTrailingWhiteSpace = True)

print (time.time()-startGlobal, time.time()-start)

multa2 = multa.withColumn('CALIFICACION',
      ↪regexp_replace('CALIFICACION', '^() ', '')) \
          .withColumn('LUGAR', regexp_replace('LUGAR',
      ↪'^(CALLE|VIA|AV|CL|PZ|CTRA)', '')) \
          .withColumn('LUGAR', regexp_replace('LUGAR', '[0-9]+$', '')) \
          .withColumn('LUGAR', regexp_replace('LUGAR', '^(M-30 XC K 19,06
      ↪CR)','M-30 KM 19'))\
          .withColumn('LUGAR', regexp_replace('LUGAR', '^(M-30 CALZADA 1
↪KM 19,)','M-30'))\
          .withColumn('HORA', regexp_replace('HORA', '.[0-9]+$','')) \
          .withColumn('MES', regexp_replace('MES', '^()', '')) \
          .groupBy('LUGAR', 'CALIFICACION', 'MES',
      ↪'HORA').count().sort(desc('count'))

multa2.show(50)
print(time.time()-startGlobal, time.time()-start)
```

Tabla 4.11 Respuesta a la consulta 4.10

| LUGAR | CALIFICACION | MES | HORA | count |
|---|---|---|---|---|
| M-30 | GRAVE | 05 | 12 | 2304 |
| M-30 KM 19 | GRAVE | 07 | 10 | 2240 |
| M-30 KM 19 | GRAVE | 07 | 11 | 2181 |
| M30 | GRAVE | 05 | 11 | 2138 |
| . | . | . | . | . |
| . | . | . | . | . |
| . | . | . | . | . |
| LEGANITOS | LEVE | 12 | 12 | 1733 |

En la siguiente consulta se verá los agentes responsables de la puesta de poner multas y el la calificación de la infracción, se puede ver que donde más multas hay es en el Servicio de Estacionamiento Regulado.

Tabla 4.12 denunciantеCalificacion.py

```
#Consulta que nos da una respuesta con las 35 calles donde más multas hay #y que nos las relaciona el denunciante y la calificación

from pyspark.sql import SparkSession
from pyspark.sql.functions import *
import time

spark = SparkSession.builder \
        .master("local") \
        .appName("multa") \
        .getOrCreate()
spark.sparkContext.setLogLevel("WARN")


startGlobal = time.time()
start = time.time()
multa = spark.read.csv("multas/", sep = ";", header = True,
    ↪ignoreLeadingWhiteSpace = True, ignoreTrailingWhiteSpace = True)
print (time.time()-startGlobal, time.time()-start)

multa2 = multa.withColumn('CALIFICACION',
    ↪regexp_replace('CALIFICACION', '^() ', '')) \
        .withColumn('DENUNCIANTE', regexp_replace('DENUNCIANTE', '^()',
```

```
↪")) \
        .groupBy('DENUNCIANTE',
    ↪'CALIFICACION').count().sort(desc('count'))

multa2.show(35)
print(time.time()-startGlobal, time.time()-start)
```

Tabla 4.13 Resultado a consulta 4.12

| DENUNCIANTE | CALIFICACION | count |
|---|---|---|
| SER | LEVE | 3305126 |
| POLICIA MUNICIPAL | GRAVE | 1777120 |
| AGENTES DE MOVILIDAD | LEVE | 1464272 |
| POLICIA MUNICIPAL | LEVE | 220172 |
| SER | GRAVE | 192509 |
| AGENTES DE MOVILIDAD | GRAVE | 168601 |
| SACE | GRAVE | 43595 |
| POLICIA MUNICIPAL | MUY GRAVE | 25781 |
| SACE | LEVE | 4326 |
| AGENTES DE MOVILIDAD | MUY GRAVE | 491 |

En esta consulta se ve dónde se ve a quien ha puesto la denuncia, y la hora y lugar donde han sido impuestas, siempre con la suma de estas multas. Se observa que quien más multas pone es el Servicio de Establecimiento Regulado y por la mañana, a partir de las 10, que es cuando empieza el movimiento en la ciudad.

Tabla 4.14 denunciantеHoraCali.py

```
#Consulta que nos da una respuesta con las 35 calles donde nos relaciona al
#denunciante con la hora en la que se puso la infraccion y la calificacion

from pyspark.sql import SparkSession
from pyspark.sql.functions import *
import time

spark = SparkSession.builder \
        .master("local") \
        .appName("multa") \
```

```
            .getOrCreate()
spark.sparkContext.setLogLevel("WARN")

startGlobal = time.time()
start = time.time()

multa = spark.read.csv("multas/", sep = ";", header = True,
        ↪ignoreLeadingWhiteSpace = True, ignoreTrailingWhiteSpace = True)

print (time.time()-startGlobal, time.time()-start)

multa2 = multa.withColumn('CALIFICACION',
        ↪regexp_replace('CALIFICACION', '^() ', '')) \
            .withColumn('DENUNCIANTE', regexp_replace('DENUNCIANTE', '^()',
↪'')) \
            .withColumn('HORA', regexp_replace('HORA', '.[0-9]+$','')) \
            .groupBy('DENUNCIANTE', 'CALIFICACION',
        ↪'HORA').count().sort(desc('count'))

multa2.show(35)
print(time.time()-startGlobal, time.time()-start)
```

Tabla 4.15 Respuesta de la consulta 4.14

| DENUNCIANTE | CALIFICACION | HORA | count |
|---|---|---|---|
| SER | LEVE | 10 | 407108 |
| SER | LEVE | 13 | 403683 |
| SER | LEVE | 11 | 362682 |
| . | . | . | . |
| . | . | . | . |
| . | . | . | . |
| POLICIA MUNICIPAL | GRAVE | 12 | 135927 |
| . | . | . | . |
| AGENTES DE MOVILIDAD | LEVE | 12 | 112574 |

A continuación veremos qué agentes son los responsables de imponer más sanciones en el ayuntamiento de Madrid, como se verá donde más infracciones hay es a la hora de estacionar el vehículo en una zona limitada horariamente al estacionamiento.

Tabla 4.16 Consulta Agentes

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
import time

spark = SparkSession.builder \
                    .master("local") \
                    .appName("multa") \
                    .getOrCreate()
spark.sparkContext.setLogLevel("WARN")

multa = spark.read.csv("multas/", sep = ";", header = True,
                ↪ignoreLeadingWhiteSpace = True, ignoreTrailingWhiteSpace =
                ↪True)

multa2 = multa.withColumn('DENUNCIANTE',
                ↪regexp_replace('DENUNCIANTE', '^() ', '')) \
                    .groupBy('DENUNCIANTE').count().sort(desc('count'))

multa2.show(4)
```

Tabla 4.17 Respuesta consulta Agentes

| DENUNCIANTE | count |
|---|---|
| SER | 3473727 |
| POLICIA MUNICIPAL | 3023073 |
| AGENTES DE MOVILIDAD | 1657272 |
| SACE | 47821 |

# 5

# Configuración Apache Spark

En el siguiente apartado se verá posibles configuraciones de Apache Spark que no son las configuraciones que vienen por defecto. Para ello, Apache Spark [35] posee distintos mecanismos de configuración, los cuales se verán a continuación, centrándose en la gestión de memoria.

En este tipo de entornos que son sistemas distribuidos, hay que valorar posibles fallos, entre ellos están el fallo por cuello de botella, donde un sistema puede bloquearse si el maestro, quien realiza todas las gestiones, no es capaz de asimilar la carga de trabajo. Además la gestión en la memoria puede ser una posible causa de fallo del sistema.

El entorno de Apache Spark se puede configurar de tres maneras:

- Propiedades de Spark. Se puede controlar más parámetros de aplicación y puede establecerse mediante un objeto SparkConf, o a través de propiedades de sistema Java. En este caso nos centraremos en el objeto SparkConf.
- Variables de entorno. Se pueden utilizar para establecer la configuración de cada máquina, como la dirección IP, a través de la escritura de la conf/spark-env.sh en cada nodo.

- Registro puede configurarse a través de log4j.properties y spark-default.conf.template, renombrado a spark-default.conf.

Al usar como lenguaje de entorno Python, se usará el comando para ejecutar spark-submit, que se encuentra en el directorio "/home/doime/apacheSpark/spark/conf", la carga de este entorno puede configurarse dinámicamente, para ello se debe usar el flag --conf para su uso, para conocer qué propiedades se pueden usar en la carga dinámica, realizaremos una consulta --help a spark-submit para conocer la lista de posibles opciones.

Tabla 5.1 Consulta ayuda

```
cd apacheSpark/spark/bin
./spark-submit –help
```

Esta consulta nos da un total de treinta y tres posibles opciones, de ellas las que más nos importan para este proyecto son las siguientes:

- --name nombre_APLICACION. Ponemos el nombre que le daremos al entorno de la aplicación que queramos usar.
- --conf PROPIEDAD = VALOR. Esta propiedad es genérica, ya que "PROPIEDADES" se debe poner el nombre de la propiedad que queramos modificar y en "VALOR" el valor de esta. Estas propiedades están en el fichero SparkConf. Por poner un ejemplo sencillo, podemos cambiar el valor de spark.executor.extraJavaOptions que nos da una cadena de opciones extra de JVM a los procesos ejecutores. Nos podrá ayudar en la recolector de Memoria (GC-Garbage collector), que hablaremos de ello más adelante.
- --py-files PY_FILES. Con esta configuración se puede añadir el fichero .py que queramos ejecutar.
- --driver-memory valor_MEMORIA. Podemos elegir la memoria que queremos que tenga el nodo controlador.
- --executor-memory valor_MEMORIA. Elegir la memoria que queremos que tenga el nodo ejecutor
- --driver-cores NUM. El número de núcleos que tenga el controlador.
- --master. La URL maestra para el nodo.

En el caso de querer hacer uso de esta carga de datos dinámica, nuestro entorno Spark se llame “multas”, que la IP del nodo maestro sea la guardada como maestro en el fichero host con el nombre “maestro” y la consulta sea denuncianteHora.py. Se tendría que realizar la carga dinámica como aparece en la tabla 5.2

Tabla 5.2 Carga dinámica de datos

```
cd apacheSpark/spark/bin/
./spark-submit \
--name “multas” \
--master maestro \
--denuncianteHora.py
```

## 5.1 Gestión de Memoria

En este apartado, se podrá ver cómo afecta al entorno la gestión de la memoria, ya que es uno de los factores más importantes en el rendimiento del sistema.

En Apache Spark, el uso de la memoria es dividida en dos categorías: memoria de ejecución y memoria de almacenamiento. La memoria de ejecución se refiere a la parte del sistema que trabaja con uniones, agregaciones, clases… y la memoria de almacenamiento, se encarga de guardar en caché la propagación interna de datos en el nodo.

Existe una región en el entorno Spark donde conviven ambas memorias, en Apache Spark la denominan “M”, esta región se denomina unificada. Cuando la memoria de ejecución ya ha efectuado su trabajo, es la memoria de almacenamiento la que obtiene el total de la capacidad de almacenamiento. La región unificada tiene una subregión que se encarga de que haya un límite para que la memoria de ejecución no tenga el total de este espacio de memoria, Apache Spark llama a este subconjunto “R”.

Existen dos configuraciones que nos permite ajustar a nuestro sistema estos valores, según consideremos necesario.

- *Spark.memory.fraction* nos permite ajustar el tamaño de la región de memoria unificada o “M”. Por defecto es de 300MB.
- *Spark.memory.storageFraction* se encarga de dar la proporción de la subregión “R” que está en la región unificada. Cuanto menor sea esta subregión menor tamaño

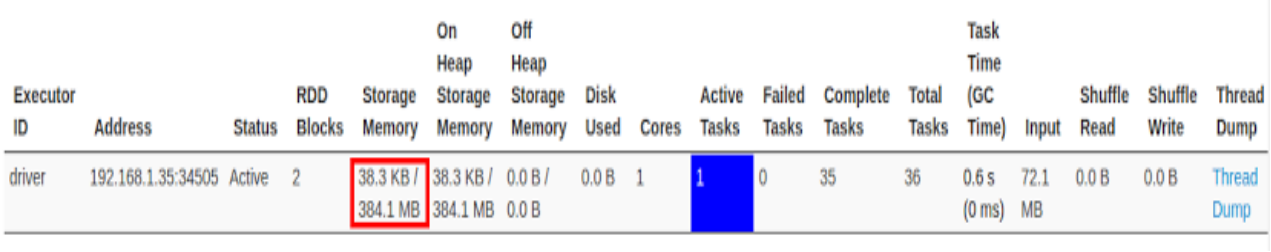
tendrá. Si quisiéramos tener el límite en un 20% del total de memoria unificada, habría que configurarlo con 0.2 como indica la tabla 5.3

Tabla 5.3 Configuración StorageFraction

| |
|---|
| # pondremos el valor en forma de fracción destinada a la región R |
| spark.memory.storageFraction 0.2 |

Para querer usar esta configuración, lo que se debe hacer es saber qué consumo de memoria se necesita. En la página web de la interfaz de usuario del entorno Apache Spark se comprobaría. A continuación se observa un ejemplo del entorno creado en el punto 4, se puede ver en la figura 5.1

Imagen 5.1 Interfaz Usuario Apache Spark



| Executor ID | Address | Status | RDD Blocks | Storage Memory | On Heap Storage Memory | Off Heap Storage Memory | Disk Used | Cores | Active Tasks | Failed Tasks | Complete Tasks | Total Tasks | Task Time (GC Time) | Input | Shuffle Read | Shuffle Write | Thread Dump |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| driver | 192.168.1.35:34505 | Active | 2 | 38.3 KB / 384.1 MB | 38.3 KB / 384.1 MB | 0.0 B / 0.0 B | 0.0 B | 1 | 1 | 0 | 35 | 36 | 0.6 s (0 ms) | 72.1 MB | 0.0 B | 0.0 B | Thread Dump |

En la imagen se observa que los RDDs no son grandes, por lo que almacenar de manera serializada la información no es necesaria, ya que cada RDD ocupa en torno a unos 20 KB y no sería necesario el uso de esta configuración en nuestro caso.

### 5.1.1 GC (Garbage Collection) Recolector de basura

La recolección de basura en JVM puede llegar a ser un problema important2e si los RDD almacenados por el programa son muy grandes. La idea general que se debe tener es que el coste de recolección de basura es proporcional al número de objetos Java, por lo que usar el menor número de ellos implica una mejora en el sistema

Otro de los factores y que hemos hablado anteriormente es el de la interferencia entre la memoria de ejecución y la memoria de almacenamiento.

Si se quiere medir el impacto de la GC, se haría uso de los comandos de ejecución, o en el mismo programa, aquí se hará en el comando de ejecución en nuestra consola Linux, de las siguientes opciones de Java. Para que Java recoja las opciones de ejecución y mostrar las estadísticas sobre la frecuencia de recolección de basura, se debería entrar por línea de comandos y resolver de la siguiente manera la consulta.

Tabla 5.4 Medición impacto CG

```
cd /home/doime/apacheSpark/spark/bin
./spark-submit \
--master local \
--name doime \
--conf "spark.executor.extraJavaOptions=-XX:+PrintGCDetails
↪-XX:+PrintGCTimeStamps" \
callesDenunSER.py
```

Si queremos hacer la configuración de ejecución como son el nombre, definimos qué URL master para nuestro nodo, y añadimos a las opciones de Java que nos imprima los detalles y estadísticas de la recolección de basura en los nodos trabajadores.

El almacenamiento dinámico de JVM se divide en dos regiones, “joven” y “viejo”. La generación vieja es diseñada para que su ciclo de vida sea duradero, sin embargo, la generación joven es todo lo contrario, se encarga de contener a objetos cuya duración es breve.

El objetivo principal de GC es el de limpiar únicamente los RDD antiguos, los almacenados en la zona vieja, y que los objetos jóvenes cuya vida es más corta, tienen el espacio suficiente para estar almacenados. Esto es importante para que no se creen GC completos para recoger objetos temporales, creados durante la ejecución de la tarea.

Para comprobar si funciona correctamente o no la recolección de basura, debemos comprobar cómo está descrito en la consulta 5.3 anteriormente explicada para saber si un GC es invocado varias veces sin hacer la tarea completa.

Si en las estadísticas de GC que se imprimen, la generación vieja, está cerca de llenarse, se tiene que reducir la cantidad de memoria usada para almacenar en caché

disminuyendo spark.memory.fraction; es mejor almacenar en caché menos objetos que ralentizar la ejecución de tareas.

También se puede probar el Recolector de Basura de G1GC con la opción -XX:+UseG1GC introduciéndose como hemos explicado anteriormente en las opciones de ejecución. Esto serviría para mejorar en situaciones en las que GC produce un problema de cuello de botella. Es de gran ayuda para aplicaciones que tienen tamaños de datos grandes en el ejecutor, en nuestro caso, no sería necesario el uso de esta configuración. Para aumentar el tamaño de la región G1, se haría con la opción -XX:G1HeapRegionSize.

## 5.2 Fichero spark-env.sh

*Fichero spark-defaults.conf*

Lo primero que haremos será renombrar el fichero a spark-defaults.conf, de la siguiente manera.

Tabla 5.5 Renombrado fichero spark-defaults.conf

```
cd apacheSpark/spark/conf
mv spark-defaults.conf.templates spark-defaults.conf
```

### 5.2.1 Propiedades de Aplicación

*spark.app.name*

Es el nombre de la aplicación que queramos darle, por defecto no viene ningún nombre, es una propiedad en la cual nos puede ayudar si tuviéramos un clúster muy grande, pero en nuestro trabajo al usar únicamente 2 máquinas, no es muy importante ésta configuración. Además nos sirve para que el nombre aparezca en la interfaz de usuario y datos de registro.

spark.driver.cores

Es el número de núcleos o cores a usar el controlador, cuando está en modo nodo; como nuestra aplicación no es de un tamaño lo suficientemente grande como para que al cambiar el número de núcleos nos afecte a nuestro rendimiento, lo dejamos en su valor por defecto, que es de 1 núcleo.

spark.driver.maxResultSize

Esta configuración sí que puede ayudarnos a mejorar el rendimiento de nuestra aplicación, ya que es el límite de tamaño total de serializado de los resultados de todas las particiones para cada acción de spark como puede ser recoger. Tiene un valor por defecto de 1 gigabyte, siendo el tamaño menor de 1 Megabyte y 0 si no se quiere limitar. Para nuestra aplicación debemos tener en cuenta algo muy importante, que es que nuestro tamaño de resultados no exceda de éste límite que hemos establecido. Por el contrario si un usuario, cree que estableciendo un límite muy alto o directamente no limitar el tamaño, ésto puede causar errores de desmemoria en el controlador. Por lo que lo ideal es ajustar lo máximo posible el resultado obtenido mediante la configuración de *spark.driver.maxResultSize* y *spark.driver.memory*. Dado que el resultado obtenido en nuestra consulta, no es lo suficiente grande para que haya una mejora considerable, lo podríamos dejar igual, pero en general se comprueba que existe una ligera mejora en tiempo de resultados cuando se limita a 1 megabyte. Por lo que modificaremos el fichero *spark-default.conf* de la siguiente manera, añadiendo la siguiente línea al fichero.

Tabla 5.6 Configuración maxResultSize

| | |
|---|---|
| spark.driver.maxResultSize | 1M |

spark.driver.memory

Nos da la cantidad de memoria para el proceso controlador o driver, esto es donde nuestro SparkContext se inicializa, el tamaño por defecto es de 1 gigabyte (1g) En modo cliente, esta configuración debe no establecer a través de la SparkConf directamente en su aplicación. En modo cliente, esta configuración no debe ser establecida directamente en el fichero de configuración ya que el controlador JVM ha empezado en este punto. Debe ser configurado mediante las opciones de *spark-submit* que nos dan la facilidad de una vez que lanzamos el cliente, poder introducir opciones de configuración, como en este caso *spark.driver.memory* mediante el comando --driver--memory MEM donde MEM es el

tamaño que queramos darle, por ejemplo 100M o 2G. Por las características de nuestro clúster y el tamaño del fichero, dejaremos esta configuración por defecto.

spark.executor.memory

Al igual que el proceso controlador, nos da la memoria que necesita el proceso ejecutor. Por defecto es de 1 gigabyte, esta configuración debe cambiarse en el fichero *spark-default.conf*. Al igual que en el caso del driver, lo dejaremos por defecto.

*spark.logConf*

Es básicamente el equivalente a un fichero Log en el que nos dirá qué ha ocurrido cuando sparkContext se ha inicializado, por lo que si ocurre un error, o si hay algún mensaje, esta acción quedará grabado.

*spark.master*

Es la configuración por la cual decidiremos donde queremos que esté el gestor de *clúster*s para conectarse, para la conexión a éste *clúster* maestro, Apache-Spark nos da varias posibles formas de hacerlo.

Tabla 5.7 Asignación de conexiones a *clúster*s

| | |
|---|---|
| local | Ejecuta Spark con un único hilo de trabajo localmente, sin nivel de paralelismo |
| local[K] | Exactamente igual que local, únicamente que la variable "K" tiene el significado de ser el número de hilos de los nodos trabajadores, lo normal es que se establezcan los hilos trabajadores o workers con relación al número de núcleos de nuestras máquinas. |
| local[K,F] | Igual que en el anterior formato, pero añadiendo la variable "F" (maxFailures).Esta variable nos da el número máximo de fallos antes de dejar un trabajo sin hacer, este número debe ser mayor o igual a 1. Si ocurre un fallo en un trabajo específico, no parará el global. El número de intentos será el del maxFailures - 1 |
| local[*] | Usará en vez de un único hilo, tantos hilos como núcleos tenga la máquina |

| | |
|---|---|
| local[*,F] | Hará uso de todos los núcleos de la máquina, pero añadiendo la variable "F", explicada anteriormente en el formato Local [K, F]. |
| spark://HOST:PORT | Conecta al maestro con el *clúster* independiente de Spark, por defecto apache-Spark tiene el puerto 7077. Abajo explicaré cómo inicializar el *clúster* independiente de Spark. |
| spark://HOST1:PORT1:HOST2:PORT2 | Conecta al maestro con el *clúster* independiente de Spark dado, con maestros en espera con Zookeaper. Estos nodos maestros deben estar en la lista de configuración de Zookeaper. El puerto será por defecto 71077, si se desea se puede configurar un puerto distinto. |
| mesos://HOST:PORT | Conectamos el maestro a un clúster de mesos dado. El puerto debe ser el que esté configurado para usar, que es 5050 de forma predeterminada. |
| yarn | Conectamos a un *clúster* Yarn, ésta conexión se hará en modo cliente, para ello debemos modificar la variable HADOOP_CONF_DIR o YARN_CONF_DIR. Este caso será cuando se haga con el entorno distribuido. |

Para inicializar el *clúster* independiente de Apache-Spark, necesitamos acceder al fichero ejecutable de Apache-Spark *start-master.sh* una vez ejecutado, nos saldrá un mensaje en la pantalla de comandos que nos dará el host que tengamos definidos, lo que nos imprimirá será lo siguiente: spark://HOST:PORT. El puerto será el 7707. Si no se quiere usar ningún host diferente, Apache-Spark nos proporcionará uno por defecto, en el que podremos encontrar en el interfaz de usuario, que será la siguiente: spark://localhost:8080 El comando será el siguiente.

Tabla 5.8 Inicialización *clúster* imaster

```
cd /home/peter/apacheSpark/spark/bin
./sbin/start-master.sh
```

Una vez que hemos ejecutado, podremos inicializar de manera parecida los nodos trabajadores y asociarlos al nodo maestro, con el siguiente comando.

Tabla 5.9 Inicialización *clúster* esclavo

```
cd /home/peter/apacheSpark/spark/bin
./sbin/start-slaves.sh <URL-nodo-maestro>
```

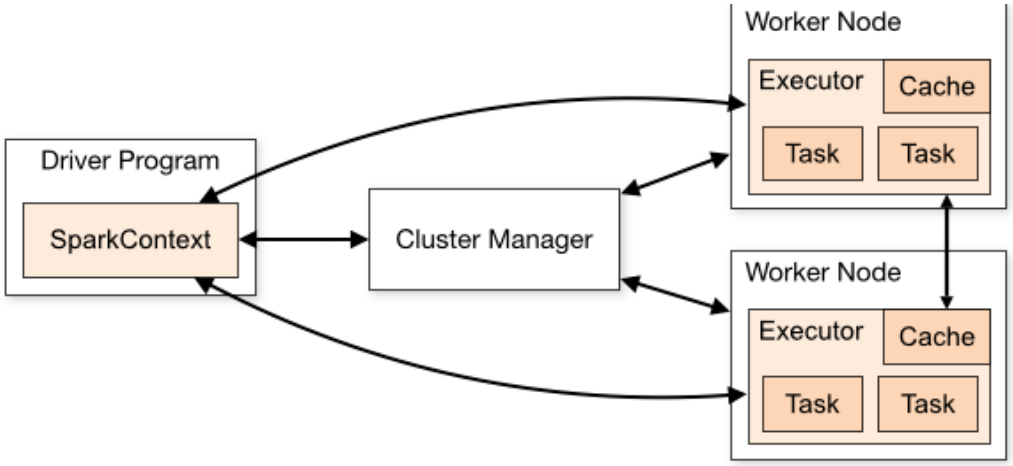
Zookeaper

Este modo lo que nos permite es tener asociados a una misma instancia de Zookeaper varios nodos maestros. Un nodo maestro será el encargado de ser el líder de todos los nodos maestros, este nodo será quien realice la función de lanzar el entorno, mientras los demás nodos maestros se encuentran en estado de latencia a la espera de la muerte del nodo líder, si esto ocurre continuará con el trabajo del antiguo nodo líder. Hay un tiempo de entre 1 o 2 minutos para recuperar el trabajo del nodo caído, este tiempo sólo se verá afectado en los nuevos trabajos, ya que en los trabajos que están corriendo cuando cae el nodo maestro, continuarán hasta ser acabados, la caída no afecta a los trabajos ya inicializados.

Mesos

Es una implementación de *clúster* independiente, lo que hace mesos básicamente es reemplazar el nodo maestro por un nodo mesos, este nodo tomará el control del nodo maestro y elegirá qué trabajos y qué máquinas son las que realicen el trabajo. La diferencia entre usar éste nodo o el original de Apache-Spark es que no tienen la misma prioridad a la hora de calcular los trabajos.Mesos tiene en cuenta de manera diferente la programación de tareas de corta duración, varios marcos pueden coexistir en el mismo clúster sin recurrir a una petición estática de recursos.

Imagen 5.2 Estructura Mesos [35]




Yarn

Para empezar, debemos asegurarnos de que HADOOP_CONF_DIR apunta correctamente al directorio de configuración, que vaya dirigido al cliente, para el clúster de Hadoop. Estas configuraciones son usadas para escribir en HDFS(Hadoop Distributed File System) y conectarse al administrador de recursos de Yarn. La configuración contenida en este directorio se distribuirá al clúster YARN para que todos los contenedores utilizados por la aplicación utilicen la misma configuración.

Usaremos la distribución del modo cliente, en donde el controlador se ejecuta en el proceso del cliente y el maestro de la aplicación únicamente se usa para pedir recursos de Yarn.

*spark.submit.deployMode*

Este modo de implementar, lo que hace al implementar el controlador de Apache-Spark, en modo cliente o clúster, es que al iniciar el programa, se hará de manera local en el cliente o de manera remota en el clúster de uno de los nodos que tenga el clúster.

# 6

# Conclusiones y trabajos futuros

## 6.1 Conclusiones

El objetivo principal de este trabajo era el diseño, despliegue y puesta en marcha de un entorno big data empleando Apache Spark, para ello se ha hecho uso de tres entornos, un pseudo-distribuido, uno distribuido para el que se tuvo que implantar un sistema de ficheros HDFS y un entorno Cloud, Databricks. En el estudio se ha podido ver las distintas mejoras a la hora de configurar nuestros entornos Apache Spark. Para probarlo se ha recurrido a hacer varias consultas con un entorno real, basado en los datos de multas de la ciudad de Madrid.

Lo que pretendía este trabajo es comprobar cómo fluctúan los tiempos entre distintos entornos, de realizar una solución a un problema utilizando el entorno Apache Spark, ver cómo se configuran distintos valores o atributos del entorno en Apache Spark, el tiempo de ejecución mejora. Para ello se ha buscado que los sistemas fuesen escalables con facilidad y eficiente, básicamente que se viera mejora en los tiempos de procesamiento entre distintos entornos.

Tras acabar de diseñar, desplegar y poner en marcha los entornos *big data* usando *Apache Spark*, podemos decir que se han cumplido los objetivos, hemos visto cómo se hace

en varios entornos y se ve lo fácil que es su uso. Además si tuviéramos más nodos, se podría ver lo fácil que es la conexión de más nodos al sistema. Se puede comprobar que estando en un entorno pseudo-distribuido, el tiempo de procesamiento de la petición es el mayor que en los otros dos entornos, y el distribuido es un poco más pesado en tiempos que el entorno *Cloud* como aparece en la tabla4.9.

Aunque se haya realizado el proyecto con escalabilidad y con fácil mantenimiento y la posibilidad de aumentar el número de nodos según terminales se disponga. El hecho de tener la posibilidad de tener todo el sistema en un entorno Cloud, da mucha más facilidad a la hora de lanzar Script y un mantenimiento mucho más fácil de llevar, ya que el entorno es quién nos lo aporta, únicamente se debe pagar el almacenamiento que queramos y el levantamiento de los nodos para la ejecución del script. Por ejemplo en precio por máquina y virtual lanzada en horas en el entorno Databricks, son los siguientes. [19]

Imagen 6.1 Precios por máquina virtual

| CARGA DE TRABAJO | PRECIOS DE DBU, NIVEL ESTÁNDAR | PRECIOS DE DBU, NIVEL PREMIUM |
|---|---|---|
| Ingeniería de datos | €0,17/DBU por hora | €0,296/DBU por hora |
| Análisis de datos | €0,34/DBU por hora | €0,464/DBU por hora |

Uno de las conclusiones que se han podido sacar es la de cómo el recolector de basuras de JVM puede hacer que nuestra aplicación pueda fallar o tener un rendimiento inferior, para optimizar el rendimiento, lo recomendable sería serializar los RDDs de nuestra aplicación o usar en la ejecución de la aplicación la configuración *-XX:+UseG1GC* para que no exista cuello de botella y la aplicación falle.

## 6.2 Trabajos futuros

En este punto se verá qué posibles aplicaciones o trabajos futuros se podrían hacer después de realizar el proyecto y sacar las conclusiones anteriores.

- Para la realización de este proyecto en un entorno real, viendo los costes, lo que sería más coherente por facilidades y costes sería el de realizarlo en entorno Cloud, ya sea Amazon, Google o Microsoft.

- Optimización del rendimiento, se podría profundizar en el recolector de Basuras de Java, o en otras configuraciones para mejorar el rendimiento.
- Hacer uso de Dashboards como representación de datos en la obtención de los resultados. Con esta herramienta, se puede observar mejor la representación de los datos. Como ejemplos de herramientas o softwares de Dashboard tenemos a Tableau o Power BI.
- En los script, se les podría añadir inteligencia artificial. Añadir algoritmos para la optimización de la obtención de datos y conseguir interpretar o predecir posibles actuaciones dependiendo del ámbito a los que estén centrados estos.
- Uno de los trabajos a los que se podría hacer frente en el caso del entorno pseudo-distribuido y distribuido, es la creación de una interfaz para el uso de lanzamiento de scripts como la configuración de estos.

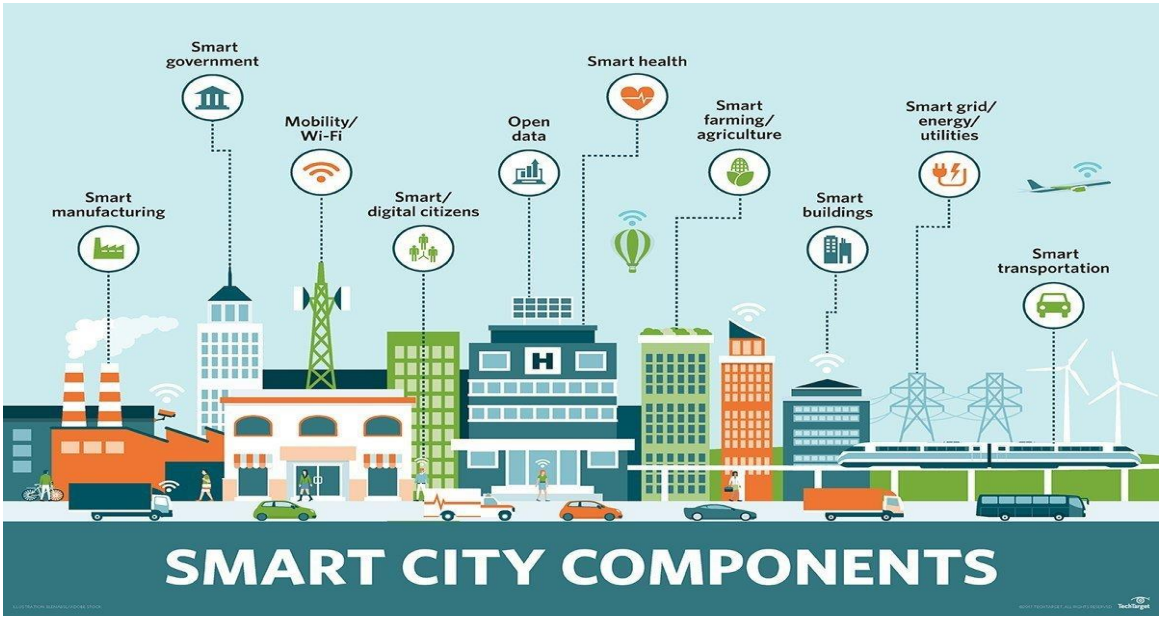
# 7

# Entorno Socioeconómico

Hoy en día, la importancia de analizar grandes volúmenes de datos como modo de lograr una ventaja competitiva sostenible es reconocida como una realidad indiscutible en el mundo empresarial [16]. Por ejemplo en los sectores de la banca y del marketing digital, se tienen en cuenta estos datos acumulados y con algoritmos de machine learning se intenta optimizar y mejorar las ofertas a los clientes. Por ejemplo la estrategia de marketing de Retargeting, se basa en realizar ofertas a clientes con los que la empresa ya ha interactuado alguna vez, con los datos obtenidos de esta relación, se puede mejorar mediante algoritmos futuras ofertas, ya sea centrándose en una hora del día, en unos sitios webs u ofreciendo productos o servicios que esa persona ya ha buscado antes.

La tecnología *big data* también está siendo protagonista en el entorno público, muchas ciudades están empezando a utilizar esta tecnología para su beneficio propio, ya sea para mejorar a la hora de monitorizar el tráfico, contaminación, seguridad, etc. Una *Smart City* se define como un sistema complejo e interconectado que aplica las nuevas tecnologías para gestionar desde el correcto funcionamiento de los sistemas de transporte público y privado, hasta el uso eficiente de los recursos energéticos o hídricos, pasando por los planos de protección civil, o aspectos socioeconómicos, como la vitalidad de los espacios públicos y del tejido comercial, o la comunicación de incidencias a habitantes y visitantes [15].

Imagen 7.1 Smart City [14]



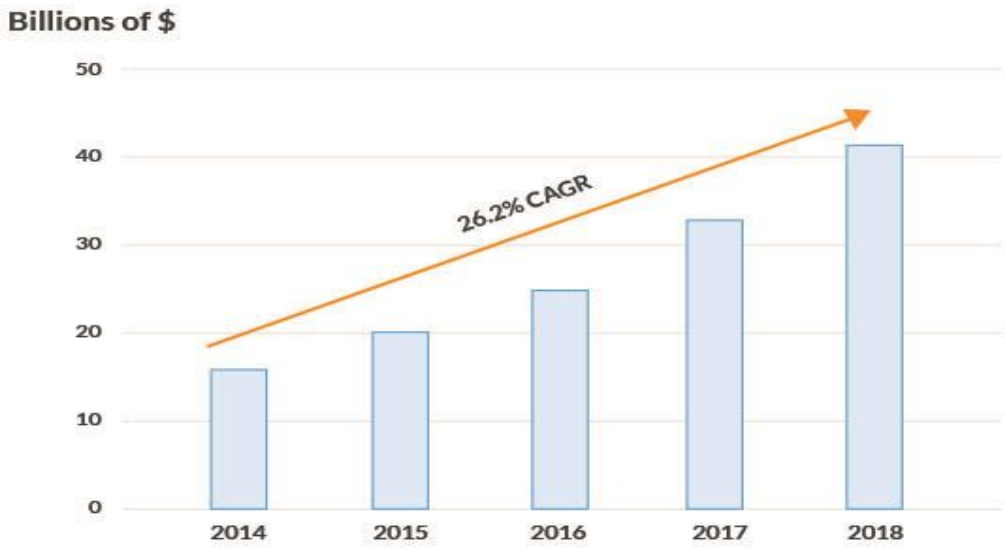

Incluso este tema tiene relevancia en la política europea, se puede ver una lidia entre grandes empresas que tienen un negocio con los datos que generan como puede ser Google, Facebook o Amazon como ejemplos más característicos y conocidos, de esta pugna ha salido la RGPD[10]. Pero también la Unión Europea conoce los beneficios económicos, industriales o de investigación por ejemplo, por ello la Comisión Europea da algunos ejemplos de cómo el tratamiento y análisis de datos puede cambiar la sociedad.[13]

- Se cambiará la industria en general en la Unión Europea ya que habrá una mayor gama de productos y servicios de información.
- Se aumentará la productividad de todos los grupos de la economía.
- Se mejorará la investigación y además se acelerará la innovación.
- Se reducirá costes con servicios más centrados en el cliente.
- Se aumentará la eficiencia en el sector público

En el aspecto económico, podríamos decir que el tratamiento de datos tendrá un peso fuerte y específico en la economía mundial. Además, el mejor y más rápido tratamiento de la información tendrá también un impacto directo en servicios como el transporte, el turismo o la salud, entre otros. En total, se estima que el impacto del *Big data* en la economía europea será de 206.000 millones de euros en 2020, repercutiendo en un

incremento medio del 1,9% del PIB de la Eurozona.[18]. En el siguiente gráfico podemos ver la tendencia de la economía hasta el año de hoy.

Imagen 7.2 Aumento de los ingresos del sector *Big data* [18]




Uno de los temas importantes en la sociedad es la indefensión ante el robo de datos a las empresas, por ello los países están regulando el sector para que se tenga el acceso al mínimo de datos personales para evitar estos robos y sobre todo que las empresas que tienen los datos sean responsables de su almacenamiento. Por poner el mayor ejemplo de robo de datos y tener conciencia del tamaño de estos, el ciberataque más importante de la historia afectó a Yahoo! en 2013 y alcanzó las cuentas de sus 3.000 millones de usuarios [17], para hacernos una idea es casi la mitad de la población mundial.  O por poner un ejemplo de datos sensibles, en septiembre 2017, la agencia de crédito Equifax, que se encarga de recabar datos personales de los consumidores que solicitan un crédito, reveló el pirateo de datos sensibles de más de 147 millones de clientes estadounidenses, canadienses y británicos. [17]

Ante el auge de estas tecnologías como hemos podido ver en este mismo punto, el encuadre de este proyecto podría hacerse tanto en un entorno Cloud como uno doméstico siendo de importancia el almacenamiento de datos y el análisis de problemas y sus soluciones, este hecho puede hacer que la empresa obtenga un valor añadido a la hora de la búsqueda de soluciones en el sector que sea implantado.

# 8

# Planificación y Presupuesto del proyecto

En este apartado hablaremos de la planificación y el presupuesto que tendría el proyecto de llevarse a cabo en la vida real. Primero hablaremos de la planificación del proyecto y de los tiempos que se tardó en realizar, para ello hay que poner un apunte, dado que por motivos de trabajo, las fechas han sido más largas de lo debido.

Por ello la duración del desarrollo de este proyecto se ha alargado más con lo que en vez de cinco meses estipulados, han sido ocho meses**.** Con lo que se empezó en diciembre de 2017 y se acabó en agosto de 2018, para así cumplir los plazos dados por la universidad Carlos III de Madrid.

## 8.1 Fases del Proyecto

En este apartado se describirán las fases en las que se ha dividido el trabajo y el tiempo dedicado a ello. Son:

1. **Entendimiento del problema e identificación de las herramientas necesarias:** El objetivo principal de esta fase es el análisis planteado por el tutor y por el alumno, en el que se debe identificar qué herramientas y tecnología y entornos se usará a lo largo del proyecto.

Tiempo de operación: 20 días.

2. **Estudio de la tecnología a desarrollar:** En este apartado se estudia el funcionamiento de la tecnología a usar. En este caso se estudiará toda la documentación del entorno Apache Spark, Apache Hadoop y se verán videos de la página y pequeños cursos de la tecnología. En este punto, compartido con el diseño del sistema, se encuentra la búsqueda de información y APIs para mejorar la configuración de los entornos.

   Tiempo de operación: 20 días.

3. **Diseño del sistema:** Desarrollo teórico de las posibles soluciones al problema planteado, se analizará la API de Apache Spark,y se hará el análisis de las posibles consultas realizadas en el proyecto. Además en este punto se encuentra la preparación de las máquinas para la implementación del entorno.

   Tiempo de operación: 14 días.

4. **Implementación del entorno pseudo-distribuido:** Preparación del entorno y desarrollo del código de las consultas para el entorno pseudo-distribuido.

   Tiempo de operación: 22 días.

5. **Implementación en entorno Cloud:** En esta fase se analizará el entorno Cloud, en este proceso Databricks, se verá el funcionamiento y se adaptarán las consultas adaptadas al entorno.

   Tiempo de operación: 23 días.

6. **Implementación Sistema de Ficheros HDFS:** Preparación y búsqueda de información del uso de Apache Haddop para el uso del sistema de ficheros en el entorno distribuido.

   Tiempo de operación: 30 días.

7. **Implementación entorno distribuido:** En esta fase se analizará el entorno distribuido, modificando el código para el funcionamiento correcto de la aplicación

   Tiempo de operación: 14 días.

8. **Análisis de resultados:** En esta fase se analizan los resultados obtenidos en los tres entornos.

   Tiempo de operación: 6 días.

9. **Elaboración de la memoria:** En este proceso se hará la redacción de todos los puntos anteriormente nombrados.

   Tiempo de operación: 35 días.

### 8.1.2 Diagrama de Gantt

A continuación mostraremos el diagrama de Gantt del proyecto, donde queda representado cada fase.

Imagen 8.1: Diagrama de Gantt del proyecto


## 8.2 Presupuesto del proyecto

En este apartado evaluaremos el presupuesto del proyecto, tanto en recursos humanos, como en recursos materiales.

### 8.2.1 Medios empleados

- **Medios materiales:** Se refiere a los recursos materiales que se usarán durante el proceso.
  - 1 ordenador portátil Toshiba Satélite-SATELITE-L850-150.
  - 1 ordenador de sobremesa autoconfigurado.
  - 1 memoria flash Toshiba de 32 GB.
  - Conexión banda ancha a internet.
- **Software y licencias:** Se refiere a todos los programas usados durante la ejecución y elaboración del proyecto.
  - 2 Sistemas Operativos Ubuntu 17.

    - Software Rufus.
    - 1 Apache Spark 2.2.1.
    - 1 Apache Hadoop 3.0.
    - Cuenta en Databricks.
- **Recursos Humanos:** Referidos a las personas que han participado en el proyecto.
    - **1 ingeniero junior:** Pedro Javier Martín-Doimeadios Pozo
    - **1 ingeniero senior:** Julio Villena Román.

## 8.2.2 Presupuesto del trabajo

Tras ver los medios que se han utilizado para la elaboración del proyecto, en este apartado se centrará en el análisis de los gastos.

**Recursos Humanos**

En este punto veremos el coste que ha supuesto la realización desde el punto de vista de las personas que han participado en el proyecto. En este caso de las dos personas que han trabajado en el proyecto, para fijar el salario partiendo del mínimo del Convenio Colectivo Nacional de Empresas de Ingeniería y Oficinas de Estudios Técnicos [11]

- Sueldo Graduado junior / hora: 25 euros.
- Sueldo Graduado junior / hora: 14 euros.

Estos sueldos están calculados en base al valor bruto, por lo que se calcularán las horas realizadas en el trabajo y se multiplicarán con el valor del salario por hora de cada ingeniero. Abajo detallaremos el coste por fase y horas en la tabla 9.1 y el coste total por trabajador en la tabla 9.2.

Tabla 8.1 Coste horas de recursos humanos

| FASE DEL PROYECTO | DÍAS | HORAS INGENIERO JUNIOR | HORAS INGENIERO SENIOR |
|---|---|---|---|
| Entendimiento del problema e identificación de las herramientas necesarias. | 20 | 35 | |
| Estudio de la tecnología a desarrollar | 20 | 80 | |

| | | | |
|---|---|---|---|
| Diseño del sistema | 14 | 60 | |
| Implementación del entorno pseudo-distribuido. | 22 | 75 | |
| Implementación del entorno Cloud | 23 | 46 | |
| Implementación de Sistema de ficheros HDFS | 30 | 100 | |
| Implementación del entorno distribuido | 14 | 30 | |
| Análisis de resultados | 6 | 18 | |
| Elaboración de la memoria | 35 | 100 | |
| **TOTAL** | 184 | 544 | 51 |

Tabla 8.2 Coste total de recursos humanos

| **Trabajador** | **Total Trabajador** | **Horas Trabajadas (€)** | **Coste por hora (€)** | **total (€)** |
|---|---|---|---|---|
| Ingeniero Junior | 1 | 544 | 14 | 7616 |
| Ingeniero senior | 1 | 51 | 25 | 1275 |
| | **TOTAL** | | | 8891 |

## Total medios materiales

En este punto veremos el total de medios materiales con los valores de la tasa de depreciación que se ha obtenido de la página de la Agencia Tributaria [12].

Tabla 8.3 Total costes medios materiales

| Material | Cantidad | Precio (€) | Periodo de uso | Tasa de depreciación | Coste total |
|---|---|---|---|---|---|
| Toshiba | 1 | 700 | 10 meses | 25,00% | 52.5 |
| Ordenador sobremesa | 1 | 1200 | 10 meses | 30,00% | 84 |
| Memoria flash | 1 | 15 | 10 meses | 25,00% | 1.125 |

| | | | | | |
|---|---|---|---|---|---|
| Conexión de banda ancha | 1 | 55 | 10 meses | 0% | 550 |
| | | | | **TOTAL** | 1759 |

**Total software y licencias**

En este punto veremos el total de software y licencias con los valores de la tasa de depreciación que se ha obtenido de la página de la Agencia Tributaria [12].

Tabla 8.4: Total coste software y herramientas

| Software | Cantidad | Precio (€) | Periodo de uso | Amortización | Coste total |
|---|---|---|---|---|---|
| Ubuntu 17. | 2 | 0 | 10 meses | 30,00% | 0 |
| Rufus | 2 | 0 | 10 meses | 30,00% | 0 |
| Apache Spark 2.2.1 | 1 | 0 | 10 meses | 30,00% | 0 |
| Apache Hadoop 3.0 | 1 | 0 | 10 meses | 30,00% | 0 |
| Cuenta Databricks | 1 | 0 | 10 meses | 30,00% | 0 |
| | | | | TOTAL | 0 |

**Coste total del proyecto.**

En este apartado veremos el valor total que tendrá el proyecto de llevarse a cabo. Esta cuantía es de 10650,00 € que a continuación detallaremos en la tabla 8.5.

Tabla 8.5 Coste Total del Proyecto

| **Concepto** | **Coste (€)** |
|---|---|
| Recursos humanos | 8891 |
| Medios Materiales | 1759 |
| Software y herramientas | 0 |
| **TOTAL** | 10650,00 |

# 9

# Extended Abstract

Big amounts of data have been growing spectacularly in the last few years. Currently, 2.5 trillions of data are generated every day, occupying 2.7 zettabytes (ZB) of information in the data universe. This amount is expected to raise to quotes unseeing until today estimating to reach 44 ZB a day.

Nowadays the technology Big Data is used in different sectors of society, including the public and private sector. In the private sector we can see the influence of this technology in the agriculture, the health system, the marketing sector or the bank sector among many others. In the public sector we can appreciate the raise of Big Data in security, the traffic system or urban development.

This is why, at present, a new concept has been developed called Smart Cities. In them the technology Big Data affect in terms of cities development or managing public services. At present, cities are a constant source of information either through users, sensors or applications which send information continuously.

This project aims to display how this technology help us having better and faster solutions, giving us the opportunity of foreseeing circumstances that not so long ago where impossible to predict. With this purpose three different environments have been configured and tested their answer in a real setting using the Apache Spark technology. In this

technology the use of Distributed File System like HDFS and the environment Cloud with Databricks had been needed.

## Goals

The main goal in this project is to obtain possible solutions to real problems with different Big Data environments using Apache Spark.

- Analysis and study of the state of the art and the Big Data tools available today, using Apache Spark.
- Design Big Data Systems based in the Apache Spark environment (distributed and pseudo-distributed) and also in the Cloud environment.
- Optimized the Apache Spark environment for better timing.
- Design and implement query scripts to obtain outcomes.
- Apply scripts and fulfilled queries in a real environment.

## State of the art

In this chapter different technologies and tools that are used during the project development like Big Data, Distributed File Systems (HDFS or Cassandra), Apache Spark, Databricks or Google Cloud are described.

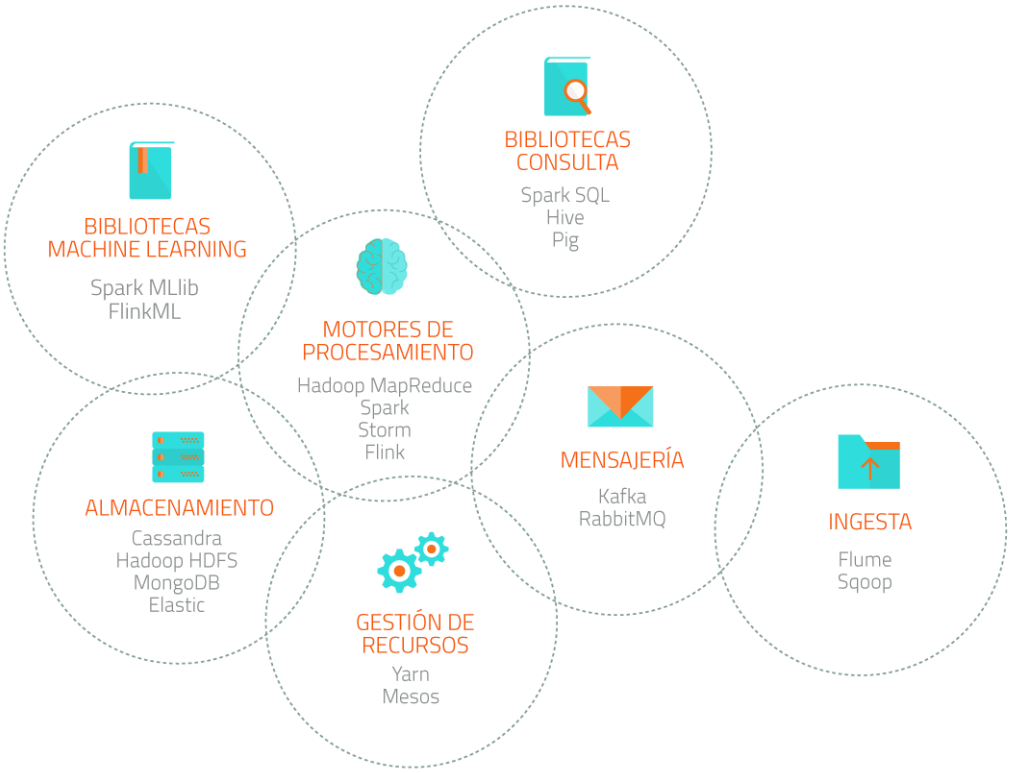
According to the European Union Big Data refers to big amount of data produced fast for a great number of different sources. Data can be produced by humans or machines, like sensors that collect climatic information, satellite imagery, digital videos and images, sells transactions records, GPS signal, etc. [26].

Big Data has 10 features, called the 10 V´s, but at this point we will show the 3 main ones, the original features described to denominate the Big Data concept.

- **Volume w**ithin the Social Media space for example, Volume refers to the amount of data generated through websites, portals and online applications. **Imagen 9.1 Entorno *Big data* [34]**
- **Velocity** with Velocity we refer to the speed with which data are being generated.

- **Variety** in Big Data refers to all the structured and unstructured data that has the possibility of getting generated either by humans or by machines.

Imagen 9.1 Entorno *Big data* [34]



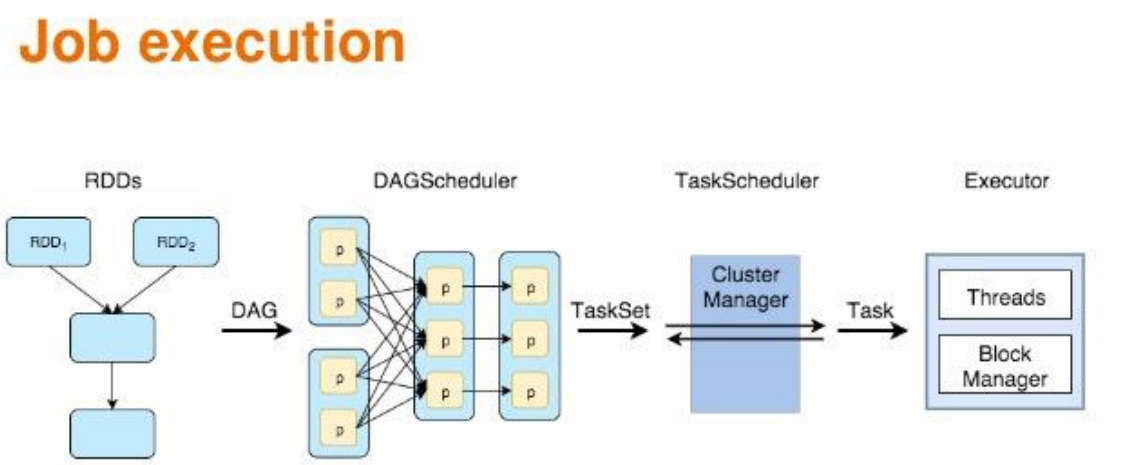

We must say that Big Data environment is based in distributed systems, this systems can be defined like a computer array connected through local network. Furthermore we need the suitable distributed software for the System been visible for the users like a sole entity capable of offer computation solutions. Basically the machines are fully independent in the sense of not shared memory or physical processors.

Hadoop Distributed File Systems are also based in distributed systems. In them hundreds of computers are connected among them and have the purpose of real time data access or streaming and the use of large volumes of data. Inside HDFS we can talk of two different types of nodes, the first one is called Namnode, responsible of giving tasks to the other nodes and store their names. The second one, known as Datanode have data storage in blocks for the use of queries.

A significant part of this project is the use of Apache Spark. Apache Spark is a Distributed Computing System in fast cluster and general use. Apache Spark provides high

standard API in Java, Scala, Python and R, and an optimized engine that accept general execution graphs. Unlike its predecessors Hadoop leave aside the MapReduce transformation to divide it in DAG and RDD where a RDD (Resilient Distributed Dataset) is a collection of elements divided in the cluster nodes that can operate in parallel and a DAG (Direct Acyclic Graph) is a acyclic leaded graph. Both of them work together since each Spark task generate a DAG of work stages. A RDD have two different functions: transformation and action. Below we can appreciate the connection between a RDD and a DAG.

Imagen 9.2 Conexión DAG y RDD [38]




Apache Spar has different libraries associated to the application.

- **Spark SQL.** It is a Spark module for structured data processing.
- **Spark Streaming.** It is an extension of the Spark´s API core, tolerant to failures that may occur in unceasing data streams.
- **MLib.** It is the machine learning library (ML) of Spark. Its goal is to make practical machine learning scalable and easy.
- **GraphX.** It is a processor of distributed graphs.

To finish with the state of the art section we need to emphasize that Databricks is a unified and fast analysis engine for large volumes of data and machine learning [40]. It is an open source project.

## Design

In this section, it has been decided to use three types of environment, a pseudo-distributed environment in which there is only one cluster that will act as a data warehouse and a master and slave cluster. A distributed environment in which there will be two nodes, one that will be the master and from whom the application is launched, and another that will be the master node, which receives orders from the master node. For this environment being accomplish, an installation of an HDFS file system has had to be performed, in which the data to carry out the project will be shared. These data are obtained from a free data portal of the city of Madrid. Once these environments are deployed, we will see how we can work in the cloud with the Databricks application that belongs to Apache Spark.

## Application scenario

At this point we will explain how this type of technology can be used in a real case, such as the fines in Madrid, focusing on where more fines have been given, fines times or who have been responsible for putting these fines and the name of them. In addition, a data query will be carefully explained and what functions have been used to carry out this query. Moreover we will see the execution time in the three environments and queries that can be helpful to understand the case study.

## Apache Spark Configuration

As a starting point, at this point we will see that it is not only setting the stage and its surroundings. In the configuration of the environment, we will focus on different forms of configuration of Apache Spark, such as the garbage collector, where we will see that if we optimize the sizes of the RDDs and make them grouped into large objects and not many small ones, the cleaning time will go down considerably. In addition different configurations of the application as well as the execution will be explained.

## Conclusions and future works

In this section, it is intended to have the main objective of designing, deploying and launching a big data environment and see solutions in a real environment. In it, times of the three developed environments with a same query have been measured, being the cloud environment the one that better performance has had with respect to the other enviroments.

In what refers to future work, the following objectives are taken as a starting point.

- Performance optimization.
- Make use of Dashboards as a representation of data in obtaining results.
- In the generated script, artificial intelligence aggregation or machine learning.
- In the pseudo-distributed and distributed environments. Creation of an interface for the use of script launching and the configuration of these.

## Socieconomic Enviroment

Nowadays, the importance of analyzing large volumes of data as a way to achieve a sustainable competitive advantage is recognized as an indisputable reality in the business world [16]. This technology has brought a change in the point of view of the businesses, since with the data mining, client focused trading is generated, ever since they have detailed data of the aspects of the society and of each individual.

The volume of business has developed in just four years from 15 Billion € to more than 50 Billion €. But these data must be hosted on a server or machine, which is vulnerable to attacks or massive sensitive nature data theft, either for a person or for a state.

## Planning and Budget

At this point we talk about the planning that the project has had over the months, and how objectives and obstacles have been overcome. For this, the following Gantt chart was made, showing the time that has been used to carry out the project.

Imagen 9.3 Diagrama de Gantt


In the budgetary aspect of the project, it must be said that the salary data of the National Collective Agreement of Engineering Companies and Technical Studies Offices [11] have been taken, the expenses have been divided into material expenses, software expenses and licenses and human resources expenses. In the following table, the total cost of the project will be displayed.

Tabla 9.1 Coste Total del proyecto

| **Concepto** | **Coste (€)** |
|---|---:|
| Recursos humanos | 8891 |
| Medios Materiales | 1759 |
| Software y herramientas | 0 |
| **TOTAL** | 10650,00 |

# 10

# Bibliografía


[1] Dircomfidencial, *El Big data dejará en España unos ingresos superiores a los 250 millones de euros en 2019* Feb de 2018. **[En línea] Disponible:***https://www.gmv.com/blog_gmv/language/es/que-aplicaciones-tiene-big-data-en-la-vida-real/*

[2] L. J. Aguilar, *Big data Análisis de grandes volúmenes de datos en organizaciones*.1th ed. México: Alfaomega,2013.

[3] H. Mohanty, P. Bhuyan, and D. Chenthati, *Big data*: *A Primer, vol. 11*. Springer, 2015.

[4] Eduard Salas, Aritz Tutor Antón *El análisis masivo de datos y la ciudad. El caso de Barcelona.* May de 2017.

[5] El Pais. *¿Qué es eso del big data?* Marzo de 2015. **[En línea] Disponible:** http://elpais.com/elpais/2015/03/26/buenavida/1427382655_646798.html

[6] Forbes España. *¿Qué es el big data?* Feb de 2016,**[En línea] Disponible:** http://forbes.es/business/3238/que-es-el-big-data/

[7] *Portal web de la Agencia Española de Protección de Datos. Mayo de 2017*. **[En línea] Disponible:** https://www.agpd.es/.

[8] *Ley Orgánica 15/1999, de 13 de diciembre, de Protección de Datos de Carácter Personal. Dic. de 1999.***[En línea] Disponible:** https://boe.es/boe/dias/1999/12/14/pdfs/A43088-43099.pdf.

[9] *Agencia Española de Protección de Datos. Principales derechos. Mayo de 2017.***[En línea] Disponible:** https://www.aepd.es/reglamento/derechos/index.html

[10] Reglamento (UE) 2016/679 del Parlamento Europeo y del Consejo de 27 de abril de 2016. Mayo de 2016.Obtenido del Boletín Oficial del Estado (BOE) **[En línea] Disponible:** https://www.boe.es/doue/2016/119/L00001-00088.pdf

[11] Disposición 542 del BOE núm.15 de 2017. **[En línea] Disponible:** https://www.boe.es/boe/dias/2017/01/18/pdfs/BOE-A-2017-542.pdf

[12] Agencia Tributaria Española. Tabla de coeficientes de amortización lineal. Jun. de 2017. **[En línea] Disponible:** http://tinyurl.com/zprp38d.

[13] Comisión Europea. 2014b. Making *Big data* work for Europe. Ago. de 2013: http://ec.europa.eu/digital-agenda/en/making-big-data-work-europe

[14]Margaret Rouse. *Smart City.* **[En línea] Disponible:** https://internetofthingsagenda.techtarget.com/definition/smart-city

[15] María Rueda Cruz. *¿Qué son las smart cities?* **[En línea] Disponible:** https://www.bbva.com/es/las-smart-cities/

[16] INNOVASPAIN.Esteban García Cuesta. *El impacto social del Big data* Ene. de 2016. **[En línea] Disponible:**https://www.innovaspain.com/el-impacto-social-del-big-data/

[17] France24 *Los principales casos de robo de datos personales.* May 2018. **[En línea] Disponible:**https://www.france24.com/es/20180515-los-principales-casos-de-robo-de-datos-personales

[18] Cámara de Valencia, *Big data: tendencias de futuro que afectarán a la economía* **[En línea] Disponible:** https://ticnegocios.camaravalencia.com/servicios/tendencias/big-data-tendencias-de-futur o-que-afectan-a-la-economia/

[19] Microsoft Azure Databricks. *Precios de Azure Databricks.* **[En línea] Disponible:** https://azure.microsoft.com/es-es/pricing/details/databricks/

[20] Opendatahandbook, *¿Qué es la ciencia abierta?* Sin fecha. **[En línea] Disponible:** http://opendatahandbook.org/guide/es/what-is-open-data/

[21] Portal de datos abiertos del ayuntamiento de Madrid. Sin fecha **[En línea] Disponible:** https://datos.madrid.es/portal/site/egob/

[22] World Bank Group. *"Open Data & Open API in US and Worldwide"*. Diciembre de 2014 **[En línea] Disponible:** https://www.slideshare.net/DataPortalIndia/open-data-open-api-in-us-and-worldwide

[23] J. Mashey. *"Big data and the Next Wave of InfraStress".*Abril de 1998 **[En línea] Disponible:** http://static.usenix.org/event/usenix99/invited_talks/mashey.pdf.

[24] grupo IGN, *"Historia del Big data".* Mayo de 2017. **[En línea] Disponible:** https://ignsl.es/historia-del-big-data/

[25] Instituto de la Ingeniería del conocimiento. *"¿A qué llamamos big data?"*. Sin fecha **[En línea] Disponible:** http://www.iic.uam.es/big-data/

[26] Comisión Europea. *"Big data"* Abril de 2018. **[En línea] Disponible:** https://ec.europa.eu/digital-single-market/en/big-data

[27] M. Jeevan. *"10 Vs of Big data"*.Mayo de 2018. **[En línea] Disponible:** https://www.edvancer.in/10-vs-big-data/

[28] J. Obero. *"Los datos de ingresos en el Big data Global".* Febreo de 2014. **[En linea]**: http://blog.soydata.net/entradas/los-datos-del-big-data-global/

[29] WordPress.com *"Sistemas distribuidos Global en internet.* Septiembre de 2011 **[En línea] Disponible**: https://internetsistemadistribuidoglobal.wordpress.com/2011/09/05/sistemas-distribuidos-global-en-internet/

[30] J.P Bustos Thames. *"Arquitectura de sistemas distribuidos"* Junio de 2017. **[En línea]Disponible:** https://es.slideshare.net/jpbthames/arquitectura-de-sistemas-distribuidos

[31]Maria Kolodynski . *"P2P".***[En línea]Disponible:** http://proyectoidis.org/p2p/

[32]Apache Foundation. *"Página web de Apache Hadoop"* Junio de 2108. **[En línea] Disponible:** https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html

[33] PowerData. *"¿Qué es el Apache Spark?".* Ocubre de 2014. **[En Línea] Disponible:** https://blog.powerdata.es/el-valor-de-la-gestion-de-datos/bid/397377/qu-es-el-apache-hadoop

[34] {Paradigma. *"Ecosistema Big data"*. Fecha sin especificar. **[En línea] Disponible:** https://www.paradigmadigital.com/ecosistema-big-data/

[35] Apache Foundation. *"Página web de Apache Spark"* .Mayo 2018 **[En línea] Disponible:** https://spark.apache.org

[36]Lorena Etcheverry "*Bases de Datos No Relacionales Instituto de Computación, FING, UdelaR"* – 2017 **[En línea] Disponible:** https://eva.fing.edu.uy/pluginfile.php/144847/mod_resource/content/2/10_inmemoryDBs.pdf

[37] DataArt *"Apache Spark overview"* Noviembre de 2016. **[En línea] Disponible:** https://www.slideshare.net/ittalk/apache-spark-overview-69310245

[38] Ari Handler*"Introducción a Apache Spark - Batch y Streaming".*Noviembre de 2015 **[En línea] Disponible:** https://www.adictosaltrabajo.com/tutoriales/introduccion-a-apache-spark-batch-y-streaming/

[39] Raúl Santamaría*"¿Qué es Cassandra?".* Nobiembre de 2017 **[En línea] Disponible:** https://www.panel.es/blog/por-que-cassandra-no-es-una-base-de-datos-cualquiera/

[40] Databricks *"Página web de databricks".*Actualizado 2018 **[En línea] Disponible:** https://databricks.com

[41] Reynold Xin *"Technichal Preview of Apache Spark 2.0 Now on Databricks"* Mayo de 2016 **[En línea] Disponible:** https://databricks.com/blog/2016/05/11/apache-spark-2-0-technical-preview-easier-faster-and-smarter.html

[42] Linode *"How to Install and Set Up a 3-Node Hadoop Clúster"*.Octubre 2016 **[En línea] Disponible:** https://linode.com/docs/databases/hadoop/how-to-install-and-set-up-hadoop-*clúster*/

[43] Penulis: T. Tavernelli. *"Bases de Datos NoSQL Caso de Estudio: Apahe Cassandra "*Mayo de 2018. **[En Línea] Disponible:** https://kulslide.com/download/bases-de-datos-nosql-caso-de-estudio-apache-cassandra-_5a05b1ced64ab24a7810fef5_pdf

[44] Jeff Kelly *"Executive Summary: Dig Data Vendor Revenue and Market Forecast, 2011-2026"* Marzo de 2015 **[En línea] Disponible:** https://wikibon.com/executive-summary-big-data-vendor-revenue-and-market-forecast-2011-2026/

[46] Página principal Google Cloud. Cloud DataProc. **[En línea] Disponible:** https://cloud.google.com/dataproc/

# 11

# Anexos

Consultas realizadas y su correspondiente salida del programa.

Consulta 10.3 callesDenunHoraCaliMes.py

```
#Consulta que nos da una respuesta con las 35 calles donde más se multa
#por horas junto a los denunciantes, su calificación y el mes.

from pyspark.sql import SparkSession
from pyspark.sql.functions import *
import time

spark = SparkSession.builder \
        .master("local") \
        .appName("multa") \
        .getOrCreate()
spark.sparkContext.setLogLevel("WARN")

startGlobal = time.time()
start = time.time()
multa = spark.read.csv("multas/", sep = ";", header = True,
    ↪ignoreLeadingWhiteSpace = True, ignoreTrailingWhiteSpace = True)

print (time.time()-startGlobal, time.time()-start)
```

```
multa2 = multa.withColumn('DENUNCIANTE',
      ↪regexp_replace('DENUNCIANTE', '^() ', '')) \
          .withColumn('LUGAR', regexp_replace('LUGAR',
      ↪'^(CALLE|VIA|AV|CL|PZ|CTRA)', '')) \
          .withColumn('LUGAR', regexp_replace('LUGAR', '[0-9]+$', '')) \
          .withColumn('LUGAR', regexp_replace('LUGAR', '^(M-30 XC K 19,06
       ↪CR)','M-30 KM 19'))\
          .withColumn('LUGAR', regexp_replace('LUGAR', '^(M-30 CALZADA 1
↪KM 19,)','M-30'))\
          .withColumn('HORA', regexp_replace('HORA', '.[0-9]+$','')) \
         .withColumn('CALIFICACION', regexp_replace('CALIFICACION',
       ↪'^()','')) \
          .withColumn('MES', regexp_replace('MES', '^()', '')) \
          .groupBy('LUGAR', 'DENUNCIANTE','CALIFICACION', 'MES',
↪'HORA').count().sort(desc('count'))

multa2.show(10)
print(time.time()-startGlobal, time.time()-start)
```

Tabla 10.3 Respuesta de la consulta 10.3

| LUGAR | DENUNCIANTE | CALIFICACION | MES | HORA | count |
|---|---|---|---|---|---|
| M-30 | SER | LEVE | 05 | 12 | 2304 |
| M-30 KM 19 | SER | LEVE | 07 | 10 | 2240 |
| M-30 KM 19 | SER | LEVE | 07 | 11 | 2181 |
| M-30 | . | . | 05 | 11 | 2138 |
| . | . | . | . | . | . |
| . | . | . | . | . | . |
| . | . | . | . | . | . |
| LEGANITOS | AGENTES DE MOVILIDAD | LEVE | 12 | 12 | 1733 |

Consulta 10.5 denuncianteHora.py

```
#Consulta que nos da una respuesta con las horas donde más se multa y #quien ha sido el denunciante
```

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
import time

spark = SparkSession.builder \
        .master("local") \
        .appName("multa") \
        .getOrCreate()
spark.sparkContext.setLogLevel("WARN")


startGlobal = time.time()
start = time.time()

multa = spark.read.csv("multas/", sep = ";", header = True,
    ↪ignoreLeadingWhiteSpace = True, ignoreTrailingWhiteSpace = True)
print (time.time()-startGlobal, time.time()-start)

multa2 = multa.withColumn('HORA', regexp_replace('HORA', '.[0-9]+$', '')) \
        .withColumn('DENUNCIANTE', regexp_replace('DENUNCIANTE', '^()',
    ↪'')) \
        .groupBy('DENUNCIANTE', 'HORA').count().sort(desc('count'))

multa2.show(35)
print(time.time()-startGlobal, time.time()-start)
```

Tabla 10.5 Respuesta de la consulta 10.5

| DENUNCIANTE | HORA | count |
|---|---|---|
| SER | 10 | 426374 |
| SER | 13 | 423887 |
| SER | 11 | 382100 |
| . | . | . |
| . | . | . |
| . | . | . |
| POLICIA MUNICIPAL | 12 | 155570 |

## Códigos de gráficas

Para la realización de las tablas para la explicación del caso práctico explicado en el punto 4.4. Se ha realizado, obteniendo los datos de las consultas abajo descritas. Para ello, una vez realizada la consulta en la que obtendremos los datos, abriremos un terminal, en el que crearemos un notebook de anaconda, con el que haremos el código de la figura. Hemos considerado que al hacer uso de *Apache-Spark* en python, la mejor opción era crear un jupyter notebook. Para ello es necesario tener instalado Anaconda y Python, no entraremos en detalle de la instalación de Anaconda, ni el arranque de Jupyter notebook ya que es una herramienta de trabajo más que un despliegue necesario para el objetivo del proyecto.

Consulta 10.6 Código figura 4.1

```
castellana = 98944
claudio_coello = 26779
delicias = 23973
velazquez = 22401
lagasca = 21758
orense = 20294
menendez_pelayo = 20191
santa_engracia = 19935
embajadores = 19814
ayala = 18844

plt.figure()

calles = ("Ayala", "Embajadores","Santa Engracia", "Menendez pelayo",
        ↪"Orense", "Lagasca", "Velazquez", "Delicias", "Claudio Coello",
        ↪"Castellana")

posicion_y = np.arange(len(calles))
unidades = (ayala, embajadores, santa_engracia, menendez_pelayo, orense,
        ↪lagasca, velazquez, delicias, claudio_coello,
        ↪castellana)

plt.barh(posicion_y, unidades, align = "edge")
plt.yticks(posicion_y, calles)
plt.xlabel('Número de multas')
plt.title("Multas SER en Madrid.")
```

Consulta 10.7 Código figura 4.2

```
ser = 3473727
policia = 2023073
agente_movilidad = 1657272
sace = 47921

agentes = ("SACE","Agentes de Movilidad", "Policia Municipal","SER")
posicion_y = np.arange(len(agentes))
total_multas = (sace, agente_movilidad,policia, ser)

plt.barh(posicion_y, total_multas, align = "edge")
plt.yticks(posicion_y, agentes)
plt.xlabel('Número de multas')
plt.title("Total de multas agentes en Madrid.")
```

Consulta 10.8 Código consulta SER

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
import time

spark = SparkSession.builder \
                    .master("local") \
                    .appName("multa") \
                    .getOrCreate()
spark.sparkContext.setLogLevel("WARN")


multa = spark.read.csv("multas/", sep = ";", header = True,
                ↪ignoreLeadingWhiteSpace = True, ignoreTrailingWhiteSpace =
                ↪True)

multa2 = multa.withColumn('DENUNCIANTE',
                ↪regexp_replace('DENUNCIANTE', '^() ', '')) \
                    .withColumn('LUGAR', regexp_replace('LUGAR',
                ↪'^(CALLE|VIA|AV|CL|PZ|CTRA|PO|PASEO)', '')) \
                    .withColumn('LUGAR', regexp_replace('LUGAR', '[0-9]+$', '')) \
                    .withColumn('LUGAR', regexp_replace('LUGAR', '^(M-30 XC K
```

```
            ↪19,06 CR)','M-30 KM 19'))\
                .withColumn('LUGAR', regexp_replace('LUGAR', '^(M-30
            ↪CALZADA 1 KM 19,)','M-30'))\
                .groupBy('LUGAR', 'DENUNCIANTE').count().sort(desc('count'))

multa2.filter(multa2.DENUNCIANTE == 'SER').show(10)
```

Tabla 10.6 Respuesta consulta SER

| LUGAR | DENUNCIANTE | count |
|---|---|---|
| CASTELLANA | SER | 98944 |
| CLAUDIO COELLO | SER | 26779 |
| DELICIAS | SER | 23973 |
| VELAZQUEZ | SER | 22401 |
| LAGASCA | SER | 21758 |
| ORENSE | SER | 20294 |
| MENENDEZ PELAYO | SER | 20191 |
| SANTA ENGRACIA | SER | 19935 |
| EMBAJADORES | SER | 19814 |
| AYALA | SER | 18844 |