UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Escuela de Ingeniería Informática

Diseño e implementación de un Clúster Spark, y análisis de datos de tráfico mediante uso de la librería SparkSQL

TRABAJO FIN DE GRADO 18/19 – INGENIERÍA INFORMÁTICA

Memoria final

Descripción del proyecto
Seguimiento y desarrollo

Tutores

Javier Jesús Sánchez Medina
Antonio Andrés Ocón Carreras

Alejandro López Santos
ALEJANDRO@ALSNET.ES | ALEJANDRO.LOPEZ104@ALU.ULPGC.ES

Índice

1. Introducción 4

1.1 Estado actual 6

1.2 Objetivos del proyecto 7

1.2.1 Objetivos 7

1.2.2 Justificación de las competencias específicas cubiertas 7

1.2.3 Aportaciones 8

2. Desarrollo 10

2.1 Introducción 10

2.2 Metodología 11

2.3 Tecnologías 11

2.3.1 Apache Hadoop 11

2.3.2 Sistema de ficheros HDFS 12

2.3.3 Apache Hadoop Yarn 12

2.3.4 Spark 13

2.3.5 SparkSQL 14

2.3.6 Ruby on Rails 15

2.3.7 R 16

2.3.8 Git 16

2.3.9 Scala 16

2.3.10 SBT 16

2.4 Herramientas 16

2.4.1 Ruby Mine 16

2.4.2 IntelliJ 16

2.4.3 R Studio 17

2.4.4 Mozilla Firefox y Google Chrome 17

2.4.5 Github Desktop 17

2.5 Trabajo realizado 17

2.5.2 Configuración Hadoop y Spark standalone 17

2.5.2.A Instalación Linux Fedora 28 17

2.5.2.B Instalación Java 8 18

2.5.2.C Instalación Hadoop 3.1.0 18

2.5.2.D Instalación Spark 2.3.0 20

2.5.2.E Instalación Jetbrains Toolbox App 20
2.5.2.F Instalación de SBT para Scala 21
2.5.2.G Instalación Ruby on Rails 21
2.5.3 Configuración Hadoop y Spark en clúster 21
2.5.3.A Instalación Linux, Ubuntu 17 21
2.5.3.B Configuración del clúster a nivel de red 23
2.5.3.C Instalación Java 8 25
2.5.3.D Instalación Hadoop 3.1.0 26
2.5.3.E Configuración Yarn 28
2.5.3.F Instalación Spark 2.3.0 30
2.5.3.G Instalación Ruby on Rails 33
2.5.4 Homogeneización del dataset 34
2.5.4.A Obtención de los dataset 34
2.5.4.B Tratar dataset de tráfico 34
2.5.4.C Tratar *dataset* de ubicación de sensores 36
2.5.5 Programación de aplicación SparkSQL 37
2.5.5.A Creación proyecto SBT para Scala 37
2.5.4.B Importar librerías Spark Core y Spark SQL 38
2.5.4.C Código de programación 39
2.5.4.D Comando para ejecutar el programa standalone 41
2.5.4.E Comando para ejecutar el programa sobre clúster 42
2.5.5 Desarrollo aplicación web 43
2.5.5.A Configuración del Ruby Mine 43
2.5.5.B Diagrama de clases de la aplicación 43
2.5.5.C Comandos básicos de Ruby on Rails 43
2.5.5.D Código de programación 45
2.5.5.E Vista previa de la aplicación 46
2.5.6 Informe Speed-Up 52
2.5.6.A Introducción 52
2.5.6.B Preparación de los datos 52
2.5.6.C Fórmula Speed-Up 54
2.5.6.D Visualización de los datos 55
2.5.6.E Análisis de los datos 56
2.5.6.F Conclusión Speed-U 56
3. Conclusiones 57
3.1 Conclusión 57

4. Fuentes de información 59

4.1 Definiciones 59

4.2 Documentación.....60

Índice de figuras

1.- Ilustración 1 – Secuencia de ejecución de la aplicación 10

2.- Ilustración 2 – Arquitectura tecnología Big Data Hadoop 12

3.- Ilustración 3 - Arquitectura Apache Hadoop Yarn 13

4.- Ilustración 4 - Arquitectura cluster de Spark 13

5.- Ilustración 5 - Librerías principales que sustentan Spark 14

6.- Ilustración 6 - Arquitectura de SparkSQL 15

7.- Ilustración 7 - Esquema de red del clúster 23

8.- Ilustración 14 – Diagrama UML de clases de la aplicación Ruby on Rails 43

9.- Ilustración 21 – Gráfica Speed-Up 55

Resumen

Desarrollo de aplicación web para la representación de los datos se ha hecho en forma de mapa de calor utilizando la librería de Google Maps en JavaScript, con soporte web utilizando el framework Ruby on Rails. Además, se hará un análisis de Speedup utilizando diferentes configuraciones de clúster, mediante un script en R.

Overview

It includes the development of a web application for the representation of a dynamic heat map of speeds, by using the Google Maps JavaScript library, and the Ruby on Rails framework for the web development. Additionally, a Speed-up analysis has been carried out, by using a R script, comparing different cluster configurations.

1. Introducción

1.1 Estado actual

En los últimos años, internet ha crecido exponencialmente año tras año, en forma de tráfico en la red y datos almacenados disponibles para consultar. Surgen nuevos conceptos y tecnologías, concretamente la que se va a tratar en este proyecto, *Big Data*.

Antes que nada, es necesario definir *Big Data* para entrar en contexto [1]. Es la tecnología capaz de manejar cantidades ingentes de datos, ya sean o no estructurados dichos datos. También se puede definir con más detalles con las cinco V del *Big Data*:

- Volumen: Aquí entra la parte de cantidades ingentes de datos, hablamos de cifras que superan los gigabytes.
- Velocidad: Característica fundamental de la tecnología, ya que es la parte diferenciadora con respecto a los métodos tradicionales, el proceso de tratar los volúmenes en un periodo de tiempo notablemente inferior.
- Variedad: Los tipos de datos pueden ser una variedad de formatos, proveniente de bases de datos, ficheros, streaming e incluso pueden estar estructurados, semiestructurados o no estructurados.
  Valor: Todo esto tiene un propósito, y es el aporte de beneficios, ya sea de tipo económico o innovador.
- Veracidad: Los datos obtenidos como resultado tienen que tener cierta calidad para que cumplan el propósito.

Según *Wikibon*[1], el ingreso de dinero en el mercado de *Big Data* incrementará de 42 mil millones de dólares americanos a 103 mil millones de dólares americanos en 2027. [11]

Esto hace que el 53% de empresas en 2017 hayan adoptado el modelo *Big Data*, frente al 17% en 2015, y cada vez más se irán uniendo a este modelo, datos recogidos de las empresas entrevistadas por la revista *Forbes*[2]. [12]

Estos datos que detallan la evolución del *Big Data* forman una parte de la motivación para realizar este proyecto. No obstante, la principal causa viene dada por

[1] *Wikibon* es una comunidad de profesionales y consultores en tecnología y sistemas empresariales que utiliza el intercambio de código abierto de conocimiento de asesoramiento gratuito. [14]
[2] *Forbes* es una revista de impacto especializada en el mundo de los negocios y las finanzas.

mi pasión y curiosidad de querer estudiar e investigar lo que no conozco relativo al campo de la ingeniería informática.

1.2 Objetivos del proyecto

1.2.1 Objetivos

Diseñar y evaluar el funcionamiento y rendimiento de esta tecnología para un uso posterior en producción. Así como la creación de la documentación detallada del proceso total.

1.2.2 Justificación de las competencias específicas cubiertas

Hemos seleccionado cuatro competencias de tecnologías de la información, TI01, TI02, TI03 y TI04.

TI01: *Capacidad para comprender el entorno de una organización y sus necesidades en el ámbito de las tecnologías de la información y las comunicaciones.*

Adaptación a las necesidades del *CICEI*[3] reutilizando el multicomputador (MIMD) que ellos disponen para el desarrollo del TFG.0

TI02: *Capacidad para seleccionar, diseñar, desplegar, integrar, evaluar, construir, gestionar, explotar y mantener las tecnologías de hardware, software y redes, dentro de los parámetros de coste y calidad adecuados.*

Despliegue y configuración de un clúster en un multicomputador (MIMD), además de la instalación del software necesario para ejecutar las aplicaciones sobre el clúster.

TI04: *Capacidad para seleccionar, diseñar, desplegar, integrar y gestionar redes e infraestructuras de comunicaciones en una organización*.

Configuración de red aislada de comunicación de los nodos que componen el clúster.

Una vez realizado el trabajo y durante el desarrollo de este apartado, se ha detectado que la competencia TI03 no ha sido relevante y sí lo han sido otras que exponemos a continuación: CII05, CII08, CII11, TI06 y TI07.

CII05: *Conocimiento, administración y mantenimiento sistemas, servicios y aplicaciones informáticas.*

[3] Centro de innovación para la Sociedad de la Información de la Universidad de Las Palmas de Gran Canaria.

Uso de entornos de desarrollo diversos, aplicaciones de ofimática y administración y del clúster.

CII08: *Capacidad para analizar, diseñar, construir y mantener aplicaciones de forma robusta, segura y eficiente, eligiendo el paradigma y los lenguajes de programación más adecuados.*

Utilización de varios lenguajes de programación en este proyecto, se han elegido por conveniencia y utilidad según el tipo de algoritmo a desarrollar, de cara a la eficiencia que proporciona otro lenguaje para la misma tarea.

CII11: *Conocimiento y aplicación de las características, funcionalidades y estructura de los Sistemas Distribuidos, las Redes de Computadores e Internet y diseñar e implementar aplicaciones basadas en ellas.*

Configuración del clúster, uso de tecnología *Big Data* instalada en el mismo, además de una aplicación preparada para ello.

TI0g3 *Capacidad de concebir sistemas, aplicaciones y servicios basados en tecnologías de red, incluyendo Internet, web, comercio electrónico, multimedia, servicios interactivos y computación móvil.*

Desarrollo de una aplicación web para la visualización de datos del resultado del programa *Big Data*.

TI07: *Capacidad para comprender, aplicar y gestionar la garantía y seguridad de los sistemas informáticos.*

Aislamiento de la interfaz de red del nodo maestro hacia el exterior por reglas iptables[4].

1.2.3 Aportaciones

Debido al crecimiento de *Big Data*, surgen nuevas tecnologías, esto implica que la comunidad de desarrolladores de dichas tecnologías sea pequeña. Por tanto, con mi proyecto quiero aportar más documentación sobre *Hadoop*, *Spark* y *SparkSQL*, así quien quiera iniciarse, pueda hacerlo con unos pasos iniciales de forma clara y directa. Además de ver un ejemplo de aplicación, puede incluso ser mejorada si fuera necesario.

Con este proyecto se pretende que la futura persona que quiera iniciarse con estas tecnologías pueda hacerlo de forma ágil, ya que se proporciona la creación de un clúster en un entorno real, ya que estos pasos pueden ser liosos por experiencia propia, tener esto documentado ayudará bastante.

[4] Programa de utilidad que permite a un administrador de sistemas configurar las tablas provistas por el firewall del núcleo del sistema operativo Linux.

El manejo de datos mediante consultas *SQL* hace que el proceso de desarrollo de la aplicación *Big Data* que se esté desarrollando, sea de manera cómoda y sencilla, se documenta como utilizar *SparkSQL* en un entorno de pruebas *standalone* y en un entorno clúster real, además, de la configuración de dichos entornos de desarrollo, con la utilización de un lenguaje actualmente importante en el mundo de la Ciencia de Datos, *Scala*.

Otra parte que destacar en las aportaciones, es la implementación de la aplicación *Big Data* en un entorno web. Utilizar los datos generados por la aplicación y poderlos usar en cualquier entorno que nos pueda favorecer, en este proyecto, para el mapa de calor utilizando *Google Maps JavaScript*, librería de fácil uso en entornos web. Además, con respecto a la aplicación web, se puede incluso mejorar para que sea capaz de tratar más *datasets* de tráfico e incluso de distintas ciudades, así poder implementar el análisis de tráfico y movilidad urbana en las *smart cities*.

Cabe destacar que *SparkSQL* será el standard de manejo de datos de *Apache Spark* en un futuro, esto se debe a que mejora en todos los casos el rendimiento frente al uso de RDD[5], ya en estado *deprecated*[6], por lo que resulta importante esta aportación dentro del mundo de estas tecnologías.

[5] Resilient Distributed Dataset, formato de dataset utilizado por librerías de Apache Spark, actualmente se pretende dejar de mantener para utilizar como fuente principal de datos, SparkSQL.
[6] Término que se utiliza en el mundo de la programación cuando se quiere indicar que esa librería, método de la librería, o trozo de código, está en desuso por algún otro mejorado.

2. Desarrollo

2.1 Introducción

Vamos a desarrollar una aplicación web capaz de representar en forma de mapa de calor, usando la librería de *Google Maps* en *JavaScript*, las velocidades medias de los vehículos que circulan en un instante por la ciudad de Madrid. Rango de fechas especificados por una fecha inicio y fin configurada previamente en la aplicación web.

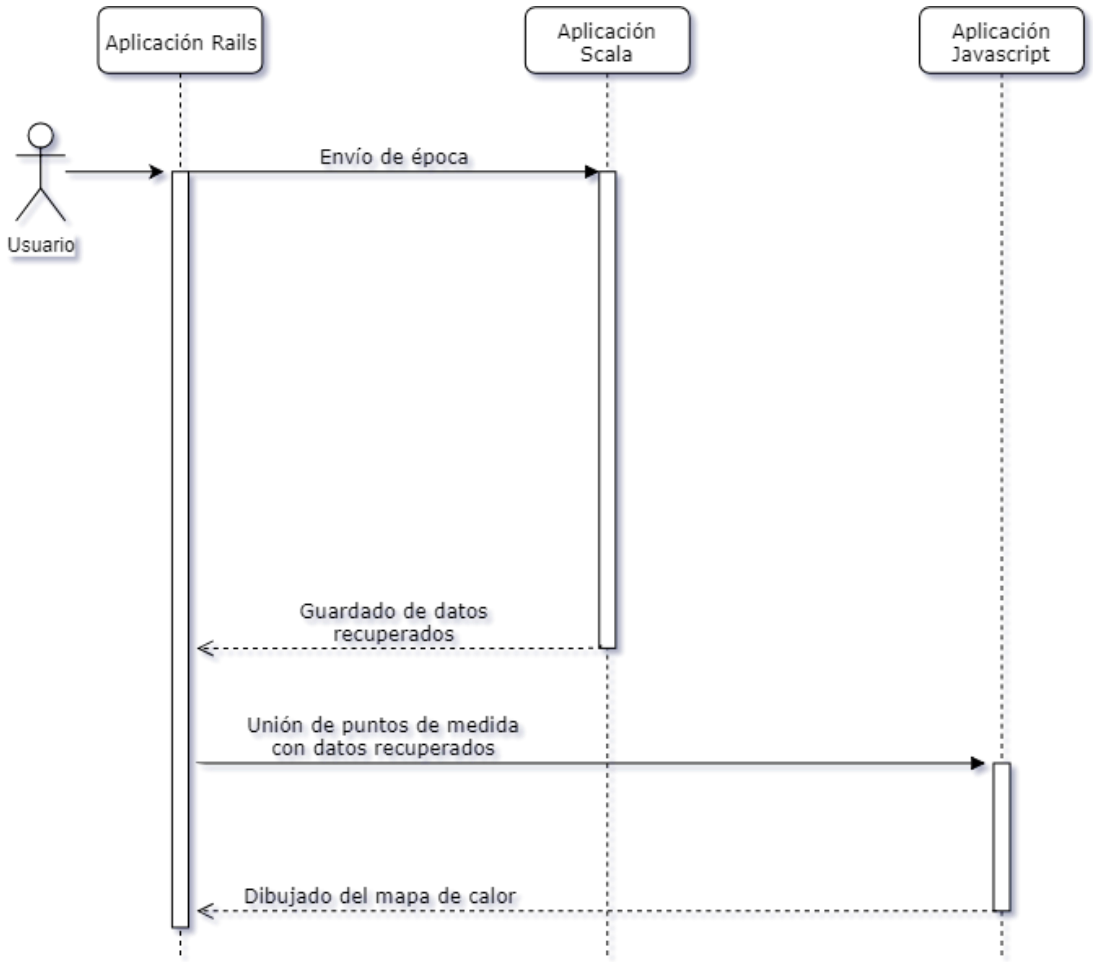

Ilustración 1 - Secuencia de ejecución de la aplicación. Fuente: elaboración propia.

2.2 Metodología

En la primera toma de contacto con la tecnología se ha realizado un curso en la plataforma *e-Learning Udemy* [7] sobre *Big Data Hadoop* y *Spar*k [13], 13 horas y media de contenido audiovisual y contenido explicativo sobre *Hadoop*, *Yarn* y *Spark*. Esto ha permitido una entrada con buen pie ya que se aprende conocimientos para configurar un clúster con máquinas virtuales usando *Virtualbox*[8], además también se ha aprendido a configurar un sistema, como por ejemplo un portátil, para programar la aplicación *Big Data*, y probarla como si fuera un clúster con un solo nodo, para así poder hacer pruebas necesarias antes de lanzarla como tarea al clúster.

Una vez adquiridos los conocimientos básicos, el siguiente paso es aprender más sobre la tecnología *SparkSQL,* y manejar el lenguaje de programación *Scala.* Para ambos casos se ha utilizado la documentación oficial ya que proporciona ejemplos prácticos.

Por tanto, las metodologías aplicadas en este proyecto se han basado en una primera documentación de las tecnologías a utilizar, ya que, en el grado o al menos en la mención cursada, tecnologías de la información, no se estudia sobre *Big Data*, aunque sí sobre clústeres. Y finalmente después de dicha documentación, se ha procedido a programar la aplicación en base a unos objetivos marcados previamente, para así mediante un proceso por objetivos lograr finalizar la aplicación.

2.3 Tecnologías

2.3.1 Apache Hadoop

Pilar de la tecnología *Big Data* que permite el procesamiento distribuido de conjunto de datos repartidos entre los nodos del clúster utilizando modelos de programación simples. Escalable desde un simple nodo a varios miles de nodos, cada nodo con su propio cómputo local y almacenamiento. [15]

Se puede observar en la siguiente ilustración cómo está formado *Hadoop*, es el conjunto del sistema de ficheros distribuidos, motor de gestión del clúster *Yarn* y otras librerías para el uso de dicha tecnología.

[7] Plataforma online para el aprendizaje a base de videos bajo demanda, con documentación y ejercicios prácticos.
[8] Plataforma software de virtualización para arquitecturas x86/amd64. Actualmente desarrollado por Oracle.

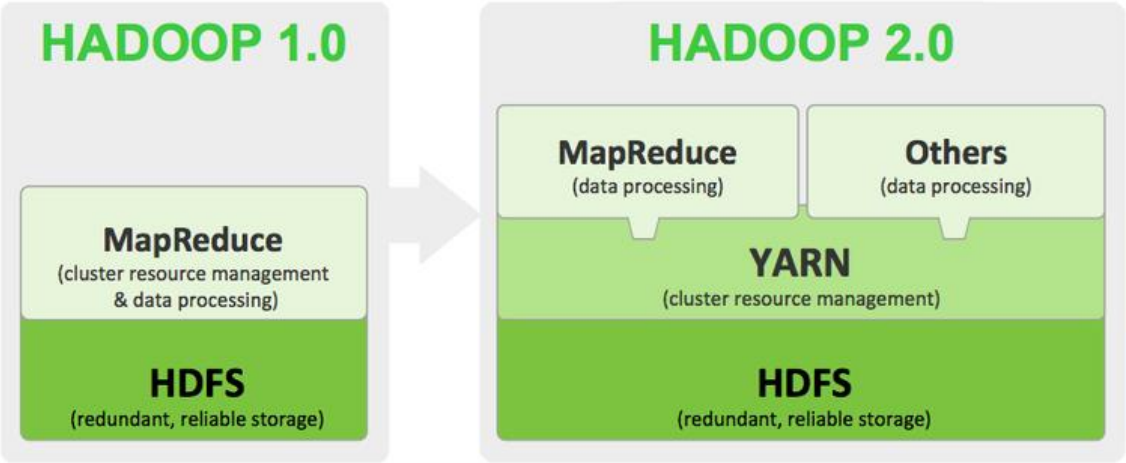

Ilustración 2 - Arquitectura Hadoop. Fuente: Medium – Cluster in yarn

2.3.2 Sistema de ficheros HDFS

Es el sistema de almacenamiento distribuido utilizado por *Hadoop*. Sus características son: Tolerante a fallos, gran capacidad de almacenamiento, capacidad de escalar de forma incremental y puede sobrevivir a fallos de hardware sin perder datos, gracias a la réplica de los ficheros entre nodos. Al ser distribuido, se reparte el trabajo entre los nodos restantes del clúster. [16]

2.3.3 Apache Hadoop Yarn

Es un gestor encargado de separar las funcionalidades de la aplicación a ejecutar. Dicho gestor es el que usa *Hadoop* por defecto. La arquitectura de *Yarn* se compone de un *resource manager* y de varios *node manager* [2]. Digamos que el *resource manager* es el master del clúster que actúa de mediador de los recursos, y los *node manager* los esclavos, encargados de lanzar y manejar los contenedores del nodo. Los contenedores son tareas especificadas por el *AppMaster*. Éste es un trozo de tarea generado por *Yarn* que pertenece a los *node manager*, indican tareas que deben ejecutarse. Al estar separado por trozos, debe haber un orden de ejecución. Se puede observar con más detalle en la siguiente ilustración. Podemos concluir que *Yarn* consigue que se ejecute el código por lotes y de forma secuencial.

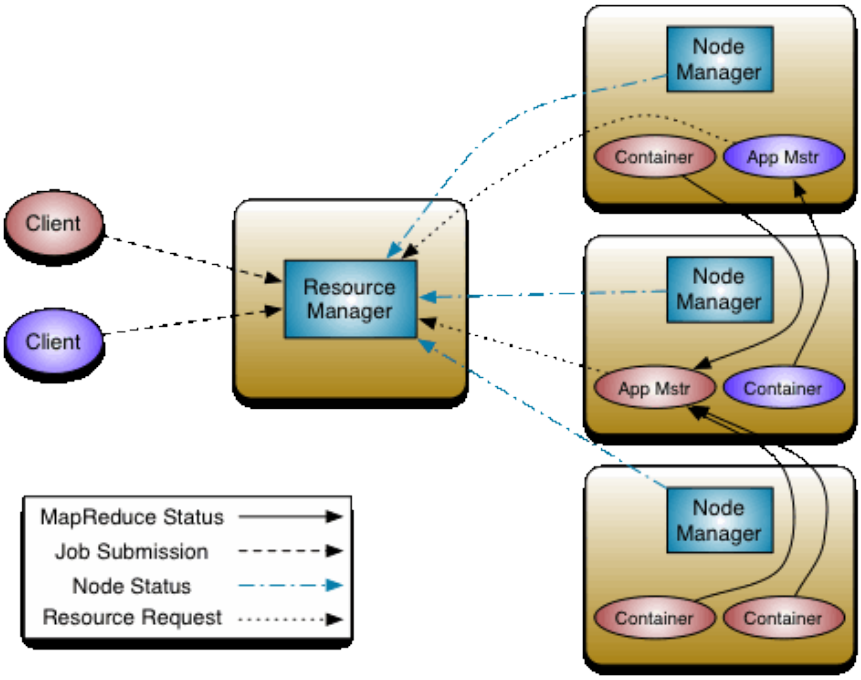

Ilustración 3 - Arquitectura Apache Hadoop YARN. Fuente: Sitio web oficial, Apache Hadoop. [3]

2.3.4 Spark

Es un entorno distribuido y paralelo que trabaja en memoria. Esto quiere decir que las tareas que *Spark* distribuya se ejecutarán de forma paralela unas a las otras, en vez de ir de forma secuencial. Se recomienda que trabaje sobre sistema de ficheros *HDFS* y además se puede lanzar la aplicación *Spark* sobre *Yarn*.

La arquitectura de *Spark* se divide en, *cluster manager* y *worker nodes*. Las tareas se coordinan gracias al *driver program* que contiene el *SparkContext*.

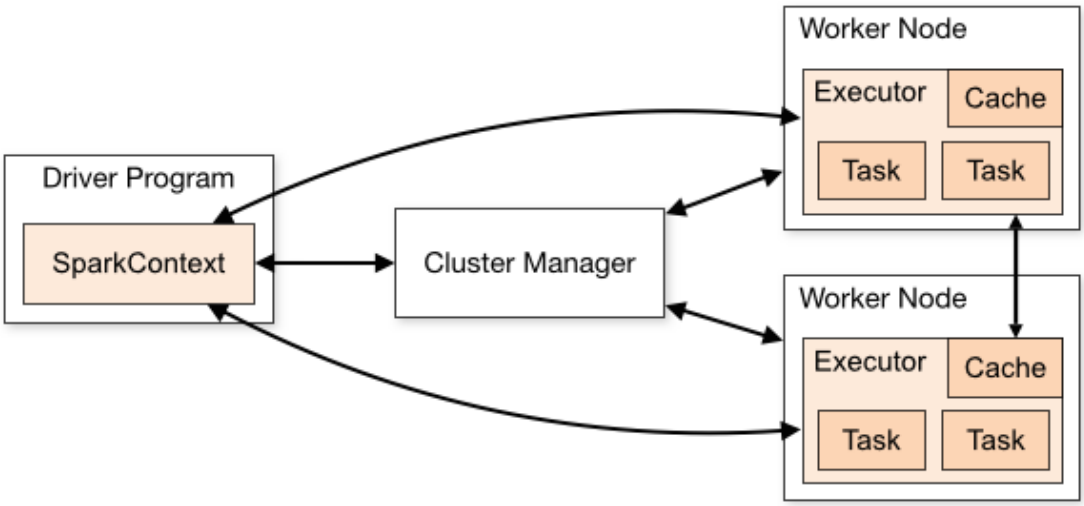

Ilustración 4 - Arquitectura clúster de Spark. Fuente: Sitio web oficial, Apache Spark. [4]

Como se puede observar en la ilustración 3, el *SparkContext* es accedido tanto por el gestor como por los trabajadores, esto es debido a que la aplicación programada se le hace referencia para poder acceder a las funcionalidades de la librería *Spark*. Se encarga en los nodos de reservar el espacio denominado *executor*, encargado de ejecutar las tareas (*Task*) y gestionar la caché.

Además de gestionar el clúster, proporciona un pilar de librerías bastante interesantes para el manejo y análisis de los datos.

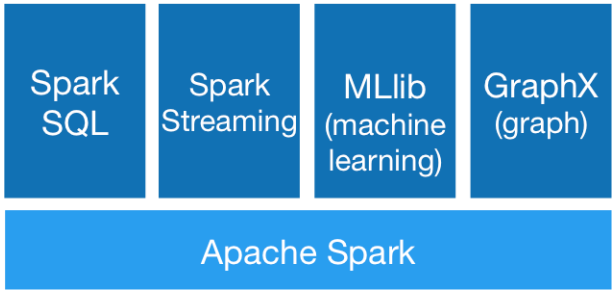

Ilustración 5 - librerías principales que sustentan Spark. Fuente: Sitio web oficial, Apache Spark.

- Streaming: Capaz de recopilar datos en periodos de tiempo, definidos por el programador. La fuente puede ser a través de socket, archivos o cualquier otra.

- SparkSQL: Accede a fuentes de datos, ya sea fichero o base de datos. Los datos deben estar estructurados o semiestructurados, esta limitación existe ya que una no estructuración de los datos evita poder utilizar lenguaje de consultas SQL[9], que es la base de esta librería, la finalidad de poder manejar los datos de forma fácil utilizando lenguaje SQL.

- GraphX: Librería para procesamiento paralelo y distribuido en grafos. En vez de asociar registros o columnas, se asocian vértices y bordes de un grafo.

- Mlib: Como el nombre indica, es una librería para *Machine Learning*[10]. Se puede utilizar para distintos entornos, tanto financieros, de educación o algoritmos de control de seguridad, por ejemplo.

2.3.5 SparkSQL

Esta tecnología es la principal de este proyecto, por lo que se va a explicar en detalle a continuación.

Como ya vimos previamente, *SparkSQL* se encarga de procesar estructuras de datos. Dichas estructuras pueden provenir de distintas fuentes de datos, manejadas por

[9] Structured Query Languaje, lenguaje de consultas estructurados.
[10] Campo de las ciencias de la computación y rama de la inteligencia artificial, cuyo objetivo es desarrollar técnicas que permitan que las máquinas aprendan. [5]

el *Data Source API*, librería de acceso a los datos de *SparkSQL*. Además, se provee de otra librería, que funcionaría como capa de abstracción a los datos, *DataFrame API*. Estos *DataFrames* se cargarán en memoria, y luego a través de sentencias *SQL* o *HQL*[11] haremos el tratamiento de los datos.

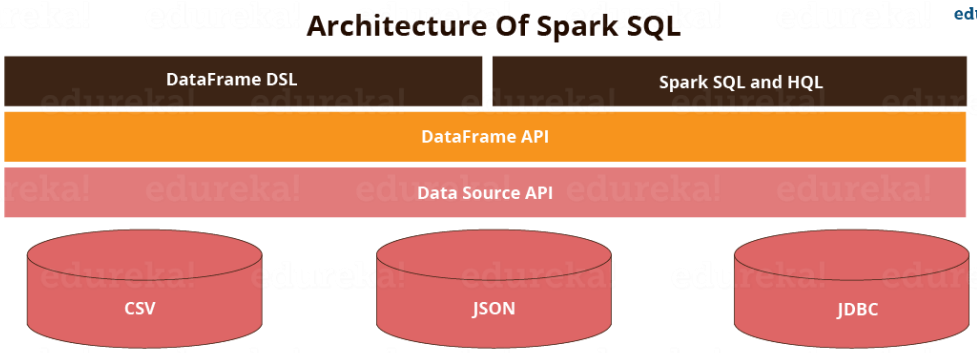

Ilustración 6 - Arquitectura de SparkSQL. Fuente: Edureka, turorial SparkSQL. [18]

- Data Source API: Lectura y almacenamiento de datos semiestructurados o estructurados.

- DataFrame API: Convierte los datos a tablas relacionales de manera interna.

- Interpretador y optimizador: Manipulación de los *DataFrames*.

- Servicio SQL: Consulta y manejo de los datos.

2.3.6 Ruby on Rails

Framework [12] para desarrollo web de código abierto escrito en lenguaje de programación *Ruby*. Utilizado en este proyecto para realizar la parte de la aplicación web. Su filosofía se basa en utilizar el patrón de diseño, modelo vista controlador (MVC). En resumidas cuentas, el usuario ve la vista que actualiza el controlador asociado a la acción requerida por el usuario, y el modelo son los tipos de datos personalizados de la aplicación.

[11] Lenguaje de consultas Hibernate. Parecido a SQL, pero orientado a objetos, por tanto, comprende nociones como herencia, polimorfismo y asociación. [6]
[12] Conjunto estandarizado de conceptos, prácticas y arquitectura software, con la finalidad de ofrecer una funcionalidad definida en alta cohesión y bajo acoplamiento. [7]

2.3.7 R

Lenguaje de programación con un enfoque al análisis estadístico. Utilizado para homogeneizar los *dataset* utilizados, y también para el análisis de *Speed-Up*[13] realizado al clúster. [8]

2.3.8 Git

Es una herramienta gratis y de código libre utilizada para el control de versiones, diseñada para manejar proyectos de programación a pequeña o gran escala. [10]

2.3.9 Scala

Scala combina programación orientada a objetos y programación funcional en un conciso y alto nivel lenguaje. Utiliza la máquina virtual de java para una construcción de un ecosistema de librerías de alto rendimiento. [20]

2.3.10 SBT

Utilizad que permite empaquetar aplicaciones Scala, además de poder gestionar las dependencias y librerías que utilices en tu programa. Proporciona una consola en la cual se indica comandos, ya sea para importar las librerías o empaquetar el software. [21]

2.4 Herramientas

2.4.1 Ruby Mine

Entorno de programación especializado en el desarrollo de aplicaciones bajo el *framework ruby on rails*. Capaz de tratar de manera ágil y eficiente el código que se utiliza en entornos web. Es software propietario, proporciona licencia para estudiantes, de la cual se ha hecho uso. [19]

2.4.2 IntelliJ

[13] Proceso utilizado para mejorar el rendimiento de un sistema que procesa un problema determinado. Análisis de la mejora en la velocidad de ejecución de una tarea ejecutada en dos arquitecturas similares.

Entorno de programación de la misma empresa que *Ruby Mine*, pero éste especializado en lenguajes como *Java* y *Scala*. Tiene versión para la comunidad con licencia gratuita, orientado a código libre, y versión para *Java EE*[14], para software comercial con licencia de pago. [22]

2.4.3 R Studio

Programa pensado para usar como entorno para R, incluye un editor de código, herramienta de visualización, por ejemplo, de gráficas y valores de variables, además, una herramienta para la depuración del código escrito en R. [23]

2.4.4 Mozilla Firefox y Google Chrome

Navegadores web potentes con consola de depuración de *javascript*, muy útil para el desarrollo de este proyecto, ya que el mapa de calor está programado utilizando la librería de *Google Maps API v3* en *javascript*.

2.4.5 Github Desktop

Software desarrollado por *GitHub*, que sincroniza carpetas de tu ordenador a un repositorio *git* de su plataforma, de manera que permite tener de manera ágil sincronizado nuestros proyectos con *GitHub*.

2.5 Trabajo realizado

2.5.1 Configuración Hadoop y Spark *standalone*

Se va a explicar la configuración del entorno de trabajo con la que se ha desarrollado este proyecto y su entorno de pruebas con *Spark standalone*, lo que incluye la explicación de todos los pasos necesarios.

2.5.1.A Instalación Linux Fedora 28:

Una instalación sencilla de *Fedora 28* en idioma español con distribución de teclado español es suficiente para continuar instalando el software necesario una vez instalado el sistema operativo [27].

[14] Versión Enterprise de Java, orientado a servicios web.

Se ha elegido este sistema operativo en el sistema y entorno de pruebas al ser el más próximo en semejanza al estudiado en las asignaturas prácticas del grado, *CentOS*.

2.5.1.B Instalación Java 8:

La instalación de *Java 8 JDK* es necesaria por parte de *Hadoop* y *Spark* ya que están programados en dichos lenguajes. El requisito mínimo de la versión de Java necesaria es Java 8 [34]. Por tanto, se ha decidido instalar la versión mínima necesaria, y no otra superior. Descargar el *RPM* de *Linux x64, jdk8u192-linux-x64.rpm*, y ejecutarlo de la página oficial de *Oracle Java* [33].

2.5.1.C Instalación Hadoop 3.1.0:

Descargamos el software desde la página web oficial, en el caso de este proyecto, se utilizó la versión 3.1.0. [29]

Se extrae el fichero descargado y se copia en */opt/hadoop*, y luego se configuran las variables de entorno, estas variables son necesarias para una ejecución de comandos tanto de *Java* como *Hadoop* sin tener que escribir la ruta completa de la ubicación de los binarios de cada programa.

```
#: mkdir /opt/hadoop
#: tar -xvzf hadoop-3.1.0.tar.gz -C /opt/hadoop
$: echo "export JAVA_HOME=/usr/java/jdk1.8.0_192-amd64" >>
~/.bashrc
$: echo "export HADOOP_HOME=/opt/hadoop/hadoop-3.1.0" >>
~/.bashrc
$: cd ~ ; . .bashrc
$: echo "export
PATH:$PATH:$HADOOP_HOME/bin:$HADOOP_HOME/sbin" >> ~/.bashrc
$: cd ~ ; . .bashrc
```

A continuación, vamos a comprobar que está instalado correctamente con el siguiente comando.

```
#: cd /opt/hadoop/Hadoop-3.1.0/share/hadoop/mapreduce
#: hadoop jar hadoop-mapreduce-examples-3.1.0.jar grep
/usr/share/dict/words /tmp/salida '[a-z]+'
```

Si se ejecuta correctamente significa que todo lo anterior se ha hecho correctamente, si hay algo que falla se deberá revisar los pasos anteriores. Ya que ahora se va a configurar el sistema para que actúe de clúster con un solo nodo. Para ello, aunque actúe todo en la misma máquina, es necesario, por comodidad, darle un alias,

por ejemplo, **nodo1**. Simplemente con editar el fichero */etc/hosts* y añadir **nodo1** al final de la primera línea, separado por un espacio, ya se tendría configurado el alias.

El primer archivo por modificar en la configuración de *hadoop* es el *$HADOOP_HOME/etc/hadoop/core-site.xml*, es el archivo donde se especifican los nodos que van a actuar dentro del sistema de ficheros *distribuido HDFS*, debe quedar de la siguiente forma:

```
<configuration>
      <property>
            <name>fs.defaultFS</name>
            <value>hdfs://nodo1:9000</value>
      </property>
</configuration>
```

Archivo $HADOOP_HOME/etc/Hadoop/core-site.xml

El segundo archivo es *$HADOOP_HOME/etc/hadoop/hdfs-site.xml*, encargado de cargar la configuración de *HDFS* al arrancar el sistema de ficheros, así como la réplica de los datos, la ubicación del directorio contenedor utilizado por el *namenode* y *datanode*.

```
<configuration>
      <property>
            <name>dfs.replication</name>
            <value>1</value>
      </property>
      <property>
            <name>dfs.namenode.name.dir</name>
            <value>/datos/namenode</value>
      </property>
      <property>
            <name>dfs.datanode.data.dir</name>
            <value>/datos/datanode</value>
      </property>
</configuration>
```

Archivo $HADOOP_HOME/etc/hadoop/hdfs-site.xml

Por tanto, debemos crear el directorio */datos* y dentro, */datos/namenode* y */datos/datanode*, porque es el asignado en el archivo de configuración y asignarle permisos al usuario que vaya a arrancar el sistema *HDFS*, que **no sea root**. Y en último lugar, para terminar con la configuración de HDFS, debemos formatear los directorios especificados en la configuración con el siguiente comando:

```
#: hdfs namenode -format
```

A continuación, debemos configurar la comunicación entre nodos, que se hace por *SSH*, y la autentificación por certificado de clave pública. Aunque sea la misma

máquina, hará un *SSH* a nodo1, que es el propio sistema, es la manera de trabajar entre nodos que tiene *Hadoop*. Con los comandos siguientes podemos configurarlo:

```
$: ssh-keygen
$: cd ~/.ssh
$: cp id_rsa.pub authorized_keys
$: chkconfig sshd on
$: service sshd start
```

Ahora solo falta arrancar el sistema *HDFS* y acceder a su panel web, que nos permite ver un resumen del sistema de ficheros.

```
$: start-dfs.sh
URL Hadoop: http://nodo1:9870
```

2.5.1.D Instalación Spark 2.3.0:

Descargar la versión 2.3.0 sin *hadoop* de la página web oficial, para luego extraer y copiar en */opt/hadoop*. También habrá que configurar nuevas variables de entorno [30]:

```
$: tar -xvzf spark-2.3.0-bin-without-hadoop.tgz -C
/opt/hadoop/hadoop-3.1.0
$: mv /opt/hadoop/hadoop-3.1.0/spark-2.3.0-bin-without-
hadoop /opt/hadoop/hadoop-3.1.0/spark-2.3.0
$: echo "export PATH=$PATH:/opt/hadoop-3.1.0/spark-
2.3.0/bin:/opt/hadoop-3.1.0/spark-2.3.0/sbin" >> ~/.bashrc
$: echo "export SPARK_DIST_CLASSPATH=$(hadoop classpath)"
>> ~/.bashrc
$: echo "export SPARK_HOME=/opt/hadoop/hadoop-3.1.0/spark-
2.3.0" >> ~/.bashrc
$: echo "export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop" >>
~/.bashrc
$: cd ~ ; . .bashrc
```

Y finalmente para probar que se ha ejecutado todo de forma correcta, con el siguiente comando ejecutamos la consola de *Spark* con *Scala*:

```
$: spark-shell
```

2.5.1.E Instalación Jetbrains Toolbox App:

Se va a utilizar esta herramienta para descargar *Ruby Mine* e *IntelliJ*, entornos de desarrollo profesionales, para nuestra parte de *Ruby* y *Scala* respectivamente. [35]

Necesitaremos identificarnos para poder acceder al paquete de software, si no dispone de cuenta y es estudiante o profesor, se crea una cuenta utilizando el correo institucional, en nuestro caso de la Universidad de Las Palmas de Gran Canaria. Dentro de la aplicación instalamos el *IntelliJ IDEA Ultimate* y *RubyMine*. [36]

2.5.1.F Instalación de SBT para Scala:

La instalación es sencilla, requiere de un comando, y ya a través del *IntelliJ* se utiliza el *SBT*.

```
$: dnf install sbt
```

2.5.1.G Instalación Ruby on Rails:

Es un *framework* que en Fedora es fácil de instalar, ya que existe un grupo de instalación por parte de Fedora, con los siguientes comandos instalaremos los paquetes necesarios para crear una aplicación *rails* y crear un proyecto de ejemplo.

```
#: dnf install rubygem-rails
#: dnf group install 'Ruby on Rals'
#: dnf install sqlite-devel
$: cd ~ ; rails new AppPrueba --skip-bundle
$: cd AppPrueba
$: bundle --local
$: rails s
```

[31]

Si la ejecución del último comando no ha dado ningún error y http://localhost:3000 entra sin problemas, todo ha ido correctamente.

2.5.2 Configuración Hadoop y Spark en clúster

2.5.2.A Instalación Linux, Ubuntu 17:

Se ha elegido la instalación de *Ubuntu 17 server* [28] para todos los nodos del clúster. Una instalación mínima es suficiente, ya que luego instalaremos los paquetes necesarios de cada software. Los pasos de instalación son un poco distintos a una versión *desktop*[15], no obstante, sigue siendo una instalación bastante intuitiva. Cabe destacar que no es una instalación desatendida, por lo que en el proceso de instalación

[15] Versión de Ubuntu Linux especializada para entornos de escritorio.

es probable que pida la interacción del administrador de sistemas que esté instalando el sistema operativo.

Importante, hay que especificar el nombre de host acorde al esquema del clúster y el nombre de usuario común a todos, en nuestro caso: ***cicei***

2.5.2.B Configuración del clúster a nivel de red:

Debemos configurar en cada nodo, los alias según el esquema de red, por lo que se debe modificar el archivo */etc/hosts* como se ve a continuación.

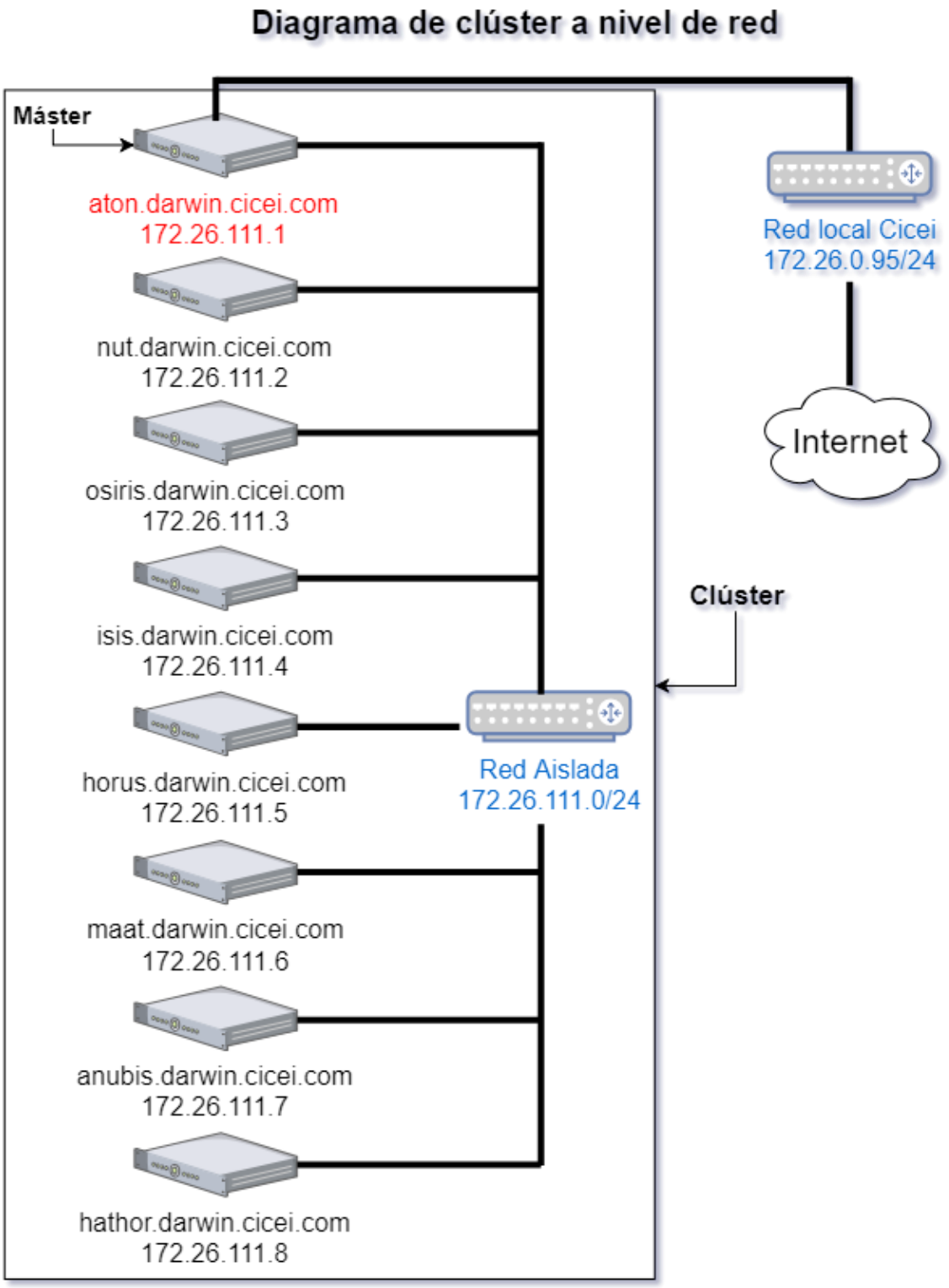

Ilustración 7 - Esquema de red del clúster. Fuente: elaboración propia.

```
127.0.0.1       localhost
# The following lines are desirable for IPv6 capable hosts
::1     localhost ip6-localhost ip6-loopback
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters

# Nombres del cluster
172.26.111.1    aton.darwin.cicei.com           aton
172.26.111.2    nut.darwin.cicei.com            nut
172.26.111.3    osiris.darwin.cicei.com         osiris
172.26.111.4    isis.darwin.cicei.com           isis
172.26.111.5    horus.darwin.cicei.com          horus
172.26.111.6    maat.darwin.cicei.com           maat
172.26.111.7    anubis.darwin.cicei.com         anubis
172.26.111.8    hathor.darwin.cicei.com         hathor
```

Archivo /etc/hosts resultante en cada nodo

En Ubuntu la configuración de red se hace en el archivo */etc/network/interfaces*, que, en el caso del maestro sería [37]:

```
# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
auto enp4s4f0
iface enp4s4f0 inet static
    address 172.26.111.1
    netmask 255.255.255.0
    network 172.26.111.0
    broadcast 172.26.111.255

auto enp4s4f1
iface enp4s4f1 inet dhcp
```

Archivo /etc/network/interfaces del nodo maestro

Y para las máquinas que actuarán de esclavo, sería la siguiente configuración, solo configurando la interfaz de red conectada a la red aislada:

```
# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
auto enp4s4f0
iface enp4s4f0 inet static
    address 172.26.111.x <- IP del esquema de red
    netmask 255.255.255.0
    network 172.26.111.0
    broadcast 172.26.111.255
```

Archivo /etc/network/interfaces de los nodos esclavos

Antes de pasar a la siguiente instalación, se va a configurar la conexión *ssh* mediante certificados de clave pública, para que todos los nodos puedan comunicarse con cualquier nodo mediante autentificación por certificado.

Los siguientes comandos, deben ser ejecutados en cada una de las máquinas, ya que debemos generar nuestro par de claves públicas y privadas por máquina, además, debemos pasar nuestra clave pública a todas las máquinas, de tal forma, que todas las máquinas tengan las claves públicas de todas, incluida la propia máquina.

```
$: ssh-keygen
$: sudo systemctl enable ssh
$: sudo service ssh start
$: ssh-copy-id -i ~/.ssh/id_rsa.pub cicei@aton
$: ssh-copy-id -i ~/.ssh/id_rsa.pub cicei@nut
$: ssh-copy-id -i ~/.ssh/id_rsa.pub cicei@osiris
$: ssh-copy-id -i ~/.ssh/id_rsa.pub cicei@isis
$: ssh-copy-id -i ~/.ssh/id_rsa.pub cicei@horus
$: ssh-copy-id -i ~/.ssh/id_rsa.pub cicei@maat
$: ssh-copy-id -i ~/.ssh/id_rsa.pub cicei@anubis
$: ssh-copy-id -i ~/.ssh/id_rsa.pub cicei@hathor
```

2.5.2.C Instalación Java 8:

Descarga de *Java 8* de la página web oficial, como no tenemos interfaz web, vamos a hacerlo a través de comandos.

```
#: wget -c --header "Cookie: oraclelicense=accept-securebackup-cookie" https://download.oracle.com/otn-pub/java/jdk/8u192-b12/750e1c8617c5452694857ad95c3ee230/jdk-8u192-linux-x64.tar.gz
#: sudo mkdir /opt/java
$: sudo tar -zxf jdk-8u192-linux-x64.tar.gz -C /opt/java
```

```
$: sudo nano /etc/environment
# editar el fichero de variables de entorno
# añadir antes de las últimas doble comillas de PATH
:/opt/java/jdk1.8.0_192/bin
# añadir la línea de abajo
JAVA_HOME="/opt/java/jdk1.8.0_192"
```

```
$: source /etc/environment
```

```
$: java -version
```

Si el comando imprime la versión 1.8.0, entonces ya se dispone de *Java 8* instalado en el sistema. Como no podía ser de otra forma, todos los nodos deben tener *Java* instalado.

2.5.2.D Instalación Hadoop 3.1.0:

La instalación inicial en modo clúster sigue los pasos iniciales a la explicación anterior en *standalone*, con alguna variante, además, habrá unos pasos extras que será replicar la instalación en los demás nodos y una nueva configuración de *Yarn*.

```
$: wget
https://archive.apache.org/dist/hadoop/common/hadoop-
3.1.0/hadoop-3.1.0.tar.gz
```

```
$: sudo mkdir /opt/hadoop
```

```
$: sudo tar -xvzf hadoop-3.1.0.tar.gz -C /opt/hadoop
```

```
$: nano /etc/environment
# añadir en PATH antes de las últimas doble comillas
:/opt/hadoop/hadoop-3.1.0/bin:/opt/hadoop/hadoop-3.1.0/sbin
# añadir nuevas líneas
HADOOP_HOME=/opt/hadoop/hadoop-3.1.0
HADOOP_CONF_DIR=/opt/hadoop/hadoop-3.1.0/etc/hadoop
```

```
$: source /etc/environment
```

```
$: Hadoop version
```

Si sale la versión de *Hadoop 3.1.0*, es que se ha instalado correctamente y podemos continuar. Seguimos los pasos de configuración del clúster en *standalone*, con alguna variación en el *hdfs-sites.xml*.

El primer archivo por modificar en la configuración de *hadoop* es el *$HADOOP_HOME/etc/hadoop/core-site.xml*, es el archivo donde se especifican los nodos que van a actuar dentro del sistema de ficheros *distribuido HDFS*, debe quedar de la siguiente forma:

```
<configuration>
      <property>
            <name>fs.defaultFS</name>
            <value>hdfs://aton.darwin.cicei.com:9000</value>
      </property>
</configuration>
```

Archivo $HADOOP_HOME/etc/hadoop/core-site.xml de todos los nodos

El segundo archivo es *$HADOOP_HOME/etc/hadoop/hdfs-site.xml*, encargado de cargar la configuración de *HDFS* al arrancar el sistema de ficheros, así como la réplica de los datos, la ubicación del directorio contenedor utilizado por el *namenode* y *datanode*. En este caso se observa que la réplica de cada archivo en el sistema de ficheros distribuido es de tres copias, a pesar de ser éste por defecto, lo especificamos en la configuración que es lo que se recomienda cuando se tiene más de 3 nodos en el clúster.

```
<configuration>
      <property>
            <name>dfs.replication</name>
            <value>3</value>
      </property>
      <property>
            <name>dfs.namenode.name.dir</name>
            <value>/datos/namenode</value>
      </property>
      <property>
            <name>dfs.datanode.data.dir</name>
            <value>/datos/datanode</value>
      </property>
</configuration>
```

Archivo $HADOOP_HOME/etc/hadoop/hdfs-site.xml de todos los nodos

Por tanto, debemos crear el directorio */datos* y dentro, */datos/namenode* y dar permisos de propietario a **cicei**. El archivo */datos/datanode* sólo va en los esclavos, al que también se le debe otorgar permisos, se verá más adelante en la configuración de los esclavos.

En último lugar, para terminar con la configuración de HDFS, debemos formatear el directorio especificado en la configuración con el siguiente comando, recordemos, sólo en el maestro:

```
#: hdfs namenode -format
```

2.5.2.E Configuración Yarn:

Este paso es nuevo con respecto a la configuración del entorno *Hadoop Spark standalone*. Los archivos que hay que modificar son: *mapred-site.xml*, encargado de la configuración del *map reduce*, y el archivo *yarn-site.xml,* encargado de la configuración del gestor de procesos del clúster.

Configuramos el archive *mapred-site.xml* para especificar que el motor que vamos a utilizar con *Hadoop* va a ser *Yarn*.

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

Archivo $HADOOP_HOME/etc/hadoop/mapred-site.xml

Ahora toca configurar el yarn-site.xml, en primer lugar, la máquina que gestiona *Yarn*, el *resource manager*. En segundo lugar, es la clase que gestiona el *map-reduce*, y hay que añadir donde se encuentra la clase.

```
<property>
     <name>yarn.resourcemanager.hostname</name>
     <value>aton.darwin.cicei.com</value>
</property>
<property>
     <name>yarn.nodemanager.aux-services</name>
     <value>mapreduce_shuffle</value>
</property>
<property>
     <name>yarn.nodemanager.aux-
services.mapreduce_shuffle.class</name>
     <value>org.apache.hadoop.mapred.ShuffleHandler</value>
</property>
<property>
     <name>yarn.application.classpath</name>
     <value>
            /opt/hadoop/hadoop-3.1.0/etc/hadoop,
            /opt/hadoop/hadoop-3.1.0/share/hadoop/common/*,
            /opt/hadoop/hadoop-3.1.0/share/hadoop/common/lib/*,
            /opt/hadoop/hadoop-3.1.0/share/hadoop/hdfs/*,
            /opt/hadoop/hadoop-3.1.0/share/hadoop/hdfs/lib/*,
            /opt/hadoop/hadoop-3.1.0/share/hadoop/mapreduce/*,
/opt/hadoop/hadoop-3.1.0/share/hadoop/mapreduce/lib/*,
            /opt/hadoop/hadoop-3.1.0/share/hadoop/yarn/*,
            /opt/hadoop/hadoop-3.1.0/share/hadoop/yarn/lib/*
     </value>
</property>
```

Archivo $HADOOP_HOME/etc/hadoop/yarn-site.xml

Para configurar este entorno en modo clúster primero debemos asegurarnos de que sean idénticos, comprobando los siguientes pasos:

- Mismo usuario en todos los nodos.
- Accesible por *ssh* autentificándose por certificado.
- Tener los mismos directorios de *hadoop* en cada nodo.
- Copiar el software *hadoop* en todos los nodos.
- Tener el directorio de datos en todos los nodos con permisos.

Por tanto, en cada esclavo, creamos la carpeta */opt/hadoop* y damos permisos de propietario al usuario, en nuestro caso **cicei**. Y luego desde el maestro, enviamos los archivos de configuración.

En el maestro
`$: sudo scp -r /opt/hadoop cicei@nut:/opt`
`$: sudo scp -r /opt/hadoop cicei@osiris:/opt`
`$: sudo scp -r /opt/hadoop cicei@isis:/opt`
`$: sudo scp -r /opt/hadoop cicei@horus:/opt`
`$: sudo scp -r /opt/hadoop cicei@maat:/opt`
`$: sudo scp -r /opt/hadoop cicei@anubis:/opt`
`$: sudo scp -r /opt/hadoop cicei@hathor:/opt`
`$: sudo scp /etc/environment cicei@nut:/opt/hadoop`
`$: sudo scp /etc/environment cicei@osiris:/opt/hadoop`
`$: sudo scp /etc/environment cicei@isis:/opt/hadoop`
`$: sudo scp /etc/environment cicei@horus:/opt/hadoop`
`$: sudo scp /etc/environment cicei@maat:/opt/hadoop`
`$: sudo scp /etc/environment cicei@anubis:/opt/hadoop`
`$: sudo scp /etc/environment cicei@hathor:/opt/hadoop`
En los esclavos
`$: sudo mkdir /opt/hadoop`
`$: sudo chown cicei:cicei /opt/hadoop`
`$: sudo mv /op/hadoop/environment /etc/environment`
`$: source /etc/environment`
`$: sudo mkdir /datos`
`$: sudo mkdir /datos/datanode`
`$: sudo chown -R cicei:cicei /datos`

Ahora hay que especificar en el maestro, qué esclavos va a tener, es tan simple como editar el archivo *$HADOOP_CONF_DIR/workers* y añadir la dirección de cada esclavo, en nuestro caso:

```
nut.darwin.cicei.com
osiris.darwin.cicei.com
isis.darwin.cicei.com
horus.darwin.cicei.com
maat.darwin.cicei.com
anubis.darwin.cicei.com
hathor.darwin.cicei.com
```

Archivo $HADOOP_HOME/etc/hadoop/workers

Para comprobar que la configuración en modo clúster está funcionando hasta el momento, ejecutamos el siguiente comando:

```
$: start-dfs.sh
$: jps
```

El comando *jps* nos indica las aplicaciones javas ejecutándose, por tanto, debe asemejarse a las siguientes ilustraciones la salida del comando *jps*. Si todo está correcto, significa que la configuración *hadoop*, *hdfs* y *yarn* está configurado en modo clúster.

```
4165 NameNode
4423 SecondaryNameNode
4543 Jps
```

Ilustración 8 - salida del comando jps en el maestro. Fuente: Elaboración propia.

```
2241 DataNode
2307 Jps
```

Ilustración 9 - salida comando jps en los esclavos. Fuente: Elaboración propia.

2.5.2.F Instalación Spark 2.3.0:

En este punto ya estamos a punto de terminar la configuración del clúster, son pasos más sencillos que los anteriores, ya que *Spark* lo vamos a ejecutar sobre *Yarn*, y como hemos visto previamente, lo hemos dejado preparado para ello.

Vamos a seguir los pasos anteriores del clúster *standalone*, adaptándolo al clúster posteriormente.

```
$: wget https://archive.apache.org/dist/spark/spark-2.3.0/spark-2.3.0-bin-without-hadoop.tgz
$: tar -xvzf spark-2.3.0-bin-without-hadoop.tgz -C /opt/hadoop/hadoop-3.1.0
$: mv /opt/hadoop/hadoop-3.1.0/spark-2.3.0-bin-without-hadoop /opt/hadoop/hadoop-3.1.0/spark-2.3.0
$: sudo nano /etc/environment
```

```
# añadir antes de las últimas doble comillas de PATH
:/opt/hadoop-3.1.0/spark-2.3.0/bin:/opt/hadoop-3.1.0/spark-
2.3.0/sbin
# añadir nuevas líneas
SPARK_DIST_CLASSPATH=$(hadoop classpath)
SPARK_HOME="/opt/hadoop/hadoop-3.1.0/spark-2.3.0"
```

```
$: source /etc/environment
```

También hay que indicarle a *Spark*, quienes van a ser los esclavos del clúster, en el archivo *$SPARK_HOME/conf/slaves*.

```
$: cp /opt/hadoop/hadoop-3.1.0/etc/hadoop/workers
/opt/hadoop/hadoop-3.1.0/spark-2.3.0/conf/slaves
```

Antes de probar que funcione, hay que pasar la configuración a los esclavos, como hicimos previamente con la configuración de *Hadoop* y *Yarn*.

```
                    En el maestro
  # Como vimos anteriormente, de master a todos los slaves
$: scp -r /opt/hadoop/hadoop-3.1.0/spark-2.3.0
cicei@nut:/opt/hadoop/hadoop-3.1.0/
$: sudo scp /etc/environment cicei@nut:/opt/hadoop
                    En los esclavos
$: sudo mv /op/hadoop/environment /etc/environment
$: source /etc/environment
```

Esta configuración, es básica, y funcionaría el clúster de manera funcional y se podrían lanzar aplicaciones sin ningún tipo de problema, no obstante, queremos activar *Dynamic Resource Allocation* de *Spark*, ya que nos permite ajustar los recursos de forma dinámica según la carga de trabajo, es decir, si el clúster deja de utilizar algún tipo de recurso, lo libera, si necesita más, lo coge, este funcionamiento está previsto para ejecuciones más largas y no deje el clúster con recursos inutilizables que no está utilizando. [39]

Entonces, vamos a modificar la configuración actual de *Yarn* para que sea compatible con *Dynamic Resource Allocation*, es decir, los siguientes pasos son opcionales, pero en nuestro proyecto es obligatorio ya que ejecutamos nuestra aplicación utilizando esta tecnología, pero para otro entorno no es obligatorio.

Estos cambios sólo se aplican a los *NodeManager*, es decir a los nodos esclavos. Se debe añadir las siguientes propiedades:

- Nombre: *yarn.nodemanager.aux-services*, valor: *spark_shuffle*
- Nombre: *yarn.nodemanager.aux-services.spark_shuffle.class*, valor: *org.apache.spark.network.yarn.YarnShuffleService*

- Nombre: *yarn.nodemanager.aux-services.spark_shuffle.classpath*, valor: */opt/hadoop/hadoop-3.1.0/spark-2.3.0/yarn/**

Y finalmente añadir */opt/hadoop/hadoop-3.1.0/spark-2.3.0/jars* en *yarn.application.classpath* se debe eliminar la propiedad al completo, con nombre, *yarn.nodemanager.aux-services.mapreduce_shuffle.class*, por lo que el fichero nuevo quedaría de esta forma final.

```
<property>
      <name>yarn.resourcemanager.hostname</name>
      <value>aton.darwin.cicei.com</value>
</property>
<property>
      <name>yarn.nodemanager.aux-services</name>
      <value>spark_shuffle</value>
</property>
<property>
      <name>yarn.nodemanager.aux-
services.spark_shuffle.class</name>
      <value>
org.apache.spark.network.yarn.YarnShuffleService</value>
</property>
<property>
      <name>yarn.nodemanager.aux-
services.spark_shuffle.classpath</name>
      <value>/opt/hadoop/hadoop-3.1.0/spark-2.3.0/yarn/*</value>
</property>
<property>
      <name>yarn.application.classpath</name>
      <value>
      /opt/hadoop/hadoop-3.1.0/etc/hadoop,
      /opt/hadoop/hadoop-3.1.0/share/hadoop/common/*,
      /opt/hadoop/hadoop-3.1.0/share/hadoop/common/lib/*,
      /opt/hadoop/hadoop-3.1.0/share/hadoop/hdfs/*,
      /opt/hadoop/hadoop-3.1.0/share/hadoop/hdfs/lib/*,
      /opt/hadoop/hadoop-3.1.0/share/hadoop/mapreduce/*,
       /opt/hadoop/hadoop-3.1.0/share/hadoop/mapreduce/lib/*,
      /opt/hadoop/hadoop-3.1.0/share/hadoop/yarn/*,
      /opt/hadoop/hadoop-3.1.0/share/hadoop/yarn/lib/*,
      /opt/hadoop/hadoop-3.1.0/spark-2.3.0/jars
      </value>
</property>
```

Archivo $HADOOP_HOME/etc/hadoop/yarn-site.xml de los esclavos (*NodeManagers*) [38]

En este punto, ya podemos arrancar el servicio de *Yarn* en el maestro con el siguiente comando, y verificar la salida del comando *JPS* sea similar a las siguientes ilustraciones.

```
$: start-yarn.sh
```

```
$: jps
```

```
6513 SecondaryNameNode
7347 ResourceManager
7646 Jps
6255 NameNode
```

Ilustración 10 - salida del comando *jps* en maestro. Fuente: Elaboración propia.

```
3592 NodeManager
3688 Jps
3130 DataNode
```

Ilustración 11 - salida del comando *jps* en esclavos. Fuente: Elaboración propia.

2.5.2.G Instalación Ruby on Rails:

Esta tecnología sólo la instalamos en el nodo maestro, ya que es el cliente del clúster y va a ejecutar la operación *Spark* en modo cliente. La instalación es bastante sencilla, sólo es seguir unos comandos.

`$: curl -sL https://deb.nodesource.com/setup_8.x | sudo -E bash -`
`$: curl -sS https://dl.yarnpkg.com/debian/pubkey.gpg | sudo apt-key add -`
`$: echo "deb https://dl.yarnpkg.com/debian/ stable main" | sudo tee /etc/apt/sources.list.d/yarn.list`
`$: sudo apt-get update`
`$: sudo apt-get install git-core curl zlib1g-dev build-essential libssl-dev libreadline-dev libyaml-dev libsqlite3-dev sqlite3 libxml2-dev libxslt1-dev libcurl4-openssl-dev software-properties-common libffi-dev nodejs yarn`
`$: cd`
`$: git clone https://github.com/rbenv/rbenv.git ~/.rbenv`
`$: echo 'export PATH="$HOME/.rbenv/bin:$PATH"' >> ~/.bashrc`
`$: echo 'eval "$(rbenv init -)"' >> ~/.bashrc`
`$: exec $SHELL`
`$: git clone https://github.com/rbenv/ruby-build.git ~/.rbenv/plugins/ruby-build`
`$: echo 'export PATH="$HOME/.rbenv/plugins/ruby-build/bin:$PATH"' >> ~/.bashrc`
`$: exec $SHELL`
`$: rbenv install 2.5.3`
`$: rbenv global 2.5.3`
`$: ruby -v`
`$: gem install bundler`

```
$: curl -sL https://deb.nodesource.com/setup_8.x | sudo -E
bash -
$: sudo apt-get install -y nodejs
$: gem install rails -v 5.2.1
$: rbenv rehash
$: rails -v
```

[32]

Si todo ha ido bien, significa que ya tenemos *Ruby on Rails* instalado en el sistema, y podemos continuar con la realización del proyecto.

2.5.3 Homogeneización del dataset

2.5.3.A Obtención de los dataset:

Hemos utilizado dos conjuntos de datos disponible en la plataforma Madrid Open Data [24] ofrecida por el ayuntamiento de Madrid. Dentro del catálogo de datos de la plataforma, hemos utilizado el conjunto llamado "histórico de datos del tráfico desde 2013", además para obtener la localización de cada sensor utilizado en éste, utilizamos el conjunto llamado: "ubicación de los puntos de medida del tráfico". Esto nos presenta una limitación y es que sólo está catalogado todo el año 2018 con la ubicación de los sensores de medida, por tanto, de ambos conjuntos de datos, sólo vamos a utilizar todo el año 2018. En ambos casos, cada fichero a descargar es un mes de datos.

2.5.3.B Tratar dataset de tráfico:

Este *dataset* presenta un cierto desorden de los datos por cada fichero, por lo que, para un tratamiento adecuado, vamos a organizar los datos para que cada mes tenga las mismas columnas de datos, y sobre todo en el mismo orden. [25]

Para ello vamos a crear un script en **R**, que nos ayude a manejar los **csv** de manera fácil y cómoda.

En el caso del primer mes, hay que cambiar el texto contenido en la columna **tipo_elem** de *M-30* por *M30*, ya que todos los demás meses lo tienen de esta forma. Y de forma común a todos los demás archivos, poner los decimales con punto en vez de coma en las columnas **x** e **y**.

De manera especial los meses del cinco al nueve, las columnas están en distinto orden, y hay que mover las columnas, todas las operaciones quedan reflejadas en el siguiente código.

```
library(data.table)

# El 1 cambiar M-30 por M30, y los decimales a .
csv01 <- read.csv(file="/home/alejandro/Documentos/PuntosMedida/01-2018.csv",
header=T, sep=";", stringsAsFactors = T)
csv01 <- as.data.table(csv01)[tipo_elem == "M-30", tipo_elem := "M30"][]
csv01$x <- gsub(",", "\\.", csv01$x) # Decimal de , a .
csv01$y <- gsub(",", "\\.", csv01$y)
write.table(sep=";", row.names=FALSE,
file="/home/alejandro/Documentos/PuntosMedidaModificados/01-2018.csv", csv01)
csv01 <- NULL
# Del 2 al 4 los decimales a .
csv02 <- read.csv(file="/home/alejandro/Documentos/PuntosMedida/02-2018.csv",
header=T, sep=";", stringsAsFactors = T)
csv02$x <- gsub(",", "\\.", csv02$x)
csv02$y <- gsub(",", "\\.", csv02$y)
write.table(sep=";", row.names=FALSE,
file="/home/alejandro/Documentos/PuntosMedidaModificados/02-2018.csv", csv02)
csv02 <- NULL

csv03 <- read.csv(file="/home/alejandro/Documentos/PuntosMedida/03-2018.csv",
header=T, sep=";", stringsAsFactors = T)
csv03$x <- gsub(",", "\\.", csv03$x)
csv03$y <- gsub(",", "\\.", csv03$y)
write.table(sep=";", row.names=FALSE,
file="/home/alejandro/Documentos/PuntosMedidaModificados/03-2018.csv", csv03)
csv03 <- NULL

csv04 <- read.csv(file="/home/alejandro/Documentos/PuntosMedida/04-2018.csv",
header=T, sep=";", stringsAsFactors = T)
csv04$x <- gsub(",", "\\.", csv04$x)
csv04$y <- gsub(",", "\\.", csv04$y)
write.table(sep=";", row.names=FALSE,
file="/home/alejandro/Documentos/PuntosMedidaModificados/04-2018.csv", csv04)
csv04 <- NULL

# Del 5 al 9 mover columnas, y los decimales a .
csv05 <- read.csv(file="/home/alejandro/Documentos/PuntosMedida/05-2018.csv",
header=T, sep=";", stringsAsFactors = T)
csv05 <- csv05[,c(2,3,4,1,5,6)]
csv05$x <- gsub(",", "\\.", csv05$x)
csv05$y <- gsub(",", "\\.", csv05$y)
write.table(sep=";", row.names=FALSE,
file="/home/alejandro/Documentos/PuntosMedidaModificados/05-2018.csv", csv05)
csv05 <- NULL

csv06 <- read.csv(file="/home/alejandro/Documentos/PuntosMedida/06-2018.csv",
header=T, sep=";", stringsAsFactors = T)
csv06 <- csv06[,c(3,2,4,1,5,6)]
csv06$x <- gsub(",", "\\.", csv06$x)
csv06$y <- gsub(",", "\\.", csv06$y)
write.table(sep=";", row.names=FALSE,
file="/home/alejandro/Documentos/PuntosMedidaModificados/06-2018.csv", csv06)
csv06 <- NULL
csv07 <- read.csv(file="/home/alejandro/Documentos/PuntosMedida/07-2018.csv",
header=T, sep=";", stringsAsFactors = T)
```

```
csv07 <- csv07[,c(3,2,4,1,5,6)]
csv07$x <- gsub(",", "\\.", csv07$x)
csv07$y <- gsub(",", "\\.", csv07$y)
write.table(sep=";", row.names=FALSE,
file="/home/alejandro/Documentos/PuntosMedidaModificados/07-2018.csv", csv07)
csv07 <- NULL



csv08 <- read.csv(file="/home/alejandro/Documentos/PuntosMedida/08-2018.csv",
header=T, sep=";", stringsAsFactors = T)
csv08 <- csv08[,c(3,2,4,1,5,6)]
csv08$x <- gsub(",", "\\.", csv08$x)
csv08$y <- gsub(",", "\\.", csv08$y)
write.table(sep=";", row.names=FALSE,
file="/home/alejandro/Documentos/PuntosMedidaModificados/08-2018.csv", csv08)
csv08 <- NULL

csv09 <- read.csv(file="/home/alejandro/Documentos/PuntosMedida/09-2018.csv",
header=T, sep=";", stringsAsFactors = T)
csv09 <- csv09[,c(3,2,4,1,5,6)]
csv09$x <- gsub(",", "\\.", csv09$x)
csv09$y <- gsub(",", "\\.", csv09$y)
write.table(sep=";", row.names=FALSE,
file="/home/alejandro/Documentos/PuntosMedidaModificados/09-2018.csv", csv09)
csv09 <- NULL
```

Script en R para tratar el dataset de histórico de datos del tráfico desde 2013

Y ya finalmente debemos subir los ficheros al sistema de archivos HDFS que ya previamente hemos montado, configurado y puesto en marcha. Supongamos que lo tenemos guardado cada archivo en la ruta *"/home/alejandro/Documentos/PuntosMedidaModificados"*.

$: cd /home/alejandro/Documentos/PuntosMedidaModificados
$: hdfs dfs -mkdir /datos.madrid.es_2018/VelocidadMedia
$: hdfs dfs -put *.csv /datos.madrid.es_2018/VelocidadMedia
 # Este último es para preparar la salida del programa Scala **$: hdfs dfs -mkdir /salida**

2.5.3.C Tratar *dataset* de ubicación de sensores:

Vamos a utilizar los **csv** descargados y para agilizar el tratamiento de los datos, este *dataset* lo vamos a introducir en la base de datos de la aplicación web, ya que entero ocupa a rededor de cuatro megas y medio, y no necesita de ningún tratamiento previo. [26]

2.5.4 Programación de aplicación SparkSQL

2.5.4.A Creación proyecto SBT para Scala:

Hay que instalar *Scala* en *IntelliJ*, una manera fácil de hacerlo es la primera vez que se instala y se ejecuta, en la parte de instalación de *plugins*[16], instalar *Scala*. Ahora toca crear un proyecto, en el cual podemos seleccionar Scala con SBT

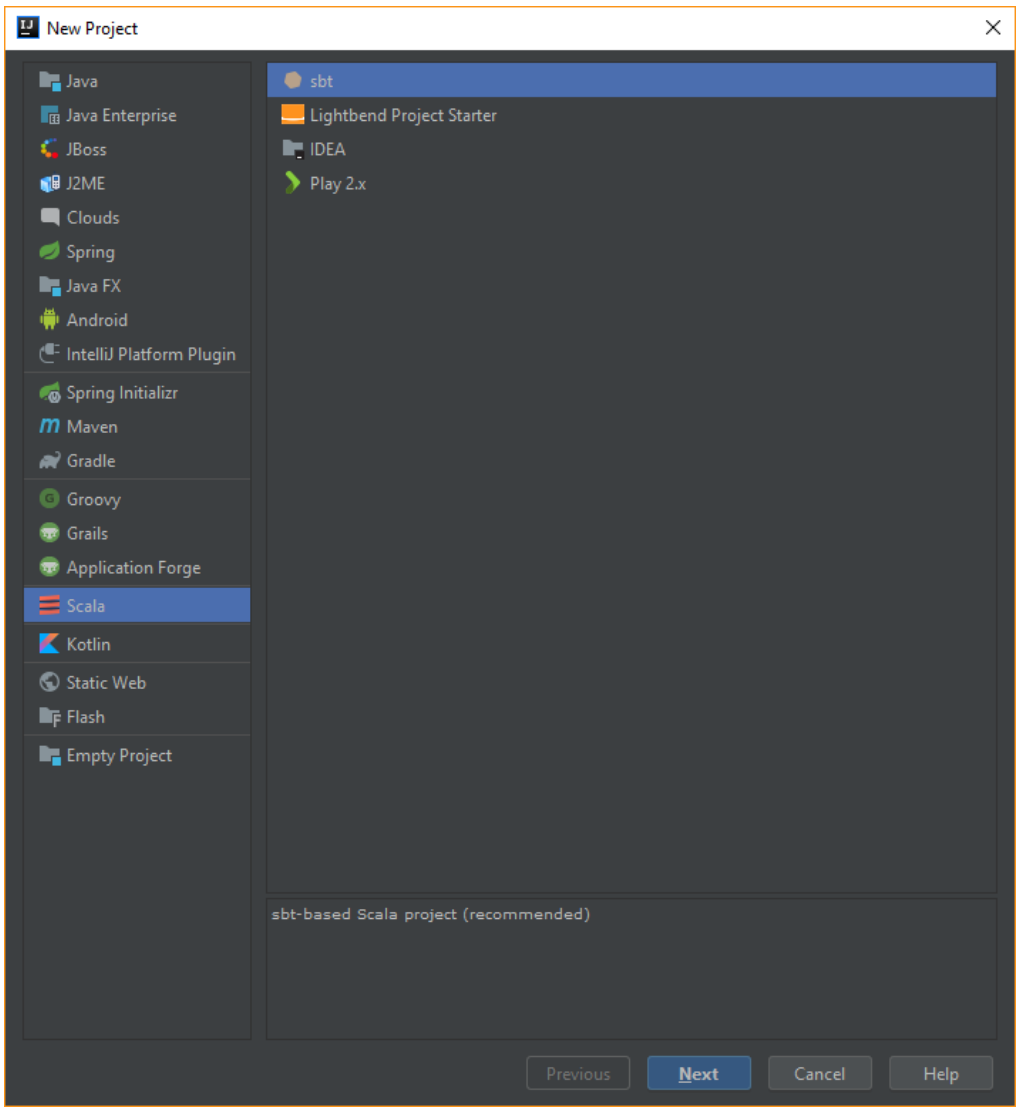

Ilustración 12 - Creación de proyecto Scala con SBT. Fuente: Elaboración propia.

[16] Plugin es un programa que añade una funcionalidad adicional o una nueva característica al software.

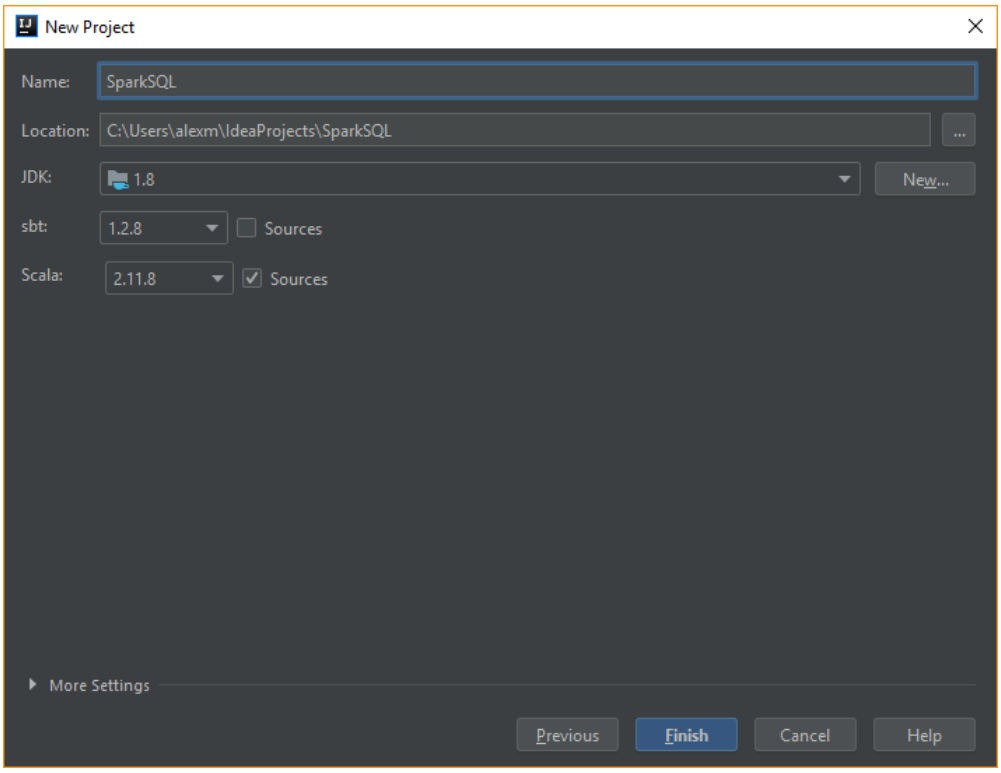

Ilustración 13 - Versión de Scala utilizada. Fuente: Elaboración propia.

2.5.4.B Importar librerías Spark Core y Spark SQL:

Es muy importante crear el proyecto con la versión *Scala* especificada en la ilustración 11, la versión *SBT* no es tan importante. Una vez creado el proyecto, hay que importar las librerías necesarias en el archivo **build.sbt** que se encuentra en la raíz del proyecto.

```
name := "SparkSQL"
version := "0.1"
scalaVersion := "2.11.8"
libraryDependencies ++= Seq(
  "org.apache.spark" %% "spark-core" % "2.3.0",
  "org.apache.spark" %% "spark-sql" % "2.3.0"
)
```

Archivo build.sbt

Mientras estemos editando el archivo, el entorno de desarrollo, estará configurando el entorno, no obstante, de mientras, en la pestaña de *sbt-shell* escribimos *update* y le damos a la tecla *intro*, durante el proceso, el entorno de desarrollo nos mostrará una alerta de importar los cambios del *sbt*, y le damos a importar. Cuando termine, que es probable que tarde unos minutos, ya que está descargando e importando las librerías para poder utilizar *Spark core* y *Spark SQL* en el proyecto *Scala*,

creamos un nuevo fichero en */src/main/scala* (botón derecho > new > *Scala class*) y lo llamamos, **Main.scala,** y ya tendríamos el proyecto listo para empezar a programar.

2.5.4.C Código de programación:

El propósito es poder recuperar datos del *dataset* con una consulta *SQL*, en la cual consigamos los datos asociados a una época, determinada por dos fechas seleccionadas en la aplicación web. Y finalmente imprimir en forma de mapa de calor los datos recuperados con la aplicación *SparkSQL*. [40]

En primer lugar, accedemos al *sparkContext*, mencionado en la explicación de *Spark*, ya que a partir de esa referencia vamos a utilizar todas las librerías.

```
val spark = SparkSession
  .builder()
  .appName("SparkSQL Data Loader")
  .getOrCreate()
val sc = spark.sparkContext
```

Creación de Spark Context

A continuación, filtramos los parámetros de entrada para poder obtener mes, año, día y horas de las fechas recibidas por parámetro, para poder acceder al fichero del mes correspondiente en el HDFS:

```
if (args.size != 3) {
  println()
  println("ERROR: Paso de parámetros inválidos")
  println()
  spark.close()
  System.exit(-1)
}

val datePattern = "(\\d{4})-(\\d{2})-(\\d{2})\\s(\\d{2})[:](\\d{2})".r
val datePattern(startYear, startMonth, startDay, startHour,
startMinute) = args(0)
val datePattern(endYear, endMonth, endDay, endHour, endMinute) =
args(1)

val startDate = args(0)
val endDate = args(1)
val executionId = args(2)
```

Descomposición de las fechas con expresiones regulares

Antes de hacer la consulta SQL, necesitamos cargar el fichero en memoria, para ello accedemos al fichero en el sistema HDFS con el código de la siguiente ilustración:

```
val fs = FileSystem.get(sc.hadoopConfiguration)
var hdfs_path = "/datos.madrid.es_2018/VelocidadMedia"

@transient val filesVelocidadMedia = fs.listStatus(new
Path(hdfs_path))

var selectedFileNameVelocidadMedia = ""

breakable {
  for (file <- filesVelocidadMedia) {
    val filePath = file.getPath.toString.split("/")
    val fileName = filePath(filePath.size - 1)
    val filePattern = "(\\d{2})-(\\d{4})[.]csv".r
    val filePattern(fileMonth, fileYear) = fileName
    if (fileMonth.equals(startMonth)) {
      selectedFileNameVelocidadMedia = hdfs_path + "/" + fileName
      break
    }
  }
}
```

Acceso al fichero HDFS

Finalmente creamos un *dataframe* del archivo recuperado del *HDFS*, la capa explicada en la arquitectura de *SparkSQL*, realizamos consulta *SQL* y guardamos el resultado en un archivo *CSV*. El acceso a ficheros ya sea escritura o lectura, debe ser en *HDFS*, ya que la aplicación corre en el clúster, y el sistema de ficheros para las aplicaciones es el sistema *HDFS*, ya que, si quisiéramos acceder al sistema de ficheros, no sabría encontrarlo ya que dependería del nodo que esté ejecutando esa instrucción.

```
// Sistema de ficheros local "file://ruta/fichero", HDFS (por defecto)
"hdfs://ruta/directorio"
val df1 = spark.read.option("header","true").option("delimiter",
";").csv(selectedFileNameVelocidadMedia)

println()
df1.printSchema()
println()

// Obtenemos la época fecha >= startDate && fecha <= endDate y
agrupamos por id y tipo_elem
df1.createOrReplaceTempView("VelocidadMedia")
val sql1 = "SELECT id, " +
                "tipo_elem, " +
            "ROUND(AVG(vmed), 2) AS vmed " +
            "FROM VelocidadMedia " +
            "WHERE vmed > 0 " +
            "AND (fecha >= '" + startDate + "') " +
            "AND (fecha <= '" + endDate + "') " +
            "GROUP BY id," +
                     "tipo_elem " +
            "ORDER BY id," +
                     "tipo_elem"
val resultSql1 = spark.sql(sql1)
resultSql1.printSchema()
resultSql1.show()
resultSql1.repartition(1).write.option("header","true").csv("/salida/"
+ executionId)

spark.stop()
```

Creación del dataframe y consulta SQL

La consulta consiste en seleccionar el rango de fechas de inicio y fin, por lo tanto selecciona la época establecida, no obstante, cada muestra de cada sensor es cada 15 minutos, esto quiere decir que si la época es de 1 hora, existirá por cada sensor 4 muestras con distintas velocidades medias, pero queremos una muestra por sensor, por tanto agrupamos por id y tipo de elemento, y hacemos una media del valor velocidad media, así logramos tener una media de esas 4 muestras del ejemplo.

2.5.4.D Comando para ejecutar el programa *standalone*:

Para probar la aplicación, debemos ejecutarla, en este punto, sólo es necesario tener el HDFS arrancado, y ejecutar el siguiente comando:

```
$: spark-submit --class SparkSQL --name "SparkSQL"
target/scala-2.11/tfg_scalasparksql_2.11-0.1.jar "2018-05-04
10:20" "2018-05-04 10:50" 12345
```

- **spark-submit** es el binario que inicia la ejecución de un programa *Spark*.
- **--class** es la clase Scala principal

- **--name** nombre del programa durante la ejecución para verlo en los registros log.
- **target/scala-2.11/tfg_scalasparksql_2.11-0.1.jar** ubicación del *jar* [17] a ejecutar, es decir, el programa Big Data.
- A partir de aquí son argumentos que recibirá la aplicación, en este caso la fecha inicio, fecha fin y la ID de la ejecución.

2.5.4.E Comando para ejecutar el programa sobre clúster:

En este caso, el comando es un poco más complejo, ya que hemos visto la configuración de *Resource Dynamic Allocation*, se lo especificamos, además de poder decirle cuántos nodos como mínimo o máximo queremos utilizar durante la ejecución.

```
 $: spark-submit --class SparkSQL --master yarn --deploy-mode
client --conf spark.dynamicAllocation.enabled=true --conf
spark.shuffle.service.enabled=true --conf
spark.dynamicAllocation.minExecutors=7 --conf
spark.driver.cores=1 --conf spark.driver.memory=6g --conf
spark.executor.memory=6g
/home/alopez/proyecto/tfg_scalasparksql_2.11-0.1.jar "2018-05-
04 10:20" "2018-05-04 10:50" 12345
```

- **--master** para especificar el driver gestor del clúster, por defecto será *yarn* pero lo especificamos para que no haya confusiones.
- **--deploy-mode** en este caso *client*, ya que, si es de tipo *cluster*, el driver lo ejecutará uno de los nodos, y al ser *client*, será el gestor el que ejecuta el comando, y en este caso se hará en el maestro.
- **--conf** son parámetros extras para la ejecución del programa, en este caso para el uso de núcleos, memoria ram por esclavo y mínimo de esclavos.
- **/home/alopez/proyecto/tfg_scalasparksql_2.11-0.1.jar** ubicación del *jar* a ejecutar, es decir, el programa Big Data.
- A partir de aquí son argumentos que recibirá la aplicación, en este caso la fecha inicio, fecha fin y la ID de la ejecución.

[17] Jar es el empaquetado que crea SBT con la aplicación Scala.

2.5.5 Desarrollo aplicación web

2.5.5.A Configuración del Ruby Mine:

Este paso es sencillo, ya que únicamente debes abrir el proyecto creado con el comando **rails new RailsSparkSQL**. Es decir, si creamos el proyecto en */home/cicei/Documentos*, con Ruby Mine abrimos el directorio */home/cicei/Documentos/RailsSparkSQL*, y ya tendríamos todo adecuado para utilizar el entorno de desarrollo. Es probable que el entorno pida hacer un *import* del **gemfile**[18].

2.5.5.B Diagrama de clases de la aplicación:

Contamos con un diagrama sencillo ya que cada clase es el modelado de cada *dataset*, ya que viene de forma estructurada, la clase **Speed** pertenece al *dataset* del histórico de datos con las velocidades medias, y la clase **Point** al *dataset* con la ubicación de los sensores.

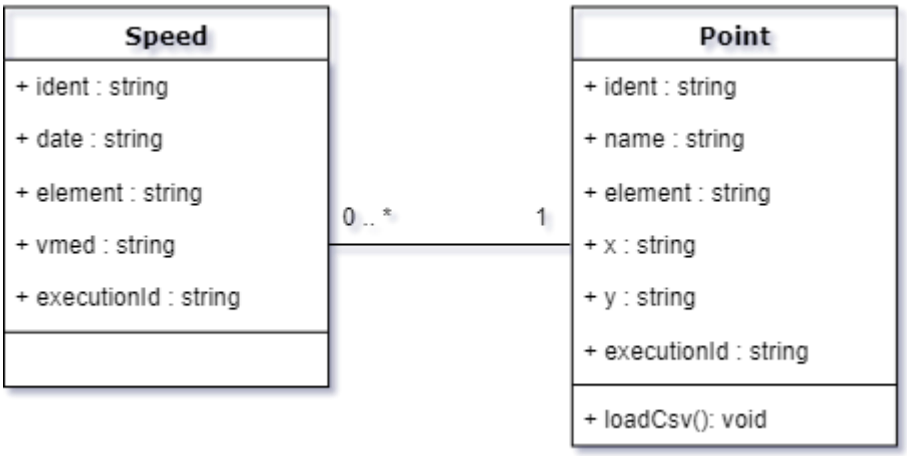

Ilustración 14 - Diagrama de clases UML aplicación Ruby on Rails. Fuente: elaboración propia.

2.5.5.C Comandos básicos de Ruby on Rails:

Basándonos en el diagrama de clases anterior, empezamos a crear la aplicación. Es muy importante tener un diagrama de clases inicial para empezar a desarrollar en *Ruby on Rails*. Estos comandos pueden ser ejecutados en la terminal de *Ruby Mine* una vez abierto el proyecto.

```
$: rails generate model Point ident:string name:string
element:string x:string y:string executionId:string
```

[18] Fichero de gemas de la aplicación *rails*, ahí queda registrado las librerías externas que vamos a utilizar.

$: rails generate model Speed ident:string date:string element:string vmed:string executionId:string
$: rails generate controller points insert loadCsv testDirectory
$: rails generate controller epochs index map map_test dfs_test

- **generate model**, creamos una clase de modelo con nombre y seguido de sus atributos y tipo de dato.

- **generate controller**, genera la clase controlador y la vista vacía asociada a dicha acción, en este caso creamos un controlador con varias acciones y vistas vacías, para dejar los ficheros listos a ser programados.

A continuación, esto genera unas migraciones que deben aplicarse para que se actualice las tablas de la base de datos con el modelo de la aplicación especificado previamente.

$: rails db:create
$: rails db:migrate

En nuestro caso, vamos a especificar la vista principal, la acción *index* del controlador *epochs*. Este controlador será el encargado de moverse entre las dos vistas que existen, el especificar la época a buscar, y mostrar el mapa de calor. La línea **root 'epoches#index'** es la que especifica que será la vista principal.

```
Rails.application.routes.draw do
  # For details on the DSL available within this file, see
http://guides.rubyonrails.org/routing.html
  root 'epoches#index'
  get '/epoches/index', to: 'epoches#index'
  get '/pointns/loadCsv', to: 'pointns#loadCsv'
  get '/pointns/testDirectory', to: 'pointns#testDirectory'
  post '/epoches/map', to: 'epoches#map'
  get '/epoches/map_test', to: 'epoches#map_test'
  get '/epoches/dfs_test', to: 'epoches#dfs_test'
end
```

Archivo Rails.root/conf/routes.rb

Y para probar que todo esté correcto, encender el servidor y acceder a través de http://localhost:3000

2.5.5.D Código de programación:

Ya hemos explicado previamente el esquema del modelo y controladores, vamos a continuar con el uso que va a tener cada acción y vista. [41]

- **PointsController** (*Rails.root/app/controllers/points.rb*) [49]
    - **loadCsv:** Encargado de cargar en la base de datos el *dataset* de puntos de medida.
- **EpochsController** (*Rails.root/app/controllers/epochs.rb*) [48]
    - **index**: Configura vista principal donde se introduce la época a buscar.
    - **map**: Recibe la época seleccionada y ejecuta un proceso de llamada a la aplicación *Big Data*, que desencadena una serie de ejecuciones para sincronizar los datos generados por la aplicación *Big Data* sean accesibles por *Ruby on Rails*.
- **ViewLayout** (*Rails.root/app/views/layouts/application.html.erb*) [45]
    - Esquema genérico a todas las vistas, en este caso para importar los ficheros de *javascript* utilizados y la cabecera de la página.
- **EpochesIndexView** (*Rails.root/app/views/epochs/index.html.erb*) [48]
    - Vista preparada para introducir fecha inicio y fecha fin que será la época establecida, botón que ejecutará la acción *map* del controlador *epochs*.
- **EpochesMapView** (*Rails.root/app/views/epochs/map.html.erb*) [47]
    - Vista que dibuja una leyenda del mapa de calor, programado en *javascript*, además del mapa de calor utilizando la librería de *Javascript Google Maps V3*.

2.5.5.E Vista previa de la aplicación:

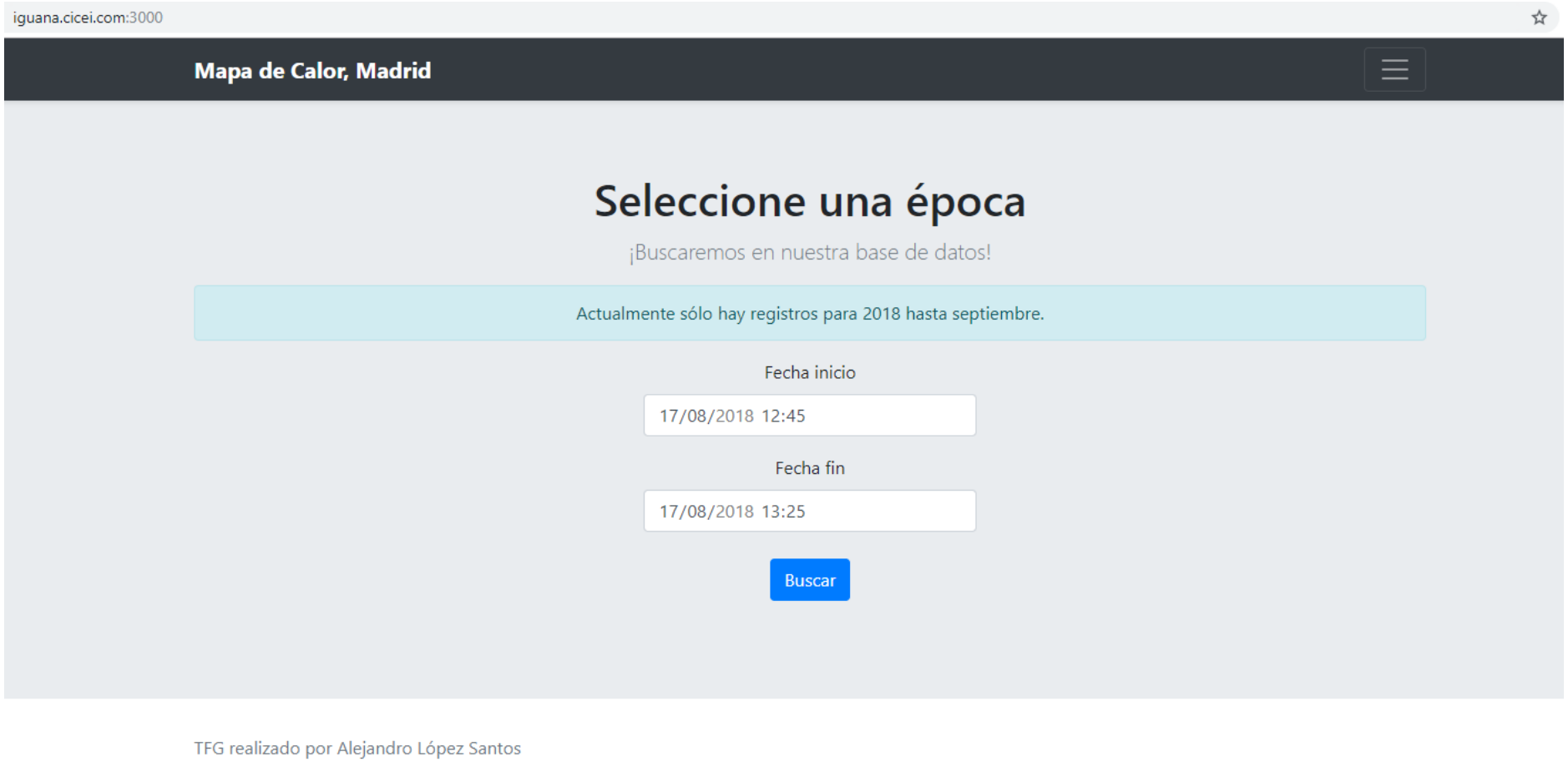

Ilustración 15 - Página inicial. Fuente: Elaboración propia.

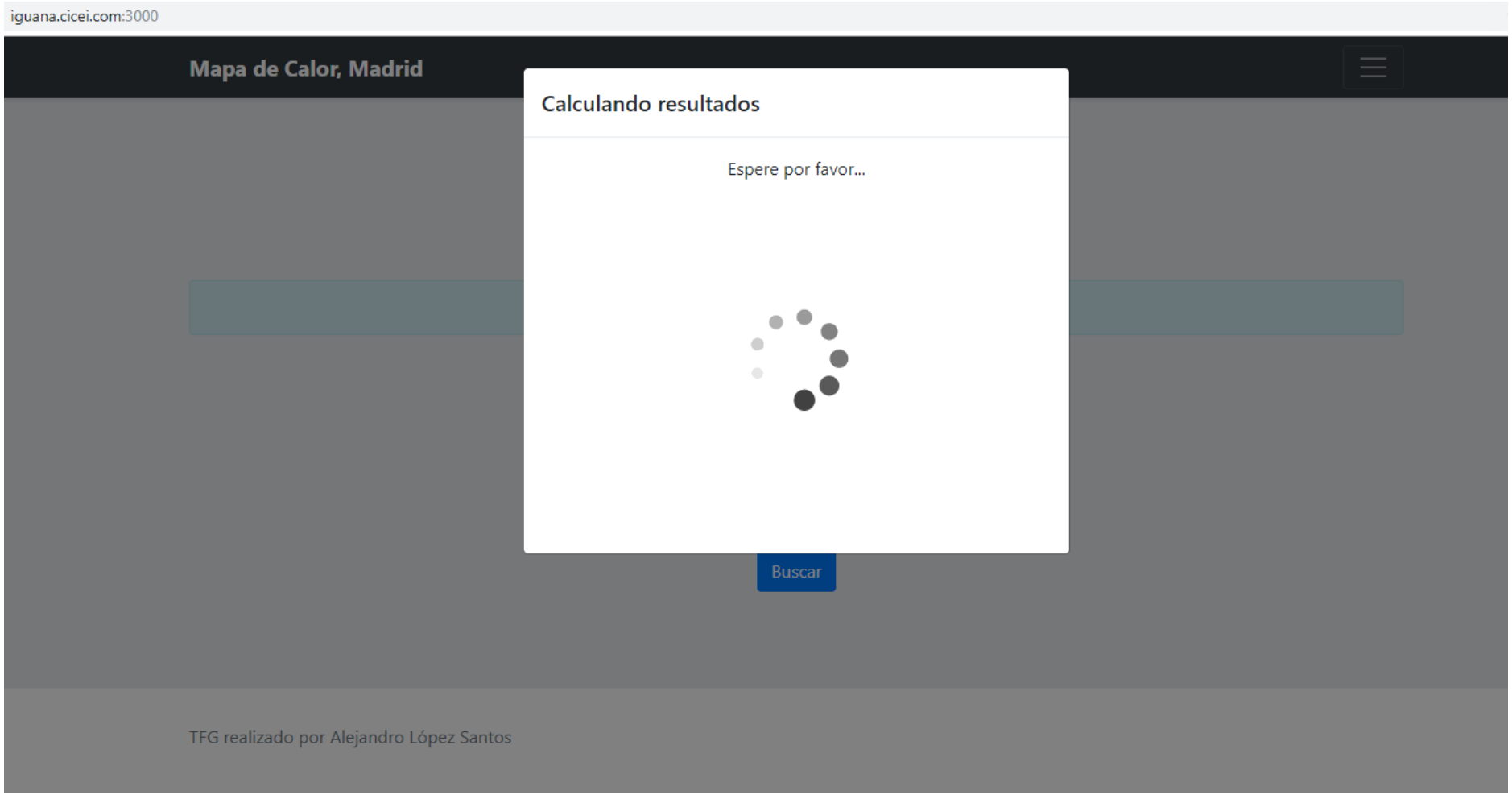

Ilustración 16 - Acción de buscar. Fuente: Elaboración propia.

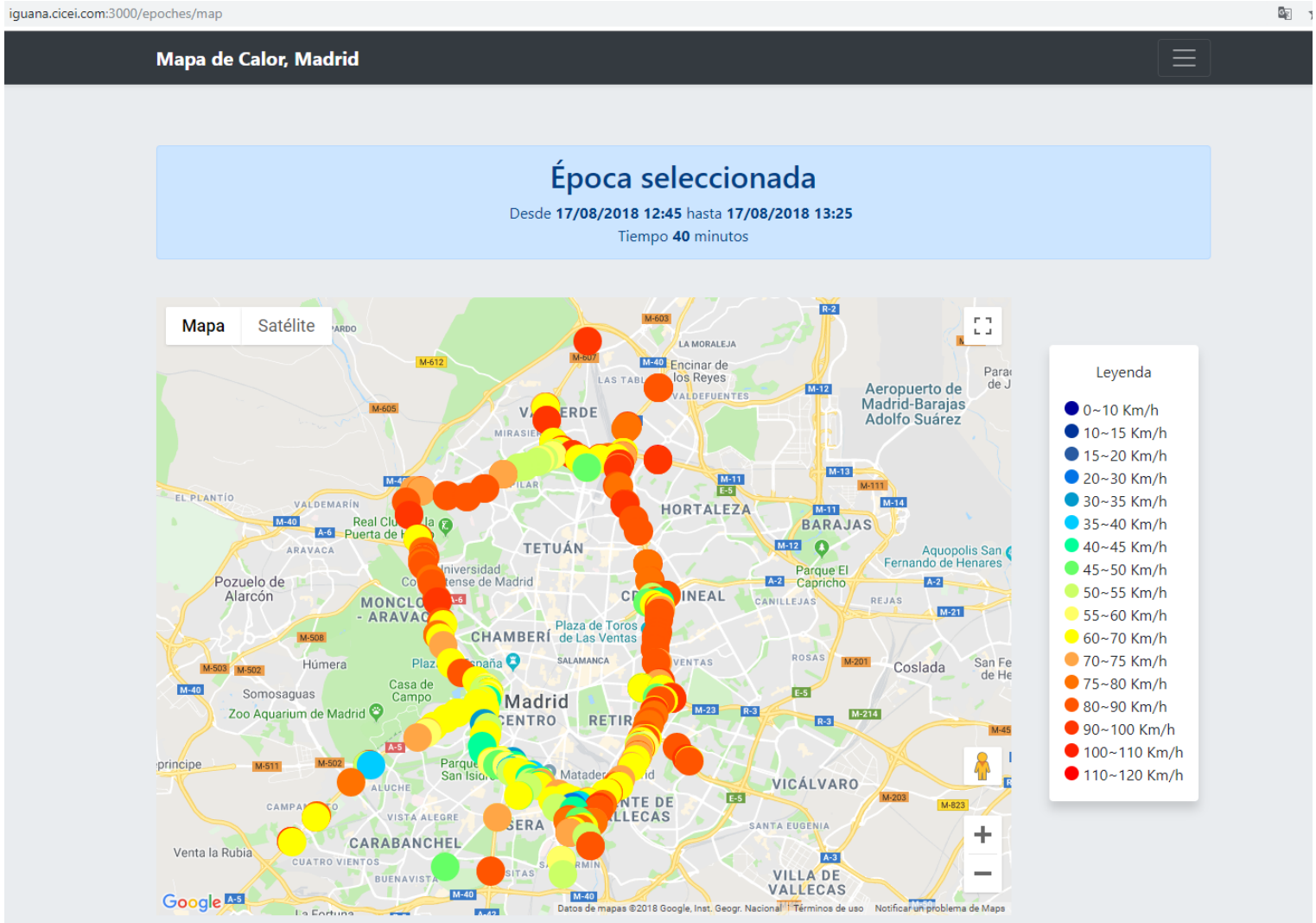

Ilustración 17 - Resultado de la búsqueda. Fuente: Elaboración propia.

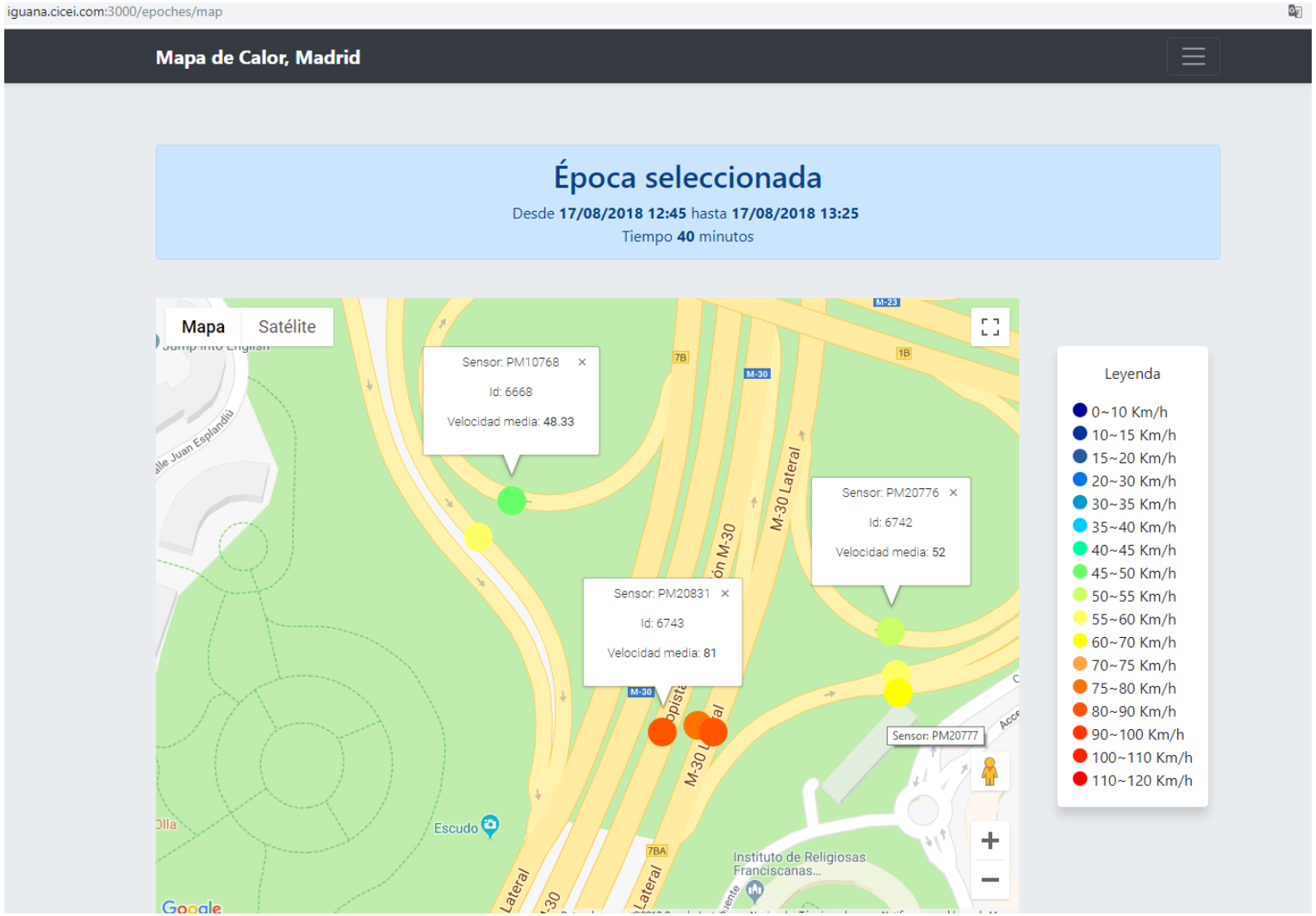

Ilustración 18 - Interacción con el mapa. Fuente: Elaboración propia.

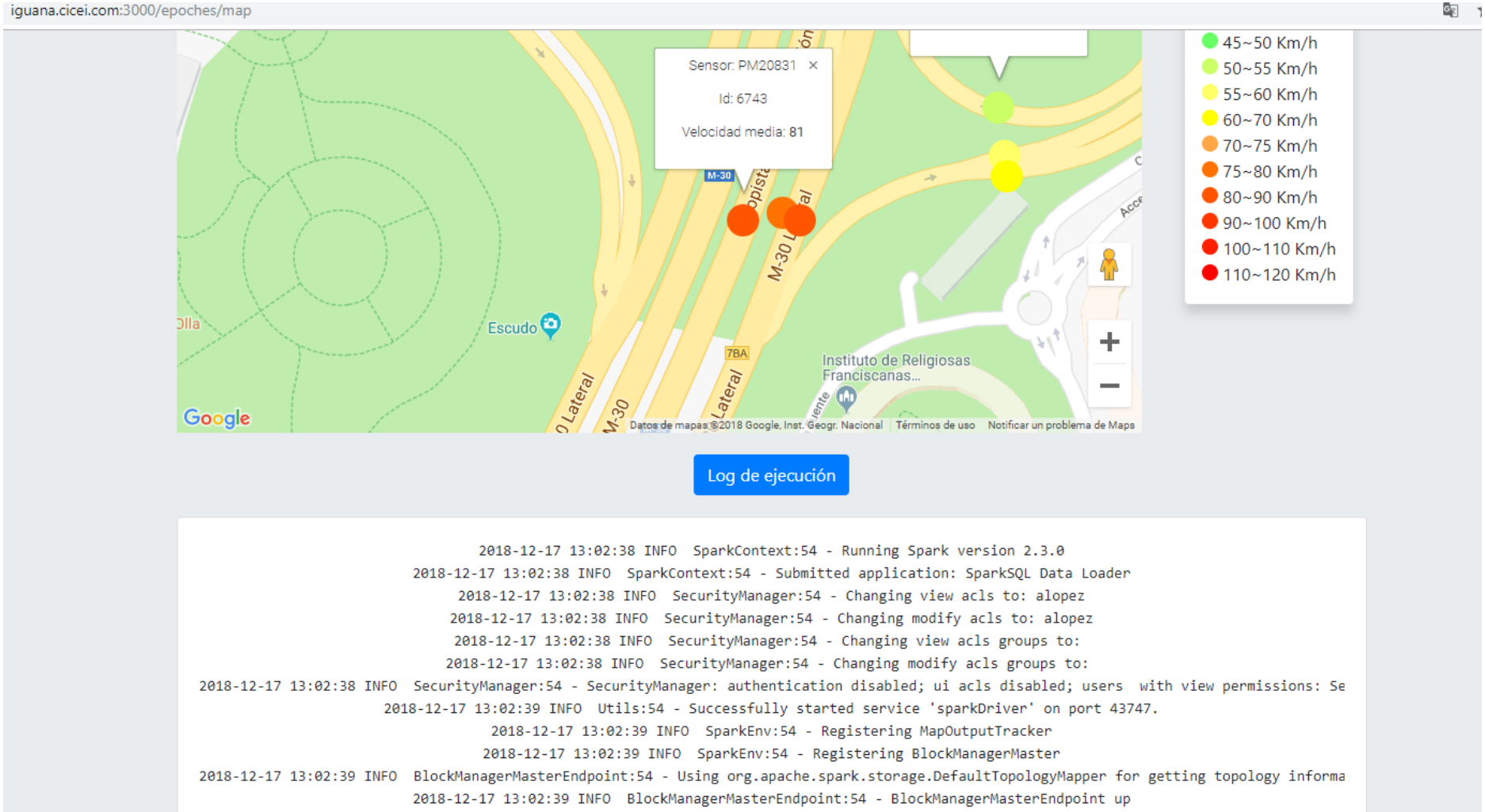

Ilustración 19 - Log de ejecución. Fuente: Elaboración propia.

iguana.cicei.com:3000/epoches/map

```
2018-12-17 13:02:58 INFO  ContextHandler:781 - Started o.s.j.s.ServletContextHandler@1b37fbec{/metrics/json,null,AVAILABLE,@Spark}
2018-12-17 13:02:58 INFO  YarnSchedulerBackend$YarnSchedulerEndpoint:54 - ApplicationMaster registered as NettyRpcEndpointRef(spark-client:
2018-12-17 13:02:58 INFO  EventLoggingListener:54 - Logging events to hdfs://aton.darwin.cicei.com:9000/sparkEventLog/application_154479391
2018-12-17 13:02:58 WARN  Utils:66 - spark.executor.instances less than spark.dynamicAllocation.minExecutors is invalid, ignoring its setti
2018-12-17 13:02:58 INFO  Utils:54 - Using initial executors = 7, max of spark.dynamicAllocation.initialExecutors, spark.dynamicAllocation.
2018-12-17 13:03:05 INFO  YarnSchedulerBackend$YarnDriverEndpoint:54 - Registered executor NettyRpcEndpointRef(spark-client://Executor) (17
                 2018-12-17 13:03:05 INFO  ExecutorAllocationManager:54 - New executor 2 has registered (new total is 1)
2018-12-17 13:03:05 INFO  YarnSchedulerBackend$YarnDriverEndpoint:54 - Registered executor NettyRpcEndpointRef(spark-client://Executor) (17
                 2018-12-17 13:03:05 INFO  ExecutorAllocationManager:54 - New executor 1 has registered (new total is 2)
2018-12-17 13:03:05 INFO  BlockManagerMasterEndpoint:54 - Registering block manager osiris.darwin.cicei.com:38465 with 3.0 GB RAM, BlockMan
2018-12-17 13:03:05 INFO  BlockManagerMasterEndpoint:54 - Registering block manager isis.darwin.cicei.com:45841 with 3.0 GB RAM, BlockManag
2018-12-17 13:03:05 INFO  YarnSchedulerBackend$YarnDriverEndpoint:54 - Registered executor NettyRpcEndpointRef(spark-client://Executor) (17
                 2018-12-17 13:03:05 INFO  ExecutorAllocationManager:54 - New executor 3 has registered (new total is 3)
2018-12-17 13:03:05 INFO  BlockManagerMasterEndpoint:54 - Registering block manager hathor.darwin.cicei.com:39987 with 3.0 GB RAM, BlockMan
2018-12-17 13:03:09 INFO  YarnSchedulerBackend$YarnDriverEndpoint:54 - Registered executor NettyRpcEndpointRef(spark-client://Executor) (17
                 2018-12-17 13:03:09 INFO  ExecutorAllocationManager:54 - New executor 4 has registered (new total is 4)
2018-12-17 13:03:09 INFO  BlockManagerMasterEndpoint:54 - Registering block manager anubis.darwin.cicei.com:37935 with 3.0 GB RAM, BlockMan
2018-12-17 13:03:10 INFO  YarnClientSchedulerBackend:54 - SchedulerBackend is ready for scheduling beginning after waiting maxRegisteredRes

Start Date: 2018-08-17T12:45

End Date: 2018-08-17T13:25

Execution ID: edc68466-b18e-4fc7-9f2a-7cf9e815bec7

VelocidadMedia: /datos.madrid.es_2018/VelocidadMedia/08-2018.csv
```

Ilustración 20 - Log de ejecución. Fuente: Elaboración propia.

2.5.6 Informe Speed-Up

2.5.6.A Introducción:

En este documento vamos a detallar el rendimiento del clúster frente al portátil donde he desarrollado las pruebas, además del rendimiento del servidor en modo clúster y modo *single node*.

2.5.6.B Preparación de los datos:

Ejecutamos el código varias veces con un *bash script*, que ejecutará el trabajo *Spark* varias veces:

```
#!/bin/bash
echo "Realizando SpeedUp Cluster Mode"

for i in {1..10}
do
    echo "Ejecucion numero $i"

    spark-submit --class SparkSQL --master yarn --deploy-mode client -
-conf spark.dynamicAllocation.enabled=true --conf spark.shuffle.servic
e.enabled=true --conf spark.dynamicAllocation.minExecutors=7 --conf sp
ark.driver.cores=1 --conf spark.driver.memory=6g --conf spark.executo
r.memory=6g /home/alopez/proyecto/tfg_scalasparksql_2.11-0.1.jar "2018
-05-04 10:20" "2018-05-04 10:50" cluster_$i > cluster_mode_execution_
$i.txt
done

echo "Realizando SpeedUp Single Node Mode"
for i in {1..10}
do
    echo "Ejecucion numero $i"

    spark-submit --class SparkSQL --master yarn --deploy-mode client -
-conf spark.dynamicAllocation.enabled=true --conf spark.shuffle.servic
e.enabled=true --conf spark.dynamicAllocation.maxExecutors=1 --conf sp
ark.driver.cores=1 --conf spark.driver.memory=6g --conf spark.executo
r.memory=6g /home/alopez/proyecto/tfg_scalasparksql_2.11-0.1.jar "2018
-05-04 10:20" "2018-05-04 10:50" single_$i > single_node_$i.txt
done
```

Al generar los **.txt** por cada ejecución, y hacer más fácil la obtención de los datos para calcular el tiempo de ejecución de cada iteración, generamos unos archivos más obteniendo el tiempo inicial y obteniendo el tiempo final de cada ejecución:

```
head -n 1 cluster_mode_execution_* -q | cut -c1-19 > cluster_start.txt
tail -n 1 cluster_mode_execution_* -q | cut -c1-19 > cluster_end.txt
head -n 1 single_node_* -q | cut -c1-19 > single_start.txt
tail -n 1 single_node_* -q | cut -c1-19 > single_end.txt
```

A pesar de que sea texto plano, tendrá una única columna con las fechas y la podremos considerar un *CSV* de una sola columna, por tanto, lo guardamos con extensión *CSV*. A continuación, con el siguiente código en **R** recogemos los cuatro ficheros generados y creamos un *DataFrame*, para un fácil manejo de los datos en **R**.

```
# Prepare TXT Output to DataFrames
startDateSingle <- read.csv(
  file="data/single_start.csv",
  header=F
)

endDateSingle <- read.csv(
  file="data/single_end.csv",
  header=F
)

# Single Node data execution samples
clusterSingleNodeData <- data.frame(
  Execution_Start = startDateSingle$V1,
  Execution_End = endDateSingle$V1
)

startDateCluster <- read.csv(
  file="data/cluster_start.csv",
  header=F
)

endDateCluster <- read.csv(
  file="data/cluster_end.csv",
  header=F
)

# Cluster data execution samples
clusterData <- data.frame(
  Execution_Start = startDateCluster$V1,
  Execution_End = endDateCluster$V1
)
```

```
#Load laptop Data
laptopData <- read.csv(
  file="data/laptopData.csv",
  header=T,
  sep=";",
  stringsAsFactors = T
)

# Calculate execution times in seconds, every row will be seconds betw
een the two dates
laptopData = as.integer(difftime(
    as.POSIXlt(laptopData$Execution_End, format="%Y-%m-%d %H:%M:%S"),
    as.POSIXlt(laptopData$Execution_Start, format="%Y-%m-%d %H:%M:%S
"),
    units="secs")
)

clusterSingleNodeData = as.integer(difftime(
    as.POSIXlt(clusterSingleNodeData$Execution_End, format="%Y-%m-%d %
H:%M:%S"),
    as.POSIXlt(clusterSingleNodeData$Execution_Start, format="%Y-%m-%d
 %H:%M:%S"),
    units="secs")
)

clusterData = as.integer(difftime(
  as.POSIXlt(clusterData$Execution_End, format="%Y-%m-%d %H:%M:%S"),
  as.POSIXlt(clusterData$Execution_Start, format="%Y-%m-%d %H:%M:%S"),
  units="secs")
)
```

2.5.6.C Fórmula Speed-Up:

Calculamos el *Speed-Up* con la fórmula descrita a continuación, la ejecución antigua pertenece a *single node* y clúster frente al portátil, además en la comparativa de *single node* y clúster, la ejecución antigua es *single node*:

$$\text{Speed-Up} = \frac{Tiempo\ de\ ejecución\ parela}{Tiempo\ de\ ejecución\ en\ serie}$$ [9]

```
# Get the mean execution time, given by 10 execution time samples
laptopDataMean = mean(laptopData)
clusterSingleNodeDataMean = mean(clusterSingleNodeData)
clusterDataMean = mean(clusterData)
```

```
dataForPlot <- c(
  clusterSingleNodeDataMean / laptopDataMean,
  clusterDataMean / laptopDataMean,
  clusterSingleNodeDataMean / clusterDataMean
)
```

2.5.6.D Visualización de los datos:

```
plot = barplot (
  dataForPlot,
  main = "Speedup Test",
  xlab = "Test type",
  ylab = "Performance %",
  col = c("darkblue", "red", "darkgreen"),
  names.arg = c("Single Node / Laptop", "Cluster / Laptop", "Single No
de / Cluster"),
  ylim = c(0,5)
)
text(plot, dataForPlot + 0.1, round(dataForPlot, digits=2), pos = 3, f
ont = 2)
```

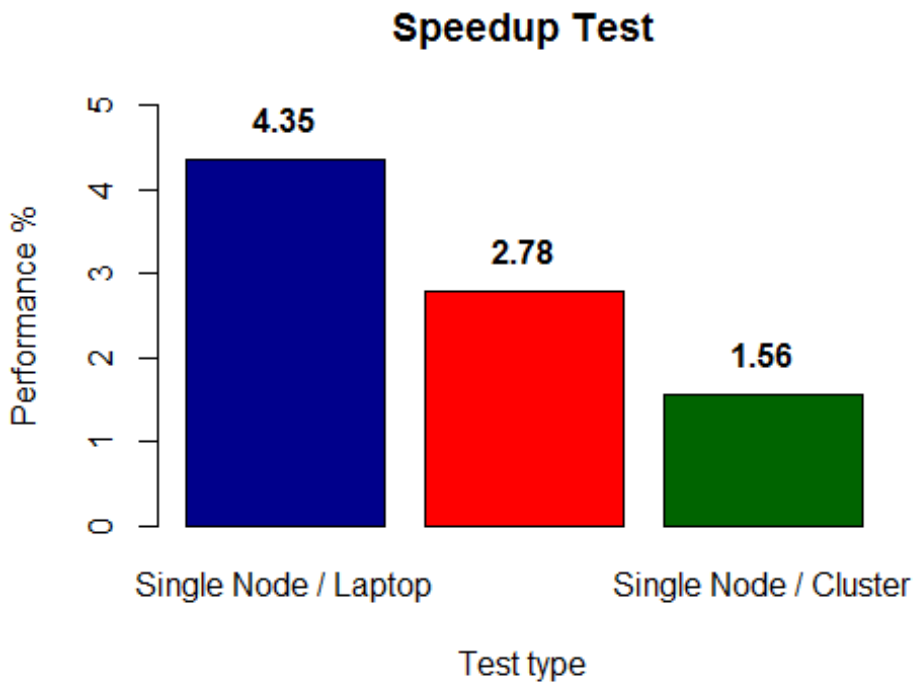

Ilustración 21 - Gráfica speedup. Fuente: elaboración propia.

2.5.6.E Análisis de los datos:

Características del clúster –

Procesador: 2x AMD Opteron Processor 246 1995mhz 1 núcleo y 1 hilo
Memoria: 7Gb. RAM
Disco: 2Tb. HDD - Sata 1
Total de 8 nodos.

Características del portátil –

Procesador: Intel i5-5200U 2,2ghz 2 núcleos 4 hilos
Memoria: 8gb ram
Disco: 128gb SSD
Gráfica: Nvidia 765m GTX

A simple vista se aprecia que el portátil es más eficiente para las tareas que ejecuta nuestro programa *Spark*, el motivo es sencillo, analizando el hardware de cada sistema, vemos una notable diferencia en el acceso a disco, no sólo la mejoría que puede llegar a proporcionar *Sata 3* frente a *Sata 1*, sino también el usar un disco de estado sólido frente a un disco duro. En nuestra aplicación *Ruby on Rails* guardamos los resultados del programa *Spark* en la base de datos para poderlos manejar a posteriori, y comparando el tiempo que tarda en guardar cada registro se observa que el **portátil** tiene una latencia de **10 milisegundos** y cualquier **nodo del clúster** entre **70-300 milisegundos**.

2.5.6.F Conclusión Speed-Up:

Se nota una mejoría de **1.56x** ejecutando el mismo programa *Spark* de *single node* a clúster. En un trabajo relativamente simple como el mío ya nota una mejoría, por tanto, en trabajos más complicados se verá reflejado una mejoría más significativa en el mismo clúster.

```
   (0.1ms)  begin transaction
  Speed Create (0.6ms)  INSERT INTO "speeds" ("ident", "element", "vmed", "creat
ed_at", "updated_at", "executionId") VALUES (?, ?, ?, ?, ?, ?)  [["ident", "7128
"], ["element", "M30"], ["vmed", "73.0"], ["created_at", "2018-12-17 12:57:11.96
5818"], ["updated_at", "2018-12-17 12:57:11.965818"], ["executionId", "8b63f6c9-
b48f-4f33-ab9a-c7590cc714cb"]]
   (222.0ms)  commit transaction
   (0.1ms)  begin transaction
  Speed Create (0.7ms)  INSERT INTO "speeds" ("ident", "element", "vmed", "creat
ed_at", "updated_at", "executionId") VALUES (?, ?, ?, ?, ?, ?)  [["ident", "7129
"], ["element", "M30"], ["vmed", "73.0"], ["created_at", "2018-12-17 12:57:12.19
2408"], ["updated_at", "2018-12-17 12:57:12.192408"], ["executionId", "8b63f6c9-
b48f-4f33-ab9a-c7590cc714cb"]]
   (236.0ms)  commit transaction
   (0.1ms)  begin transaction
  Speed Create (0.7ms)  INSERT INTO "speeds" ("ident", "element", "vmed", "creat
ed_at", "updated_at", "executionId") VALUES (?, ?, ?, ?, ?, ?)  [["ident", "7130
"], ["element", "M30"], ["vmed", "62.0"], ["created_at", "2018-12-17 12:57:12.43
2802"], ["updated_at", "2018-12-17 12:57:12.432802"], ["executionId", "8b63f6c9-
b48f-4f33-ab9a-c7590cc714cb"]]
   (195.9ms)  commit transaction
```

Ilustración 22 - Tiempo de latencia guardado de datos en clúster. Fuente: Elaboración propia.

3. Conclusiones

3.1 Conclusión

En primer lugar, hay que destacar que es una tecnología que lleva pocos años en desarrollo y no tiene una gran comunidad, esto es contraproducente porque siempre existen errores únicos o bugs [19] que no están documentados y/o que sólo ocurre al que está desplegando la aplicación, por tanto, no se podrá solicitar ayuda de forma rápida y habrá que investigar profundamente la documentación y gastar tiempo de investigación para, lo que posiblemente sea, un error tonto o simplemente un bug. Esto ha llevado a que el proyecto se haya ido realizando de manera más lenta que un trabajo final de grado con tecnologías más documentadas, no obstante, la recompensa final ha sido bastante gratificante.

Por otra parte, para este tipo de trabajos hace falta una infraestructura que no hemos podido obtener, es decir, un clúster con mejores características hardware, ya se ha visto en la prueba de *speed-up* que mi portátil ha dado mejor rendimiento que el clúster con varios nodos, por la evidente falta de un clúster con mejor hardware. Lo cual es una espinita de poder probar este entorno en condiciones ideales. Agradezco profundamente la posibilidad brindada por el tutor el poder trabajar con un clúster en un entorno real, pero el hardware no ha sido suficiente.

[19] Bug es un fallo informático debido a un error de código de programación en la aplicación que se está usando o programando.

La tecnología utilizada la veo con bastante potencial, y bastante útil, ya que proporciona, como se ha podido demostrar, un uso y manejo de datos de forma bastante cómoda y eficiente, es cierto que entrar en contacto y montar la infraestructura puede ser algo confuso debido a todas las posibilidades de configuración que proporciona la tecnología, pero una vez que uno ya está acostumbrado nota la sensación de poder hacer grandes cosas. Quiero decir, que me gustaría profundamente continuar con esta rama de la ingeniería informática y seguir formándome en este entorno.

Para concluir, este trabajo ha hecho dar lo mejor de mí, sobre todo por no haber estudiado esta rama en el grado de ingeniería informática, a excepción de la creación de un clúster, que, a pesar de no utilizar la misma tecnología, he tenido una visión bastante clara de cómo configurar esta parte. Y he descubierto que investigar tecnologías de la información y aprender campo de la computación es algo que realmente me apasiona.

4. Fuentes de información

4.1 Definiciones:

1	Concepto macrodatos	https://es.wikipedia.org/wiki/Macrodatos

También llamado Big Data, término que hace referencia al concepto relativo a conjuntos de datos tan grandes y complejos como para que hagan falta aplicaciones informáticas no tradicionales.

2	Apache Hadoop Yarn	http://hadoop.apache.org/docs/stable/hadoop-yarn/hadoop-yarn-site/YARN.html

Motor del clúster Hadoop que se encarga de gestionar los recursos del nodo y los procesos a ejecutar en cada nodo.

3	Explicación NodeManager	http://hadoop.apache.org/docs/stable/hadoop-yarn/hadoop-yarn-site/NodeManager.html

NodeManager es una parte del motor Yarn que es el responsable de ejecutar y administrar los contenedores de un nodo del clúster.

4	Componentes del clúster Spark	https://spark.apache.org/docs/latest/cluster-overview.html

Esquemático del funcionamiento de un clúster utilizando como motor, Spark.

5	Explicación de Machine Learning	https://es.wikipedia.org/wiki/Aprendizaje_autom%C3%A1tico

También conocido como aprendizaje automático, subcampo de las ciencias de la computación y una rama de la inteligencia artificial.

6	Explicación HQL	https://docs.jboss.org/hibernate/orm/3.5/reference/es-ES/html/queryhql.html

Lenguaje de consulta de datos similar a SQL, pero totalmente orientado a objetos.

7	Explicación Framework	https://es.wikipedia.org/wiki/Framework

Conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular, que sirve como referencia, para enfrentar y resolver nuevos problemas de índole similar.

8	Explicación de R	https://es.wikipedia.org/wiki/R_(lenguaje_de_programaci%C3%B3n)

Lenguaje de programación muy conocido en el entorno del análisis estadístico, por tanto, utilizado en el campo de la Ciencia de Datos.

9	Speed-Up	https://es.wikipedia.org/wiki/Speedup

Proceso realizado para analizar y mejorar el rendimiento de un sistema que procesa un problema determinado.

10	Explicación de Git	https://git-scm.com/

Software gestor de proyectos de programación, permite guardar versiones del código.

4.2 Documentación

11	Incremento de ingresos en el mercado Big Data	https://www.forbes.com/sites/louiscolumbus/2018/05/23/10-charts-that-will-change-your-perspective-of-big-datas-growth/#162afa929268
12	Cada vez más empresas adoptan el modelo Big Data	https://www.forbes.com/sites/louiscolumbus/2017/12/24/53-of-companies-are-adopting-big-data-analytics/#17f2aa5339a1
13	Curso Big Data, Hadoop y Spark en Udemy	https://www.udemy.com/monta-un-cluster-hadoop-big-data-desde-cero/
14	Wikibon	https://en.wikipedia.org/wiki/Wikibon
15	Página de Hadoop	https://hadoop.apache.org/
16	Documentación HDFS	http://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs-rbf/HDFSRouterFederation.html

17	Página de Apache Spark	https://spark.apache.org/
18	Documentación de SparkSQL	https://www.edureka.co/blog/spark-sql-tutorial/
19	Ruby Mine	https://www.jetbrains.com/ruby/
20	Scala programming language	https://www.scala-lang.org/
21	SBT	https://www.scala-sbt.org/
22	IntelliJ	https://www.jetbrains.com/idea/
23	RStudio	https://www.rstudio.com/
24	Madrid Open Data	https://datos.madrid.es/portal/site/egob/
25	Catálogo de datos tráfico histórico	https://goo.gl/EF5uEX
26	Catálogo de datos ubicación de puntos de medida	https://goo.gl/xpAmJz
27	Get Fedora	https://getfedora.org/en/workstation/download/
28	Get Ubuntu Server	https://www.ubuntu.com/download/server
29	Get Apache Hadoop	https://archive.apache.org/dist/hadoop/common/hadoop-3.1.0/
30	Get Apache Spark	https://spark.apache.org/downloads.html
31	Get Ruby on Rails on Fedora	https://developer.fedoraproject.org/tech/languages/ruby/ror-installation.html
32	Get Ruby on Rails on Ubuntu	https://gorails.com/setup/ubuntu/17.10
33	Install Java 8 JDK, Linux Fedora x64	https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html
34	¿Por qué Java 8 con Hadoop?	https://hadoop.apache.org/docs/r3.0.0/
35	JetBrains Toolbox App	https://www.jetbrains.com/toolbox/app/

36	Cuenta de estudiante Jet Brains	https://account.jetbrains.com/login
37	Configurar interfaces de red Ubuntu	https://www.swiftstack.com/docs/install/configure_networking.html
38	Configuración: Dynamic Resource Allocation	https://spark.apache.org/docs/latest/job-scheduling.html#configuration-and-setup
39	Documentación: Dynamic Resource Allocation	https://spark.apache.org/docs/latest/running-on-yarn.html#configuring-the-external-shuffle-service
40	Código Scala de la aplicación SparkSQL	https://goo.gl/RQJ2BU
41	Código Ruby on Rails	https://goo.gl/CE4Lv7
42	Repositorio GitHub del Proyecto	https://github.com/rokimoki/TFG_SparkSQL
43	Documentación Google Maps Api v3	https://developers.google.com/maps/documentation/javascript/tutorial
44	Código Javascript	https://goo.gl/4XwrUi
45	Layout aplicación rails	https://goo.gl/G4hRYK
46	Vista principal aplicación rails	https://goo.gl/eHA5jP
47	Vista de mapa de calor de la aplicación rails	https://goo.gl/AExhhg
48	Controlador epochs de aplicación rails	https://goo.gl/vb1bUx
49	Controlador points de aplicación rails	https://goo.gl/KQHD5d