UNIVERSIDAD DE EXTREMADURA

Escuela Politécnica

Grado en Ingeniería Informática en Ingeniería del Software

Trabajo Fin de Grado

Meteo Big Data: Integración de datos espaciales meteorológicos para Data Mining utilizando almacenamiento NoSQL MongoDB

Luis Fernando Melón Pérez

Febrero, 2016

UNIVERSIDAD DE EXTREMADURA

Escuela Politécnica

Grado en Ingeniería Informática en Ingeniería del Software

Trabajo Fin de Grado

Meteo Big Data: Integración de datos espaciales meteorológicos para Data Mining utilizando almacenamiento NoSQL MongoDB

Autor: Luis Fernando Melón Pérez
Fdo.:

Director: Félix Rodríguez Rodríguez
Fdo.:

Codirectora: Mª Luisa Durán Martín-Merás
Fdo.:

Tribunal Calificador

Presidente: Miguel Ángel Pérez Toledano
Fdo.:

Secretario: Miryam J. Salas Sánchez
Fdo.:

Vocal: José Moreno del Pozo
Fdo.:

CALIFICACIÓN:
FECHA:

Agradecimientos

Quiero agradecer especialmente a Félix Rodríguez, tutor de este Trabajo Fin de Grado, el apoyo, consejo y dedicación demostrados durante el desarrollo del mismo, los cuales me han permitido llevarlo a buen puerto.
También a mi familia. Y a Cristina.

Índice de contenido

ÍNDICE DE ALGORITMOS **9**

ÍNDICE DE FIGURAS **10**

RESUMEN **13**

1 INTRODUCCIÓN **15**

1.1 BIG DATA 16

1.2 MISIÓN SEAWINDS, QUIKSCAT & RAPIDSCAT 17

1.3 NOSQL 20

1.3.1 Teorema CAP *22*

1.3.2 TAXONOMÍA NOSQL 23

1.3.3 MongoDB *24*

1.4 Hadoop *27*

1.5 EXTRACCIÓN, TRANSFORMACIÓN Y CARGA (ETL) 33

1.6 ALCANCE Y OBJETIVOS DEL PROYECTO 33

2 MATERIALES: ENTORNO, HERRAMIENTAS Y LIBRERÍAS **35**

2.1 SOFTWARE 35

2.1.1 MacOs X *35*

2.1.2 Java *35*

2.1.3 Python *36*

2.1.4 MongoDB *36*

2.1.5 Hadoop *36*

2.1.6 Eclipse *37*

2.1.7 LIBRERÍAS PARA LA TRANSFORMACIÓN DE DATOS 37

2.1.7.2 Transformación de datos de RapidSCAT *37*

2.2 HARDWARE 38

3 DESARROLLO DEL PROYECTO **39**

3.1 ESTUDIO DE VIABILIDAD DEL PROYECTO 39

3.1.1 Tareas Realizadas en el proyecto *40*

3.2 ANÁLISIS GENERAL DEL SISTEMA DESARROLLADO 44

3.2.1 Análisis del Conjunto de Programas *45*

3.2.2 Extracción de los Ficheros desde el Repositorio *47*

3.2.3 Transformación de los Ficheros extraídos de PODAAC *48*

3.2.4 Podaac2MongoDB *57*

3.2.5 Mongo2Hadoop *67*

3.2.6 Hadoop *70*
3.2.7 Flujo de Ejecución del Sistema *75*

4 RESULTADOS **77**
4.1 ANÁLISIS DE LOS RESULTADOS DE IMPORTACIÓN DE DATOS 78
4.2 ANÁLISIS DE LOS RESULTADOS DE CONSULTAS TEMPORALES 79

5 CONCLUSIONES **81**

7 ANEXO I: MANUAL DE PROGRAMADOR **83**
7.1 INSTALACIÓN DE PYTHON 83
7.2 INSTALACIÓN DE FILEZILLA 83
7.3 INSTALACIÓN DE MONGODB 84
7.3.1 Instalación de MongoDB utilizando HomeBrew *84*
7.3.2 Instalación de MongoDB utilizando los binarios oficiales *84*
7.4 INSTALACIÓN DE HADOOP 90
7.4.1 Configuración de Hadoop *90*
7.5 INSTALACIÓN DE ECLIPSE 94
7.5.1 Eclipse para MongoDB *94*
7.5.2 Eclipse para Hadoop *96*

8 ANEXO II: MANUAL DE USUARIO **101**
8.1 DESCARGA DE LOS FICHEROS 101
8.2 USO DEL PROGRAMA DE TRANSFORMACIÓN 102
8.2.1 Ejecución y menú principal *102*
8.2.2 Obtención de los datos de RapidSCAT *103*
8.2.3 Unión de los ficheros de datos *104*
8.3 USO DEL PROGRAMA PARA MONGODB 104
8.3.1 Importación de Proyecto a Eclipse *105*
8.3.2 Ejecución de la aplicación *107*
8.4 USO DEL PROGRAMA PARA LA EXPORTACIÓN DE DOCUMENTOS EN MONGODB 110

BIBLIOGRAFÍA **113**

Índice de Algoritmos

ALGORITMO 1 -. PROCEDIMIENTO DE OBTENCIÓN DE FICHEROS EN EL PROGRAMA PRINCIPAL 50
ALGORITMO 2 -. ESPECIFICACIÓN DEL PROCEDIMIENTO QUE SE ENCARGA LA CONVERSIÓN DE FICHEROS QUIKSCAT A CSV 52
ALGORITMO 3 -. ESPECIFICACIÓN DEL PROCEDIMIENTO QUE SE ENCARGA LA CONVERSIÓN DE FICHEROS RAPIDSCAT A CSV 53
ALGORITMO 4 -. ESPECIFICACIÓN DEL PROCEDIMIENTO QUE SE ENCARGA DE LA CONVERSIÓN DE FICHERO NETCDF A CSV 55
ALGORITMO 5 -. ESPECIFICACIÓN DEL ALGORITMO DE UNIÓN DE LOS ARCHIVOS EN UNA MISMA CARPETA 56
ALGORITMO 6 -. PROCEDIMIENTO PRINCIPAL PODAAC2MONGO 64
ALGORITMO 7 -. PROCEDIMIENTO MENÚ PODAAC2MONGO 64
ALGORITMO 8 -. PROCEDIMIENTO DE CONVERSIÓN DE CSV A JSON 66
ALGORITMO 9 -. PROCEDIMIENTO DE IMPORTACIÓN DE LOS DATOS EN MONGODB 66
ALGORITMO 10 -. PROCEDIMIENTO DE EXPORTACIÓN DE MONGODB A HADOOP 68
ALGORITMO 11 -. PROCEDIMIENTO QUE PERMITE OBTENER LOS CAMPOS PARA LA EXPORTACIÓN 70
ALGORITMO 12 -. PROCEDIMIENTO QUE PERMITE OBTENER LA CONSULTA PARA LA EXPORTACIÓN 70
ALGORITMO 13 -. PROCEDIMIENTO MAP ETLHADOOP 74

Índice de Figuras

FIGURA 1 -. SEÑAL DEL SATÉLITE QUIKSCAT ATENUADA POR EL PASO A TRAVÉS DE LAS NUBES [5] 17
FIGURA 2 -. IMAGEN ARTÍSTICA DEL SATÉLITE QUIKSCAT [8] 18
FIGURA 3 -. TEOREMA CAP Y BASES DE DATOS NOSQL ASOCIADAS SEGÚN CUMPLAN UNAS U OTRAS PROPIEDADES [20] 22
FIGURA 4 -. MODELO DE DATOS DE MONGODB VS MODELO DE DATOS RELACIONAL [24] 26
FIGURA 5 -. ECOSISTEMA HADOOP [28] 28
FIGURA 6 -. ARQUITECTURA DEL SISTEMA DE FICHEROS DISTRIBUIDOS HADOOP [30] 29
FIGURA 7 -. DIAGRAMA DE GANTT DE LAS TAREAS PROPUESTAS 42
FIGURA 8 -. DIAGRAMA DE GANTT DE LAS TAREAS REALIZADAS 43
FIGURA 9 -. DIAGRAMA DE PROCESOS DEL SISTEMA METEO BIG DATA 44
FIGURA 10 -. DIAGRAMA DE PROCESOS DETALLADO PARA EL SISTEMA METEO BIG DATA 46
FIGURA 11-. DIAGRAMA DE CLASES DEL PROGRAMA DE TRANSFORMACIÓN 49
FIGURA 12 -. FORMATO DE UN FICHERO CSV DE SALIDA DEL PROGRAMA DE TRANSFORMACIÓN 52
FIGURA 13 -. DIAGRAMA DE CLASES DEL PROYECTO PODAAC2MONGODB 58
FIGURA 14 -. EJEMPLO DE DOCUMENTO METEO BIG DATA 59
FIGURA 15 -. COMANDO QUE PERMITE LEVANTAR LOS SERVICIOS DE MONGODB 61
FIGURA 16 -. COMANDO PARA ACCEDER AL TERMINAL DE MONGODB 61
FIGURA 17 -. TERMINAL DE MONGODB 62
FIGURA 18 -. COMANDO PARA LA CREACIÓN DE LA BASE DE DATOS 62
FIGURA 19 -. SALIDA DEL COMANDO DE CREACIÓN DE BASES DE DATOS 62
FIGURA 20 -. PROCESO DE MAPREDUCE METEOBIGDATA 71
FIGURA 21 -. REGIONES SELECCIONADAS [52] 72
FIGURA 22 -. CICLO DE EJECUCIÓN DEL SISTEMA METEO BIG DATA 76
FIGURA 23 -. COMPARACIÓN DE TIEMPOS DE CARGA DE DATOS 78
FIGURA 24 -. COMPARACIÓN DE TIEMPOS EN CONSULTAS TEMPORALES 79
FIGURA 25 -. PÁGINA DE DESCARGA DE PYTHON 2.7 83
FIGURA 26 -. PÁGINA DE DESCARGAS DE MONGODB 85
FIGURA 27 .- CONTENIDO DEL ARCHIVO DESCARGADO DE MONGODB 85
FIGURA 28 -. ARCHIVOS BINARIOS DE MONGODB 86
FIGURA 29 -. COMPROBACIÓN DE LA CORRECTA INSTALACIÓN DE MONGODB 87
FIGURA 30 -. ACCESO A LA CONSOLA DE MONGODB 88
FIGURA 31 -. RESULTADO DE LA CREACIÓN DEL ÍNDICE TEMPORAL 89
FIGURA 32 -. RESULTADO DE LA CREACIÓN DEL ÍNDICE GEOSPACIAL 89
FIGURA 33 -. RESULTADO COMANDO DE FORMATO DE HDFS 93
FIGURA 34 -. RESULTADO DE INVOCAR HSTART 93
FIGURA 35 -. ECLIPSE MARS IDE PACKAGES 94
FIGURA 36 -. VENTANA DE PROPIEDADES DEL PROYECTO PODAAC2MONGO EN ECLIPSE 95

FIGURA 37 -. CUADRO DE DIÁLOGO DE SELECCIÓN DE JARS 96
FIGURA 38 -. ECLIPSE KEPLER IDE PACKAGES 97
FIGURA 39 -. VENTANA DE MAP/REDUCE LOCATIONS EN ECLIPSE 98
FIGURA 40 -. CREACIÓN DE LOCALIZACIÓN HADOOP 98
FIGURA 41 -. APLICACIÓN FILEZILLA 101
FIGURA 42 -. MENÚ DEL PROGRAMA DE TRANSFORMACIÓN 103
FIGURA 43 -. RESULTADO DEL PROGRAMA DE TRANSFORMACIÓN PARA DATOS PROCEDENTES DE RAPIDSCAT 104
FIGURA 44 -. IMPORTACIÓN DE UN PROYECTO A ECLIPSE 105
FIGURA 45 -. ELECCIÓN DEL TIPO DE PROYECTO A IMPORTAR A ECLIPSE 106
FIGURA 46 -. SELECCIÓN DEL PROYECTO A IMPORTAR 107
FIGURA 47 -. MENÚ DE APLICACIÓN PODAAC2MONGO 108
FIGURA 48 -. ASISTENTE DE TRADUCCIÓN DE DATOS PODAAC2MONGO 108
FIGURA 49 -. -. ASISTENTE PARA LA CARGA DE DATOS EN MONGODB 109
FIGURA 50 -. PROGRAMA DE EXPORTACIÓN DE MONGODB PARA HADOOP 110

Resumen

El objetivo de este Trabajo Fin de Grado es conseguir una integración de los datos registrados por los satélites meteorológicos sobre las superficies, tanto marinas como oceánicas, utilizando para ello una distribución de base de datos NoSQL, que permita el acceso a ellos sin importar la procedencia o formato inicial.

Dicha integración consiste en homogeneizar el almacenamiento de los datos provenientes de diversas fuentes, como pueden ser en este caso los satélites QuikSCAT y RapidSCAT, en un mismo almacén de datos, para el que se decidió utilizar MongoDB.

Los datos con los que se va a trabajar tienen la característica de estar almacenados en distinto formato, como podremos observar en sucesivos apartados del documento y de recoger información en cuya densidad de toma varía según el satélite que los haya recogido.

Inicialmente, deberán descargarse desde los repositorios de la NASA y, posteriormente, aplicar algoritmos de transformación que permitan la correcta carga en el sistema escogido para su persistencia.

Finalmente, los datos serán preparados para realizar sobre ellos un refinamiento que permitirá discernir cuáles son óptimos para la aplicación de tareas de Data Mining y cuáles deben ser desechados sin tener que tratar continuamente con la información extraída de los satélites. Para conseguir llevar a cabo esta parte, se utilizará la plataforma de procesamiento Hadoop.

1 Introducción

El mundo que nos rodea sufre constantes cambios que hacen evolucionar la terminología con la que trabajamos diariamente. Nos encontramos en la era de la información y en ella podemos descubrir conceptos como NoSQL o Big Data, entre otros.

Estos vocablos nacen por la necesidad intrínseca que genera la obligación de manejar la ingente cantidad de datos que se produce actualmente en el planeta, ya sea por los dispositivos electrónicos, los individuos, las organizaciones, etc. Toda esta información debe ser administrada y procesada para conseguir obtener de ella un valor oculto que permita facilitar la toma de decisiones, o bien conseguir esclarecer dudas planteadas sobre un tema de terminado, entre otras muchas cosas.

Para ello podemos utilizar una infinidad de herramientas que nos faciliten estos aspectos. En sucesivos apartados se definirán conceptos claves, como pueden ser Big Data, el significado de las siglas ETL en el proceso de Data Mining, los criterios fundamentales del mundo NoSQL, y qué herramientas han sido las escogidas para resolver las diferentes problemáticas que se han planteado a la hora de llevar a cabo este proyecto.

1.1 Big Data

El término Big Data se define como “La disciplina encargada de gestionar datos masivos” [1], pero es un término cambiante, dado que está suscrito a cambios constantes que hacen evolucionar la terminología. No es una herramienta, sino una materia prima y surge a partir del estudio de las grandes cantidades de datos existentes, que vienen definidos por su Volumen, Variedad, Velocidad, Veracidad y Valor.

La primera de las características más importantes de este concepto hace referencia a la circunstancia de que la cantidad de datos que se manejan supera actualmente el desproporcionado rango de los Exabytes de información. Obviamente, toda esta gran cantidad de datos puede obtenerse de diversas fuentes o ser presentados en infinidad de formas (variedad). Todas las aplicaciones que hacen uso de estos datos necesitan obtener unos tiempos de respuesta (velocidad) mínimos que permitan lograr la obtención de la información correcta en el momento preciso.

Esta información debe ser lo más veraz posible; es decir, las fuentes de las cuáles se obtiene deben ser lo más fiable posible para así poder generar el valor tan ansiado que haga que nuestros datos sirvan para un fin concreto, como puede ser la toma de decisiones críticas en organizaciones o la comprobación de la evolución del tráfico en un portal de Internet, por ejemplo. Debido a esto, en el mundo en el que nos encontramos es necesario determinar qué información queremos obtener, para que el volumen de los datos no nos desborde. Para tal fin, se utilizarán un conjunto de herramientas que permitan el almacenamiento, procesamiento, recuperación y análisis de una cantidad inmensa de datos.

1.2 Misión SeaWinds, QuikSCAT & RapidSCAT

El proyecto SeaWinds [2] fue creado con la firme misión de recolectar datos meteorológicos marinos. Dicho proyecto nace como colaboración entre la NASA [3] (National Aeronautics and Space Administration) y la NOAA [4] (National Oceanic & Atmospheric Administration). La finalidad real de esta misión era el registro de los datos acerca de los aspectos más importantes del viento en una zona geográfica determinada, que permitiera incluso operar con condiciones climatológicas adversas.

Figura 1 -. Señal del Satélite QuikSCAT atenuada por el paso a través de las nubes [5]

Antes de que QuikSCAT [6] –el satélite encargado de registrar la velocidad y dirección del viento en la superficie marina– estuviera operativo, y de que consiguiera realizar tomas de datos incluso con condiciones climatológicas adversas, dispuso de un antecesor, NSCAT [7] (NASA Scatterometer), el cual, tras su lanzamiento en 1996, y después de un fallo imprevisto en 1997, obligó a la NASA a cancelar la misión.

En 1999, al retomarse esta misión, entra en juego QuikSCAT, que estuvo operativo hasta que un fallo mecánico en su instrumento principal, producido a finales de 2009, le impidió continuar con su trabajo.

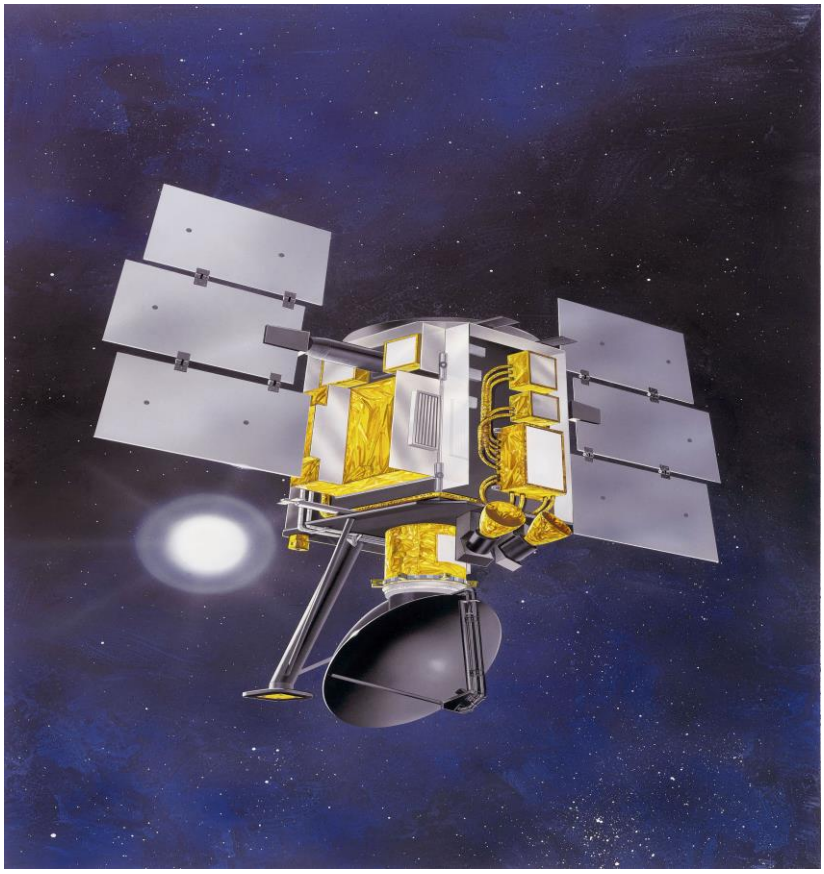

Figura 2 -. Imagen artística del Satélite QuikSCAT [8]

Este satélite orbitaba circularmente alrededor de la tierra a una distancia de 800 Km de la superficie, realizando un ciclo que le permitía efectuar 14 rotaciones a la Tierra en un día.

La misión tenía como objetivos los siguientes [9]:

- Sin que las condiciones climatológicas supusieran un problema, obtener los datos de vientos cercanos a la superficie oceánica.
- Unificar datos provenientes de diferentes instrumentos científicos en ramas del conocimiento dispares, lo que contribuiría al estudio del cambio climático.
- Estudio tanto semestral como anual de los cambios en la flora de las selvas.
- Estudio diario de los movimiento glaciares.

- Determinar la respuesta de los océanos y su interacción con el aire en diferentes escalas espaciales y temporales.
- Mejora y monitorización de la predicción de las tormentas marítimas.
- Mejora en la predicción meteorológica de las zonas costeras.

El componente más importante del QuikSCAT, del cual tomaba su nombre el proyecto, era el escaterómetro *SeaWinds* [10]. Este dispositivo se encargaba de transmitir pulsos de microondas de alta frecuencia (13,4 GHz) en dirección de la superficie oceánica, midiendo los pulsos retornados al satélite por eco.

Los datos captados por el escaterómetro se transmitían al centro de recepción PODAAC, gestionado por la NASA[11], los procesaba y los dejaba libremente en un repositorio FTP de ficheros de datos con formato científico HDF [12]. Los datos de las observaciones registradas contienen básicamente la velocidad y dirección del viento en la superficie oceánica con distintas resoluciones (una observación cada 25 km^2, o bien la resolución con mayor precisión cada 12.5 km^2), realizando barridos en superficies de aproximadamente 1.800 Km de ancho y 400.000 tomas de datos, el equivalente al 90% de la superficie terrestre en un día.

Una vez el satélite QuikSCAT deja de funcionar en 2009, y dada la importancia de la misión, se hace necesario continuar con la obtención de los datos de viento con una gran precisión y calidad. Nace así RapidSCAT [13], consistente en la reutilización del proyecto SeaWind para obtener datos con una precisión similar a la resolución más fina de la misión QuikSCAT, pero con una franja de mediciones reducida a la mitad, ya que este escaterómetro se termina implantando en la Estación Espacial Internacional (ISS), cuya órbita es sumamente menor a la del proyecto original. La idea no es otra que la de no dejar de recopilar estas observaciones, ya que al residir el ingenio empotrado en la ISS se puede realizar un mantenimiento continuo de los componentes.

Los datos extraídos de RapidSCAT son almacenados en un formato ligeramente diferente al HDF y pasan a utilizar un formato autodescriptivo, independiente de la máquina y centrado en vectores de datos, comúnmente conocidos como arrays, creado única y exclusivamente para la distribución de datos de carácter científico. Este formato de almacenamiento es el denominado netCDF-4 (Network Common Data Form) [14].

La extracción de los datos de RapidSCAT se realiza mediante la modificación del código fuente disponible en el repositorio de *PODAAC* de la NASA, desarrollado en Python, para el cual necesitaremos tener instaladas un conjunto de librerías que proporcionan el acceso eficiente a dichos datos.

1.3 NoSQL

Las siglas NoSQL [15] corresponden a Not only SQL, categoría general de sistemas de gestión de bases de datos que difiere de los RDBMS en diferentes modos, ya que estos no tienen esquemas de diseño (Schemas), no permiten los JOIN entre tablas de datos, no garantizan ACID (Atomicidad, Consistencia, Atomicidad y Durabilidad) y su escalado es horizontal (al insertar una nueva tupla, mejora el rendimiento del sistema).

Ha de tenerse en cuenta que, tanto las BDs relacionales, como las NoSQL, son tipos de almacenamiento estructurado. El término fue creado por Carlo Strozzi en 1998 y resucitado por Eric Evans en 2009. Este último sugiere que es mejor referirse a esta forma de almacenamiento como Big Data, mientras que Strozzi cree que es mejor hablar de NoREL (No Relacional).

La principal diferencia entre las BDs relacionales y las NoSQL radica en que en las primeras la información se almacena en un conjunto de tablas que albergan una relación y que, a través de un lenguaje de programación SQL [16], dicha información es convertida en objetos de la vida real. Mientras que en NoSQL, al ser libre de esquemas, no es necesario diseñar tablas y la estructura de los datos, sino que se almacena la información como mejor convenga en el momento del desarrollo. Hay que insistir, no obstante, en que NoSQL no es la panacea; es decir, quizás no es siempre la mejor forma de almacenar los datos, ya que si nuestra aplicación trabaja con datos que soportan mejor un diseño relacional, es mejor utilizar este paradigma, puesto

que los RDBMS ofertan mejores condiciones para los datos en algunos aspectos que los NoSQL.

Las características principales de estos tipos de gestores de bases de datos son [17]:

- Facilitan la escalabilidad horizontal.
- No tienen esquemas fijos y permiten la migración de esquemas.
- Suelen usar un sistema de consultas propio.
- Tienen propiedades ACID en un nodo clúster y tienden a ser "eventualmente consistentes" en el clúster.

Una de las dudas que tratan de resolver tanto los administradores de bases de datos, como los programadores, es por qué surge la necesidad de utilizar BDs NoSQL. Esto es debido a que los desafíos de gestión de información son difíciles de resolver con tecnología de BDs relacionales. Las BDs relacionales no escalan al tráfico de nuestras aplicaciones con un coste aceptable, dado que el tamaño de los esquemas de datos ha ido creciendo exponencialmente.

Se generan muchos datos temporales que no corresponden al almacén de datos principal y se utiliza la desnormalización para conseguir un rendimiento que sin esta estrategia es difícil conseguir. Aparecen los tipos de datos de texto e imágenes (BLOB - tipo binario) y se suelen realizar consultas contra datos que no implican relaciones jerárquicas sencillas, además de utilizar transacciones locales que no necesitan ser durables.

Debido a estos problemas que hemos podido tratar, aparece la filosofía NoSQL que, aunque a menudo ofrece garantías de consistencia débiles, emplea una arquitectura distribuida, donde los datos se guardan de modo redundante en distintos servidores que usan tablas hash, ofreciendo estructuras de datos sencillas como arrays asociativos o almacenes de pares clave-valor (BigTable, DynamoBD y Windows Azure Tables).

1.3.1 Teorema CAP

Eric Brewer [18], ingeniero eléctrico e informático, enunció en 1998 un teorema [19] que hacía referencia a todas las bases de datos de carácter distribuido y que concluía afirmando que sólo es posible garantizar de forma simultánea dos de las siguientes propiedades:

- **Consistency** (Consistencia): todos los nodos de nuestro almacén de datos disponen de los mismos datos en tiempo de lectura.
- **Availability** (Disponibilidad): garantiza que los datos que hay almacenados devuelvan una respuesta a la solicitud del usuario.
- **Partition Tolerance** (Tolerancia a la Partición): el sistema seguirá funcionando incluso si existe la pérdida de alguno de los nodos del sistema, es decir, si por cualquier razón el nodo del cual estamos obteniendo los datos fallase, otro se encargará de dar respuesta a la petición del usuario.

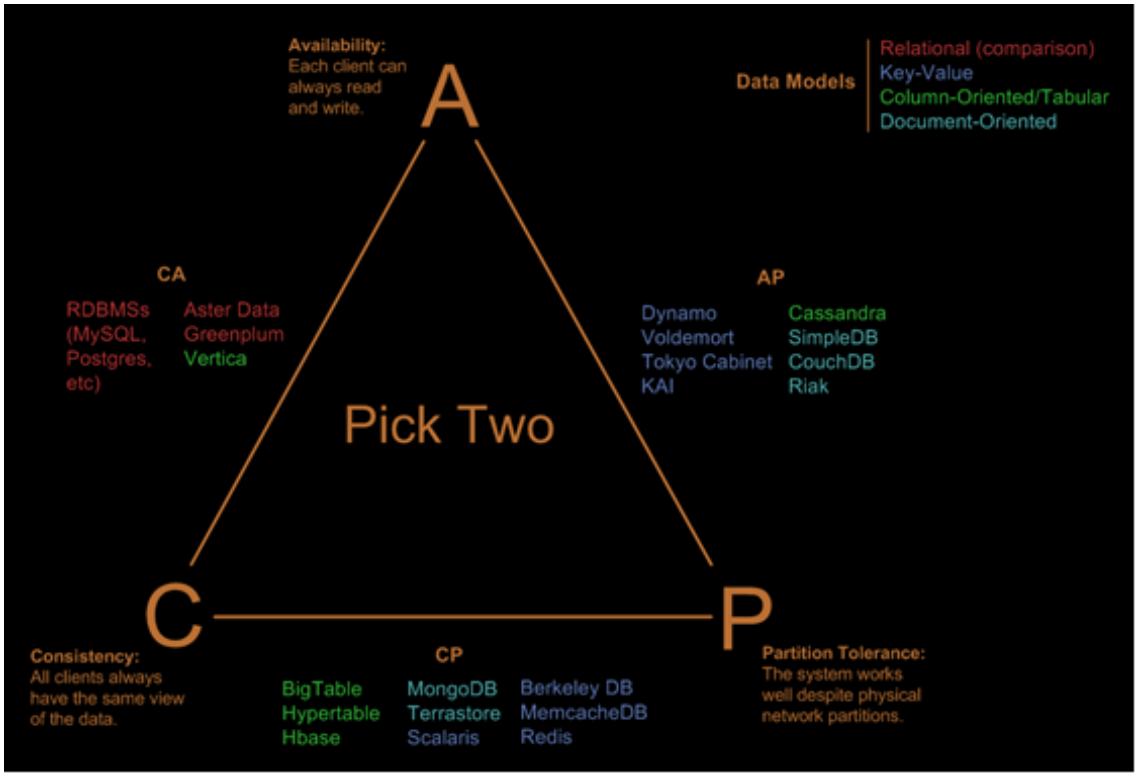

Figura 3 -. Teorema CAP y bases de datos NoSQL asociadas según cumplan unas u otras propiedades [20]

Las bases de datos distribuidas podrán garantizar como máximo dos de las propiedades enunciadas anteriormente. En definitiva, escoger la base de datos con la que vamos a trabajar debe ser una decisión tomada en base a las necesidades de nuestro proyecto.

1.3.2 Taxonomía NoSQL

De la misma manera que podemos clasificar cada sistema gestor de bases de datos NoSQL en base al teorema CAP, podemos hacer una distinción centrándonos en la estructura de almacenamiento interno de los datos de cada SGBD. Los diferentes tipos [21] son los siguientes:

- **Clave-valor**: cada valor almacenado en la base de datos está identificado por una clave. Podríamos entender esta clasificación como un mapa de datos.
- **Orientada a documentos**: el modelo de datos se basa en colecciones de documentos que contienen conjuntos de pares de clave-valor que pueden contener distintos campos entre documentos. Son óptimas en el modelado de datos natural, tienen un implementación bastante ligera y una orientación al desarrollo web.
- **Orientadas a columnas**: es un modelo tabular donde cada fila puede tener una configuración diferente de columnas, es decir, los datos son almacenados por columnas en vez de por filas. Este esquema es bueno a la hora de gestionar el tamaño, las cargas de escritura masivas y la alta disponibilidad.
- **Orientadas a grafos**: su modelo de datos está compuesto por nodos, que son relaciones de clave-valor en ambos. Son buenas para modelar directamente un dominio en forma de grafo, de manera que simplifican el entendimiento de los conjuntos de datos y su representación, además de ofrecer un excelente rendimiento cuando los datos están interconectados y no son tabulares. Permite realizar transacciones que exploten las relaciones entre entidades.

1.3.3 MongoDB

MongoDB [22] es un sistema de bases de datos NoSQL orientado a documentos y desarrollado bajo el concepto de código abierto. MongoDB almacena la estructura de datos en documentos tipos JSON con un esquema dinámico denominado BSON, consiguiendo que la integración de los datos en ciertas aplicaciones sea más fácil y rápida. Esta plataforma cumple con la combinación de *Consistencia* y *Tolerancia a Partición* dentro del teorema CAP, por el cual se rigen las bases de datos que se salen del esquema relacional. Sus características principales son:

- **Consultas Ad-hoc**: soporta la búsqueda por campos, consultas de rangos y expresiones regulares. Las consultas pueden devolver un campo específico del documento, pero también puede ser una función JavaScript definida por el usuario.
- **Indexación**: cualquier campo en un documento de MongoDB puede ser indexado, al igual que es posible hacer indexación secundaria. El concepto es similar a los encontrados en BDs relacionales.
- **Replicación**: soporta el tipo de replicación maestro-esclavo (el maestro puede ejecutar comandos de L/E y el esclavo puede copiar los datos del maestro y sólo se puede usar para lectura, teniendo este último la habilidad de poder elegir un nuevo maestro en caso de que se caiga el servicio con el maestro actual).
- **Balanceo de carga**: distribución de los datos entre los diferentes servidores dedicados al almacenamiento de la información.
- **Agregación**: permite que los usuarios puedan obtener el tipo de resultado que se logra cuando se utiliza un comando “group-by”.
- **Posibilidad de ejecutar JavaScript en el lado servidor**: posibilidad de realizar consultas JavaScript, haciendo que estas sean enviadas directamente a la base de datos para ser ejecutadas.

Debido a su facilidad de uso, al rendimiento en tiempo de consultas y la eficacia con la que MongoDB opera con documentos de un tamaño considerable, esta ha sido el motor de base de datos escogido para el almacenamiento de los datos extraídos del repositorio de *PODAAC*.

La plataforma de estudio cuenta también con un amplio lenguaje de consultas propio que, como características importantes, permite realizar consultas en rango, por tipografía textual, clave, rangos de valores o geoespaciales entre otras. Además de todas estas cualidades, MongoDB se caracteriza por asegurar la posibilidad de realizar una cantidad enorme de consultas simultáneas garantizando el menor tiempo de respuesta posible, siempre y cuando se hayan utilizado correctamente los mecanismos que facilitan estas tareas (indexación, agregación, etc.)

1.3.3.1 Modelo de Datos

El modelo de datos [23] de nuestra plataforma dista mucho del clásico esquema relacional en el que se realiza un almacenamiento tabular, haciendo corresponder las columnas de la tabla a aspectos del mundo real de los que queremos guardar una información determinada, mientras que las filas se equiparan con las ocurrencias de esos elementos.

Una de las exigencias de este tipo de esquemas es que los datos sean creados a priori y respeten –sí o sí– el esquema diseñado para ellos. En este momento, nuestras tablas pasarán a ser colecciones que almacenen elementos con un cierto grado de similitud, lo que significa que podemos encontrar dentro de una misma colección elementos que difieran en la información que almacenan.

Una buena praxis es la agrupación en colecciones de los documentos que tengan un alto porcentaje de relación, aunque esto no sea estrictamente obligatorio. Este ejercicio permite que la indexación, la cual trataremos en apartados posteriores, se realice de una manera más correcta y eficiente, maximizando la velocidad de respuesta de las consultas.

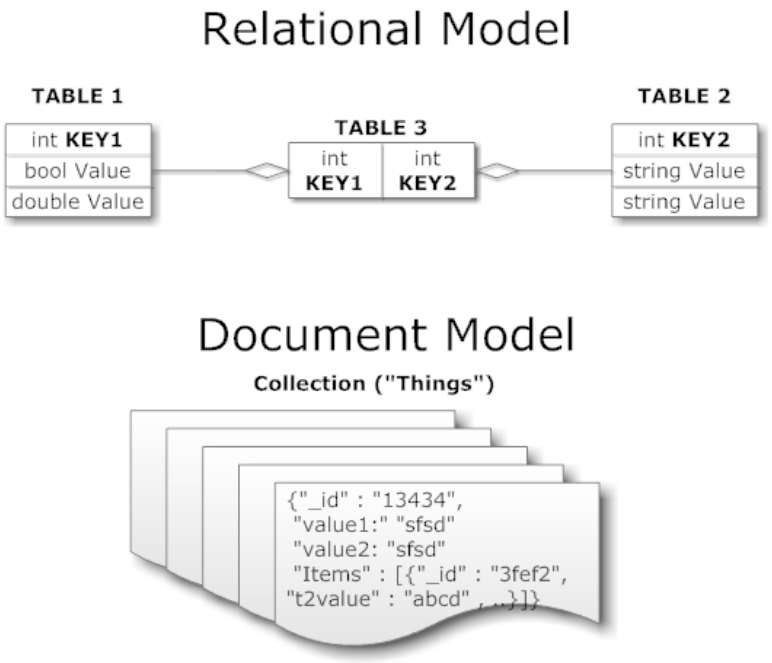

Figura 4 -. Modelo de Datos de MongoDB vs Modelo de Datos Relacional [24]

La solución adoptada para nuestro sistema almacena todos los documentos en una misma colección, encargada de persistir todos los datos relacionados con las tomas sobre las superficies oceánicas y marinas realizadas por los satélites explicados anteriormente (sección 1, apartado 2).

1.3.3.2 Indexación

El mecanismo de indexación [25] que utiliza MongoDB hace que los datos se almacenen en forma de Árbol-B, manteniendo cada uno de los nodos de este balanceado. Esto incrementa la velocidad a la hora de buscar y también a la hora de devolver resultados ya ordenados.

MongoDB, es capaz de recorrer los índices en los dos sentidos, lo que significa que con un solo índice podemos obtener una ordenación ascendente y descendente de la información. Este sistema gestor ofrece la posibilidad de resolver consultas en base a criterios de ordenación utilizando directamente la norma de ordenación con la que generó el índice, evitando así operaciones adicionales.

Además, al realizar una consulta que coincida con el criterio de ordenación del índice, la plataforma devolverá todos los documentos indexados sin la

necesidad de escanear la colección o cargar dichos datos en memoria. Los diferentes tipos de índices más importantes en nuestra plataforma son:

- **Índice por defecto:** todas las colecciones dispondrán de un índice individual por el campo _id (identificador único del documento dentro de la colección).
- **Índices individuales:** índices que se configuran sobre un único campo, pudiendo tener orden ascendente o descendente.
- **Índices compuestos:** índices que permiten acceder a los datos a través de dos campos simultáneamente.
- **Índices multiclave:** permite realizar indexación en base a los campos de un array asociativo.
- **Índices geoespaciales:** permiten la consulta eficiente a bases de datos espaciales, pudiendo existir únicamente un índice de este tipo por colección. Son de dos tipos:
  - **2d:** se utilizan para indexar puntos definidos en un plano euclidiano de dos dimensiones.
  - **2dsphere:** permiten soportar consultas con cálculos geométricos en una esfera similar a la tierra.
- **Índices de texto:** índices que permiten realizar búsqueda por textos.
- **Índices Hash:** se utilizan en entornos distribuidos e indexan el valor hash de un campo determinado, facilitando una distribución más aleatoria de los datos a través del sistema distribuido.

1.4 Hadoop

Hadoop [26] es un framework de Apache [27] que permite el procesamiento de grandes volúmenes de datos a través del uso de clústeres, utilizando para ello un modelo de programación en paralelo. Es un sistema distribuido que utiliza una arquitectura Maestro-Esclavo y utiliza como forma de almacenamiento **Hadoop Distributed File System** (HDFS) y algoritmos **MapReduce** para el procesamiento de datos.

Hadoop abarca diversas herramientas diseñadas e implementadas para trabajar juntas. El resultado es poder utilizarlo para diversas cuestiones. Debemos clasificar Hadoop como un ecosistema integrado por muchos componentes, que van desde el almacenamiento de datos hasta la integración, el procesamiento de los mismos o las herramientas especializadas para analistas de datos.

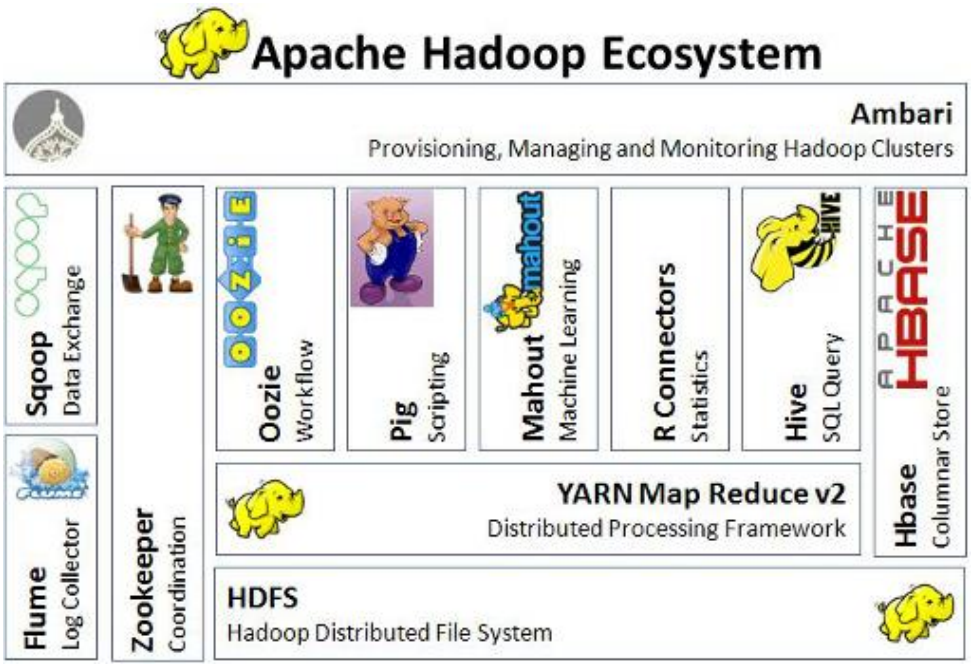

Figura 5 -. Ecosistema Hadoop [28]

Dentro del animalario que nos facilita Hadoop, podemos encontrar que los elementos que más van a interesar al proyecto van a ser los siguientes:

- **HDFS (Hadoop Distributed File System):** Es básico y es el mecanismo por el que una gran cantidad de datos puede distribuirse en un clúster de ordenadores; los datos se escriben una vez, pero se leen múltiples en base a los análisis. Es la base de otras herramientas.
- **MapReduce:** Es un modelo de programación para trabajos distribuidos, que procesan tanto datos en paralelo, fraccionables en los procesos denominados map y reduce. Se utiliza para agilizar el acceso a los datos.
- **YARN:** entorno de gestión de recursos y aplicaciones distribuidas.

1.4.1 HDFS

HDFS o Hadoop Distributed File System [29] está diseñado para el almacenamiento de una gran cantidad de datos, permitiendo así que un gran número de clientes puedan acceder simultáneamente a ellos. Su arquitectura está basada en el sistema de ficheros de Google.

Otorga fiabilidad y control de fallos o pérdidas de clúster utilizando la replicación de los datos. HDFS permite que los datos sean procesados localmente para mejorar la integración con MapReduce.

No es una arquitectura de propósito general, ya que está optimizada para soportar lecturas de flujos elevados, solamente soporta un conjunto de operaciones CRUD sobre los ficheros y no incorpora un mecanismo para el almacenaje de los datos en caché local, dado que el coste es muy alto.

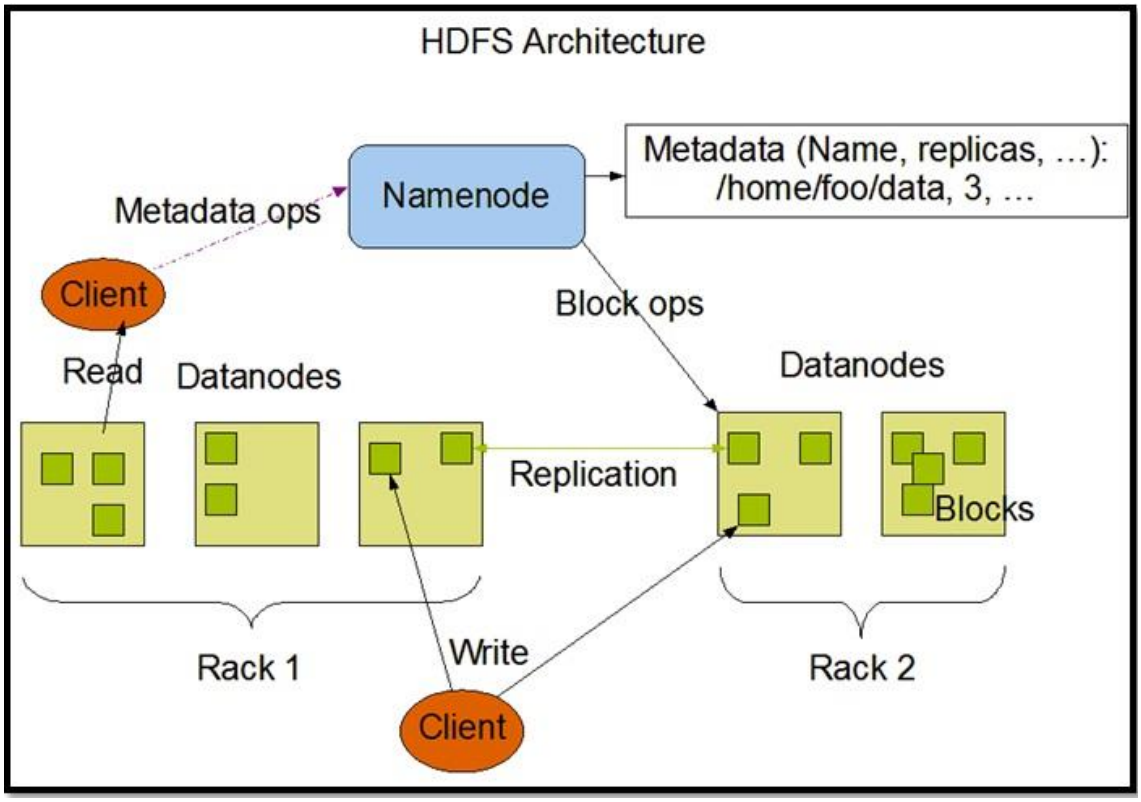

Figura 6 -. Arquitectura del Sistema de Ficheros Distribuidos Hadoop [30]

El DataNode almacena todos los bloques de datos HDFS en un archivo individual ubicado en un sistema de archivos local sin que los archivos HDFS lo sepan (encapsulación de la información). Este no crea todos los

archivos en el mismo directorio, sino que utiliza un método de prueba y error para determinar la cantidad óptima de archivos por directorio/subdirectorio. Debe garantizar la fiabilidad de acceso rápido a los metadatos.

Las estructuras de los metadatos se pueden modificar mediante un gran número de clientes concurrentes, lo que puede generar una pérdida de información a la hora de realizar la sincronización. Esto lo solventa mediante un máquina denominada NameNode, que almacena todos los metadatos del clúster (implementando una arquitectura master/slave). El hecho de que exista un elemento NameNode significa centralizar el control de errores en una sola máquina.

Todo el espacio de nombres del FS está contenido en un fichero denominado FsImage, almacenado en local. Se puede usar un NameNode secundario como Namenode de respaldo y sirve como mecanismo de control para el primario.

Además de almacenar el estado del primario, mantiene dos estructuras de datos en el disco que continúan el estado del sistema de archivos actual: una imagen y un registro de modificaciones.

Durante los inicios y reinicios de NameNode, el estado actual se reconstruye leyendo la imagen y luego volviendo a reproducir el registro de modificaciones.Para mejorar el inicio, un registro de edición se cierra periódicamente y se crea un archivo de imagen nuevo aplicando un registro de edición a la imagen existente.

Para minimizar el impacto a la hora de la creación de puntos de control y el funcionamiento de NameNode, dejamos la tarea de control en manos del secundario, que normalmente está en una máquina diferente. Como resultado del control, el NameNode secundario tiene una copia del estado persistente del primario bajo la forma del último archivo de imagen.

El inconveniente de HDFS es que varios nodos de datos están implicados en el servicio de un archivo, lo que significa que un archivo puede no estar disponible en caso de que alguna de esas máquinas se pierda. Para evitar estos problemas, HDFS replica todos los bloques a través de algunas máquinas.

1.4.1.1 Replicación de Datos

Está implementada como parte de una operación de escritura en la forma de una segmentación de datos. Cuando un cliente realiza una escritura, los datos se escriben primero en un archivo local, y una vez haya completado un bloque completo de datos, se consulta el NameNode para obtener una lista de los DataNode asignados para albergar las réplicas.

El cliente escribe los datos en fragmentos de 4K. El DataNode seleccionado almacena los bloques en su sistema de archivos local y reenvía esta porción de datos al siguiente de la lista. Si uno de los nodos falla, mientras se está escribiendo el bloque, se elimina de la segmentación. Cuando se completa la operación de escritura, se vuelve a repetir el proceso para compensar la réplica faltante. El tamaño de bloque y el factor de replicación están especificados en la configuración de Hadoop.

Una de las herramientas más potentes de HDFS es la optimización de la ubicación de las réplicas, aspecto crucial para la fiabilidad y el rendimiento de HDFS.

Todas las decisiones que atañen a la replicación las toma NameNode, que recibe periódicamente un pulso (garantiza el correcto funcionamiento de los DataNode) y un informe de bloque (verifica que la lista de los bloques en un DataNode corresponde con la información de NameNode).

NameNode se encarga de determinar la ID del rack (clúster) mediante el proceso de conciencia de Hadoop. Una política sencilla para realizar esta tarea es la colocación de réplicas en estructuras únicas.

Una optimización de la política consciente de la estructura es cortar el tráfico de escritura entre estructuras utilizando la cantidad de estructuras, que es inferior a la cantidad de réplicas.

1.4.2 MapReduce

MapReduce [31] Es un modelo de programación sencilla, basada en el paradigma de programación en paralelo, que permite el procesamiento de grandes cantidades de datos mediante la división del algoritmo en dos funciones: *Map* y *Reduce*. Dichas funciones serán aplicadas a todos los datos de entrada y faculta la abstracción de los problemas de los sistemas

distribuidos, centrándose únicamente en el desarrollo de las funciones anteriormente enunciadas.

- **Map:** transforma un conjunto de datos en otra colección de datos formada por números de pares clave/valor. Produce como resultado otra colección de datos en la que se identifica un solo registro por cada valor utilizado como clave, ordenando dichos registros por la clave.
- **Reduce:** procesa el resultado del paso anterior generando un nuevo conjunto de datos a su salida, resultado de ciertas operaciones que se realicen sobre estos y que suelen reducir el número de particiones.

Las funciones Map y Reduce están definidas ambas con respecto a datos estructurados en parejas <clave, valor>. Podemos resumir una ejecución de MapReduce de la siguiente forma [32]:

1. El sistema solicita un número de funciones Map y entrega a cada una de éstas un pedazo de los datos a procesar (que generalmente provienen de un sistema de gestión de datos distribuidos). Cada función Map transforma estos datos en una secuencia de pares (clave, valor).

   Map(k1,v1) list(k2,v2).

2. Entre las fases Map y Reduce, existe una operación de agrupación que reúne todos los pares con la misma clave de todas las listas, creando un grupo por cada una de las diferentes claves generadas y los ordena de acuerdo a sus claves; a continuación distribuye todas las claves sobre varias funciones Reduce, de forma que todos los pares (clave, valor) que tienen la misma clave van a parar a una misma copia de Reduce.
3. Cada Reduce toma todos los valores que recibe para una misma clave y realiza algún tipo de cómputo con los valores que recibe.

   Reduce(k2, list (v2))list(v3)

4. El resultado final de la operación es la combinación de los resultados de cada una de las funciones Reduce.

Las tuplas generadas por la función Reduce son grabadas de forma permanente en el sistema de ficheros distribuido. Por tanto, la ejecución de la aplicación acaba produciendo r ficheros, donde r es el número de tareas Reduce ejecutadas.

1.5 Extracción, Transformación y Carga (ETL)

Son los procesos [33], dentro de la minería de datos, que se encargan de obtener los datos desde el origen de éstos, aplicarles un conjunto de funciones que permitan convertirlos al formato deseado por el programador y, por último, cargarlos correctamente para mantenerlos almacenados en un sistema especifico, o procesarlos para extraer una información que facilite la toma de decisiones por parte de los profesionales de la minería de datos. Se compone de tres fases:

- **Extracción (Extract):** durante ella se realiza la obtención de los datos de uno o varios orígenes, los cuales pueden almacenarlos en formatos diferentes.
- **Transformación (Transform):** terminada la fase de extracción, es necesario aplicar un conjunto de modificaciones a los datos para amoldarlos a la lógica de negocio de nuestra aplicación.
- **Carga (Load):** por último, los datos serán cargados en el sistema destino, ya sea para almacenarlos en un gestor de base de datos, o bien para realizar un procesamiento a través del cual obtener información para la posterior toma de decisiones por parte de los analistas de datos.

1.6 Alcance y Objetivos del Proyecto

Los objetivos de este Trabajo Fin de Grado son principalmente tres. En primer lugar, conseguir realizar correctamente la extracción y transformación de los datos originarios de los repositorios de la NASA.

En segundo lugar, modelar los datos anteriormente descargados para adecuarlos al sistema de bases de datos que vamos a utilizar para su persistencia, consiguiendo así centralizar la información en un solo lugar otorgándola una gran facilidad de acceso.

Por último, preparar los datos que se han almacenado en nuestra base de datos para realizar sobre ellos un refinamiento que permita aplicarles un conjunto de tareas de Data Mining a fin de conseguir un valor oculto, como se ha explicado en apartados anteriores.

2 Materiales: Entorno, herramientas y librerías

En este apartado del documento se explica en detalle el conjunto de herramientas y materiales, tanto software como hardware, que han sido utilizados durante el desarrollo del proyecto.

2.1 Software

A continuación se detalla el entorno software que ha sido utilizado en el proyecto, incluyendo una descripción de los sistemas operativos empleados, así como las librerías y los lenguajes de programación necesarios.

2.1.1 MacOs X

Este sistema operativo ha sido utilizado en su versión 10.11.2 (también denominado “El Capitan”) durante todo el desarrollo del Trabajo Fin de Grado. Sobre él se han ejecutado los programas de transformación de datos descargados del repositorio de la NASA que convierten del formato científico netCDF4 o HDF-4 dependiendo de si los datos provienen de RapidSCAT o QuikSCAT respectivamente a ficheros CSV [34].

Estos ficheros son un tipo de documento que organiza los datos en forma tabular, en los que las columnas se separan mediante comas y las filas por saltos de línea.

Además, también se han instalado el sistema gestor de bases de datos que almacena la información extraída, el framework para el procesamiento de los datos, y la aplicación que permite la visualización de los datos procesados por Hadoop.

2.1.2 Java

Ha sido utilizada la versión 1.7 de la máquina virtual de Java para el desarrollo de las aplicaciones que interactúan con el sistema gestor de bases de datos MongoDB y la plataforma de procesamiento de datos Hadoop.

Para conseguir ejecutar los diferentes programas desarrollados, se han utilizado diferentes APIs proporcionadas en forma de JAR. Podemos encontrarlo de forma gratuita en la web [35].

2.1.3 Python

Se ha utilizado la versión 2.7 de este lenguaje de programación para los programas de transformación de datos de los distintos satélites en ficheros CSV de formato similar. Este lenguaje ha sido escogido por la experiencia ya obtenida por el desarrollo de aplicaciones basadas en él, además de la sencillez de uso y potencia.

Además, se utilizó una librería llamada TKinter [36] que posibilita desarrollar interfaces gráficas de usuario, y que permitió implementar el segundo programa de extracción y transformación de los datos del usuario.

Para instalar el lenguaje, es necesario acceder a la página de descargas de la web de Python [37].

2.1.4 MongoDB

Este ha sido el sistema gestor de bases de datos no relacionales utilizado para el almacenamiento de los datos extraídos de los repositorios de la NASA. La versión escogida ha sido la 3.2.0, debido al conjunto de herramientas y mejoras que incorpora con respecto a versiones anteriores.

La razón de utilizar dicha base de datos se apoya la experiencia que se tiene en su manejo, así como la facilidad y sencillez que tiene su uso y la agilidad con las que las consultas obtienen respuesta utilizando correctamente los índices implementados para la misma.

Se puede realizar la descarga desde la página web de la organización [38] o utilizar *brew* [39], aplicación que permite la automatización en la instalación y configuración de herramientas bajo entornos MacOS X.

2.1.5 Hadoop

Para el procesamiento y caracterización de los datos se ha utilizado el framework Hadoop en su versión 2.7.1. Para la instalación de la misma puede seguirse el Anexo I: Manual de Programador que detalla este proceso.

2.1.6 Eclipse

Se trata en un entorno de desarrollo integrado (IDE en inglés: Integrated Development Environment) que contiene una serie de herramientas de programación de código abierto y multiplataforma. Se han utilizado dos versiones del mismo para realizar la implementación de las diferentes aplicaciones Java:

- **Eclipse Mars:** para el desarrollo de las aplicaciones que permiten interactuar con la base de datos MongoDB, el cual puede descargarse de su página web [40].
- **Eclipse Kepler:** para el desarrollo del conjunto de aplicaciones que hacen factible caracterizar los datos en Hadoop, el cual necesita un módulo software añadido (plugin) para la conexión con Hadoop y cuya instalación se especifica en el Anexo I: Manual de Programador.

2.1.7 Librerías para la transformación de datos

A continuación se describen las librerías necesarias para la correcta ejecución del programa de transformación de datos en Python.

2.1.7.2 Transformación de datos de RapidSCAT

- netCDF-4 [41]: librería que permite tratar ficheros de tipo NetCDF, formato en el que se encuentran los datos del satélite RapidSCAT. El archivo de instalación puede obtenerse en la dirección del proyecto [42].
- Numpy [43]: paquete fundamental para computación científica en Python. Instalable mediante su descarga desde la página web correspondiente [44].

2.2 Hardware

El desarrollo del proyecto ha sido realizado en un ordenador personal de tipo general, con un procesador Intel i7 de doble núcleo a 2,7Ghz, con 4MB de caché de nivel 3 compartida, 10GB de memoria RAM y 500GB de disco duro.

Realmente, este entorno no es el óptimo para el procesamiento de grandes cantidades de datos, ya que siempre pueden existir los problemas de no tener la suficiente memoria RAM o el espacio de disco duro necesario para el almacenamiento de los datos.

3 Desarrollo del Proyecto

3.1 Estudio de Viabilidad del Proyecto

Debido al conglomerado de elementos software de los cuales depende el proyecto, la viabilidad del mismo depende de varios factores. El proceso general consistió en la descarga de los datos de fuentes heterogéneas, la transformación de los mismos y el volcado masivo de estos en un sistema gestor de bases de datos NoSQL, en el que se creó un conjunto de índices que permitieron agilizar el acceso a los documentos de la colección para su exportación.

Inicialmente, es necesaria la descarga de los datos para la que se utilizó un programa de gestión de descargas FTP [45] como es Filezilla [46], evitando la preocupación de la continuidad y corrección de las descargas.

El siguiente paso dentro del proceso era la adecuación de los datos al esquema que se consideró oportuno, es decir, la transformación de los datos, que se llevó a cabo utilizando el lenguaje de programación Python en su versión 2.7. Dicha tarea ha sido viable debido a que se partía de un programa inicial desarrollado por Daniel Teomiro para su TFG [47], el cual tuvo que ser modificado para obtener el esquema deseado.

Para la transformación es necesario utilizar las librerías científicas NetCDF y HDF-4 [12], cuya instalación puede suponer un problema debido a que no existe un amplio soporte documental de las mismas.

La descarga e instalación de la base de datos MongoDB no supuso un gran problema, ya que se había trabajado con anterioridad con la plataforma y además existen diferentes métodos de instalación disponibles en la propia web del sistema [38].

Para continuar con el desarrollo del proyecto era necesario un elemento intermedio que permitiera interconectar nuestro sistema gestor de bases de datos y nuestra plataforma de procesamiento de grandes cantidades de datos: Hadoop. Para conseguirlo, se desarrolló un programa en Python 2.7 que ofrecía al usuario una interfaz que permitiera escoger qué datos extraer de la base de datos para ingresar en Hadoop.

Las principales razones por las que se prefirió trabajar con Hadoop, en lugar de con otras plataformas existentes en el mercado, era por ser de código abierto, además de permitir el procesamiento de gran cantidad de datos en tiempos mínimos, incluso tratándose de ordenadores de uso personal. Además, esta plataforma permite la migración sencilla a procesamiento distribuido en base a la máquina que lo ejecute.

La salida del programa de extracción y transformación comentado anteriormente implica que los datos van a ser cargados en el sistema de ficheros HDFS de Hadoop y a los cuáles se le aplicará un programa de refinamiento para escoger un conjunto de zonas del planeta.

Dados los límites de tiempo en los que se enmarca todo TFG (alrededor de 300 h.), no se llegó a realizar la carga de los datos en la plataforma de procesamiento Hadoop, pero se dejaron preparados los algoritmos que realizan la agrupación de estos según la región geográfica en la que se encuentran.

3.1.1 Tareas Realizadas en el proyecto

Este apartado realiza una descripción de las diferentes tareas realizadas durante el desarrollo del proyecto. Debido a la propia limitación temporal del mismo, no se han podido llevar a cabo todas las tareas inicialmente propuestas. La tabla 1, que aparece a continuación, muestra una estimación temporal de las tareas que inicialmente se planificaron para ser desarrolladas, y la tabla 2 indica las tareas que finalmente se han llevado a cabo.

Tareas Propuestas	Tiempo (días)
Estudio de Hadoop	90
Estudio de MongoDB	25
Instalación de los Programas	5
Descarga de los Datos	2
Mejora del programa de Transformación	10
Transformación de los datos	1
Mejora del programa de Carga en MongoDB	15
Mejora de aplicaciones MapReduce	15
Clasificación y Clustering	5
Mejora de Algoritmos Data Mining	10
Documentación del Proyecto	31

Tabla 1 -. Tareas propuestas y estimación temporal de las mismas

Tarea Realizada	Tiempo (días)
Estudio de Hadoop	90
Estudio de MongoDB	25
Instalación de los Programas	5
Descarga de los Datos	2
Mejora del programa de Transformación	10
Transformación de los datos	1
Mejora del programa de Carga en MongoDB	15
Desarrollo del segundo programa de extracción	15
Mejora de aplicaciones MapReduce	15
Documentación del Proyecto	50

Tabla 2 -. Tareas realizadas y tiempo empleado en las mismas

A continuación, en las figuras 7 y 8, se puede observar la representación en forma de diagrama de Gantt de la duración inicial estimada y de la duración real del proyecto a lo largo del tiempo.

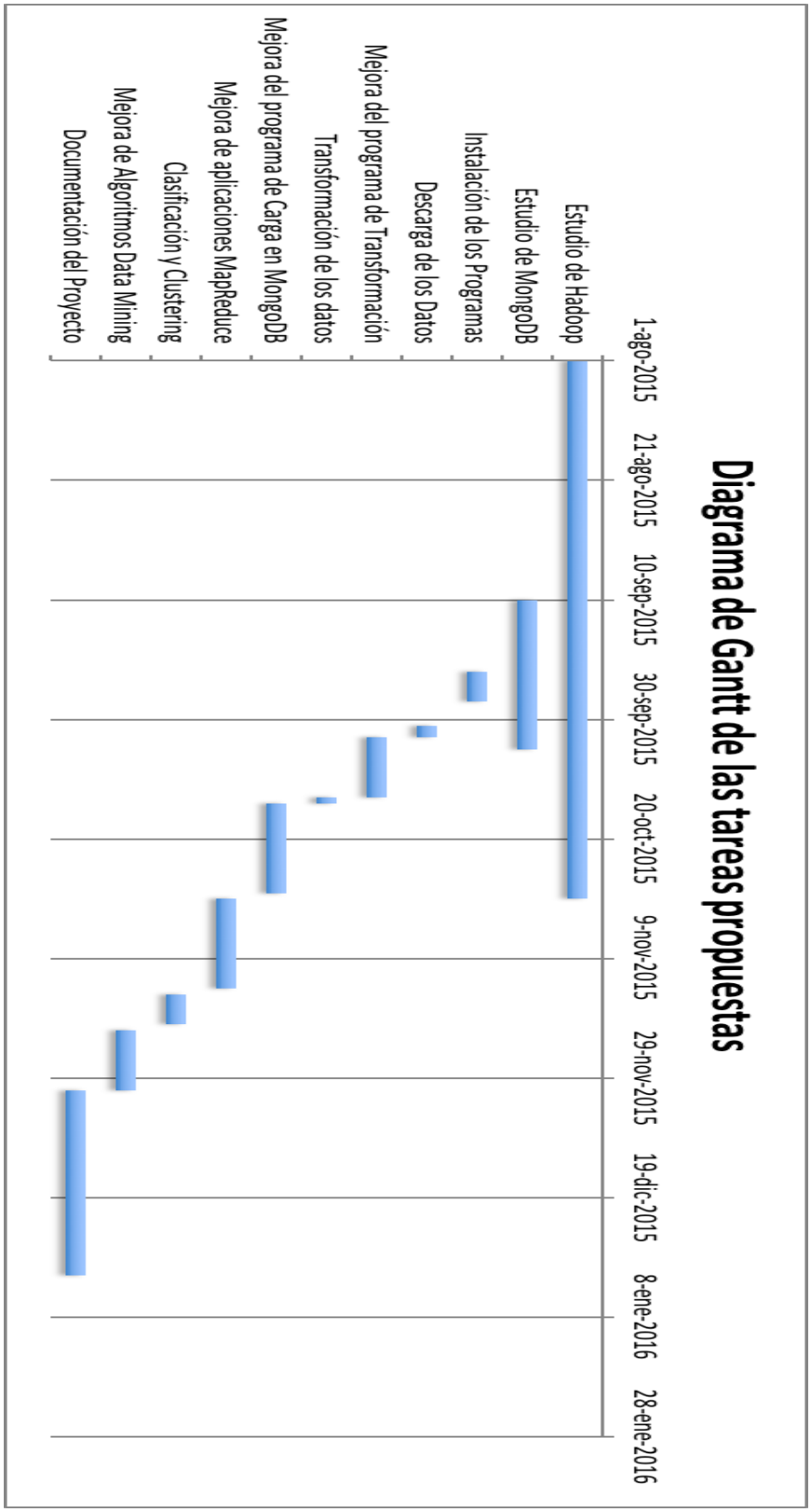

Figura 7 -. Diagrama de Gantt de las tareas propuestas

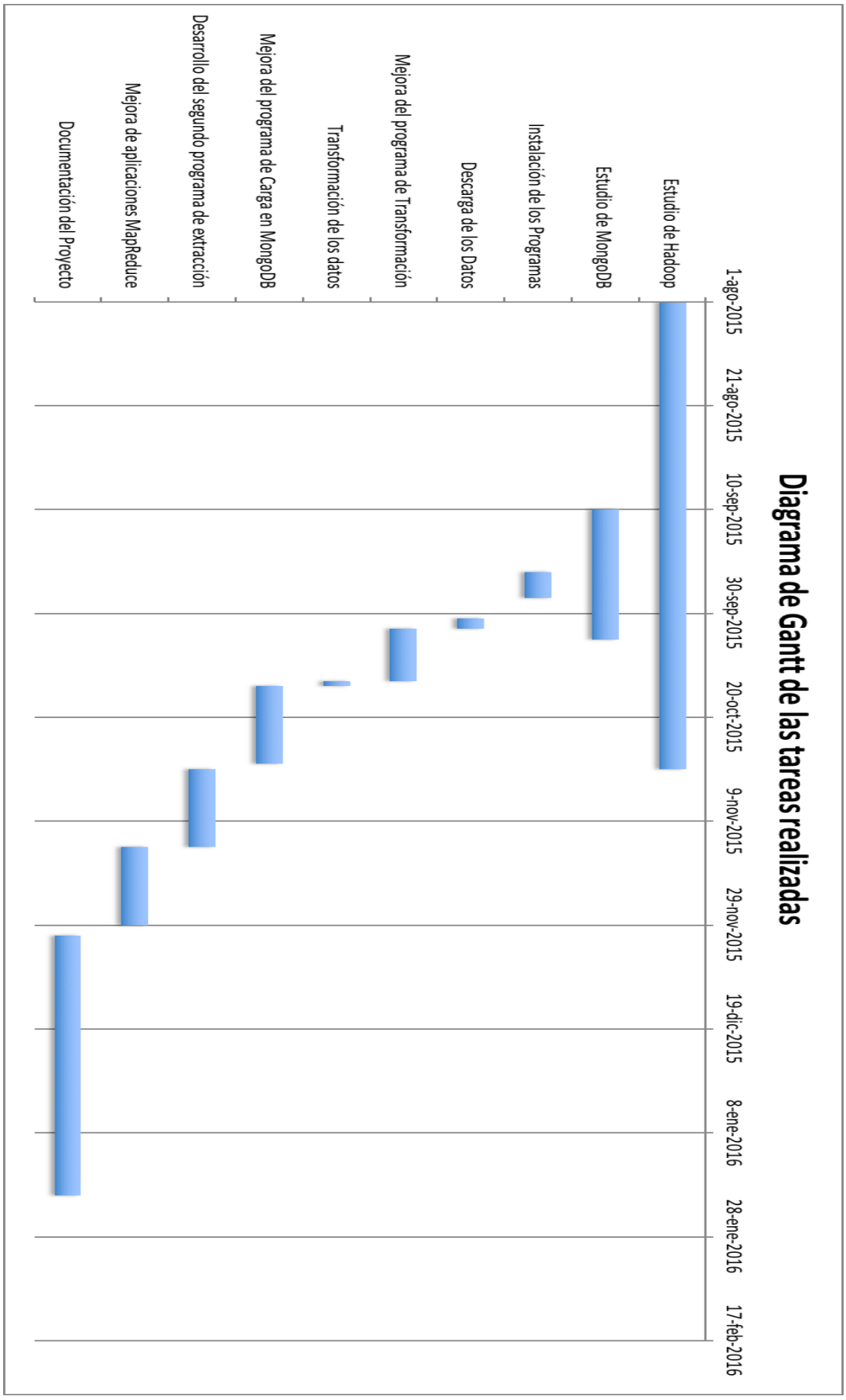

Figura 8 -. Diagrama de Gantt de las tareas realizadas

3.2 Análisis General del Sistema Desarrollado

El sistema desarrollado para llevar a cabo el Trabajo Fin de Grado puede descomponerse en varias partes claramente diferenciadas. Para comprender cada una de ellas se va a utilizar un proceso descendente en el cual se incorpora una explicación minuciosa de cada uno de los componentes.

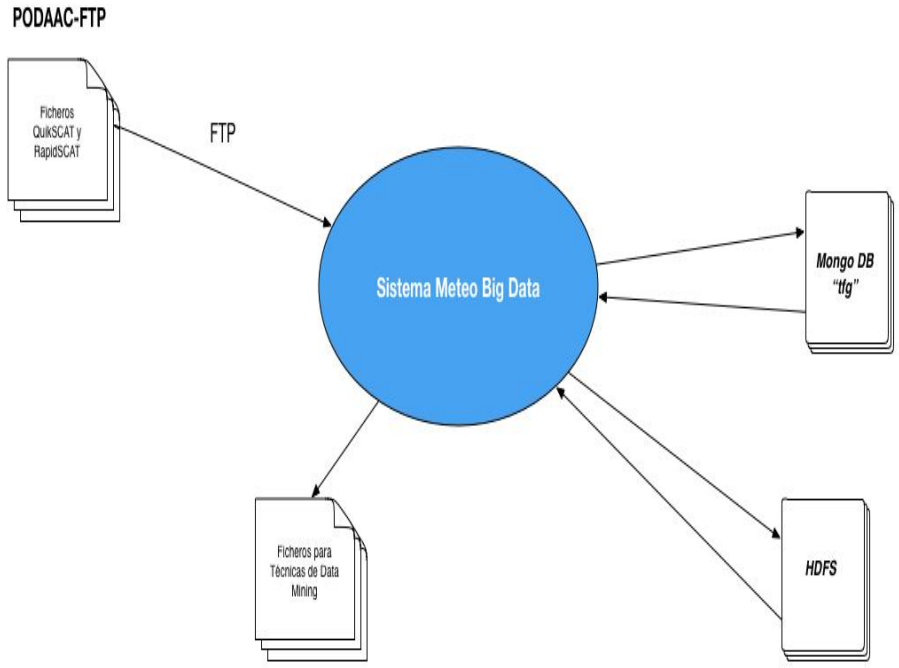

Figura 9 -. Diagrama de Procesos del Sistema Meteo Big Data

La figura 9 se corresponde con la descripción general del sistema. Este recibirá como entrada un conjunto de ficheros almacenados en los repositorios de la NASA que se han extraído utilizando el protocolo de transferencia de ficheros FTP [45].

Una vez que los datos de las observaciones meteorológicas hayan sido descargados, sufrirán un conjunto de transformaciones que permitirán almacenarlos en un sistema de bases de datos NoSQL MongoDB (capítulo 1, sección 3).

Terminada la carga de los datos en la plataforma, se les aplicará los procesos de extracción y transformación que los convierta en archivos HDFS (capítulo 1, sección 4).

Por último, los datos almacenados en MongoDB podrán ser extraídos para ser ejecutados por procesos Hadoop destinados a labores de Data Mining. Los datos extraídos de mongoDB sufrirán un procesamiento paralelo mediante técnicas MapReduce (capítulo 1, sección 4), lo que conllevará la elección de un conjunto de regiones a las que se le aplicarán algoritmos de análisis de datos. Por razones de limitación temporal del TFG, las tareas subsiguientes de minado y visualización de datos analizados quedan pospuestas al futuro para ser continuadas con otros trabajos que se orienten en esta misma dirección.

3.2.1 Análisis del Conjunto de Programas

Los programas implementados constituyen un entorno de interacción con MongoDB y Hadoop, cuyo proceso es reiterativo, ya que suponen la obtención de los datos, la transformación, la carga y el procesamiento de los mismos.

En la figura 10 podemos encontrar una especificación más completa del sistema visto en el apartado anterior. En sucesivos apartados se realiza una explicación más detallada de cada una de las partes del sistema en cuestión.

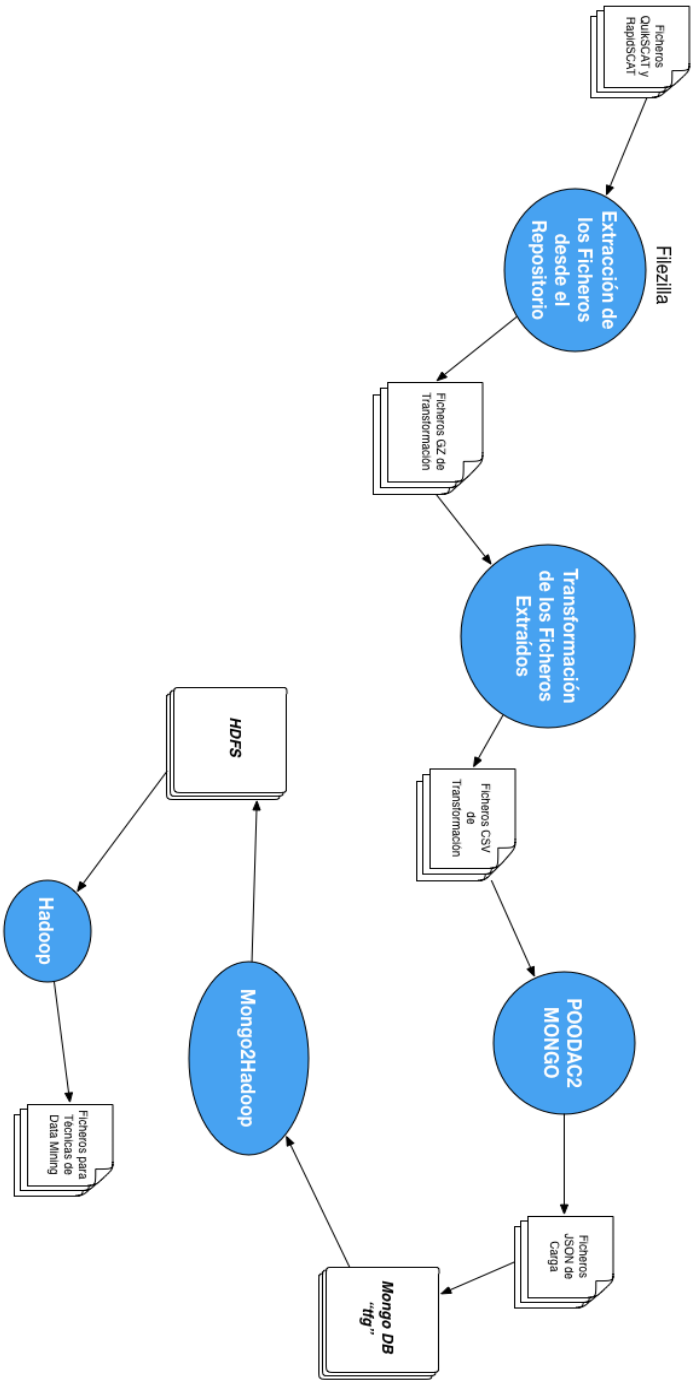

Figura 10 -. Diagrama de Procesos detallado para el sistema Meteo Big Data

3.2.2 Extracción de los Ficheros desde el Repositorio

La extracción de los datos es el proceso inicial que se debe realizar, si deseamos obtener la información que va a ser posteriormente almacenada en nuestra base de datos. Para este paso se ha optado por la solución más sencilla y rápida, consistente en descargar los archivos desde el repositorio FTP de la NASA [50], en el cual se almacenan no solo datos de vientos oceánicos, sino también datos relacionados con la geodesia o la temperatura de los mares.

La forma de descarga puede variar en base a las necesidades del momento o a los deseos del desarrollador, pero para facilitar esta tarea se ha utilizado un programa externo que permite la pausa y reanudación de las descargas, así como las conexiones a servidores rápidas y fiables, de manera que los fallos del sistema estén siempre controlados. El programa escogido ha sido FileZilla [46], libre y de código abierto, cuya instalación y uso se detalla en el Anexo I y II que aparecen al final del documento.

Como se describió en el capítulo 1, sección 2, se desean utilizar datos de dos fuentes en particular, como son los provenientes de los satélites QuikSCAT y RapidSCAT. Para su descarga, debe accederse a la dirección FTP de PODAAC, navegando hasta el directorio OceanWinds, en el que se pueden observar dos directorios, uno para cada una de las misiones de observación comentadas anteriormente.

Para acceder a los datos de QuikSCAT podemos utilizar la dirección oficial de la NASA [48], que almacena aquellos datos cuya densidad sea de 25 km^2 desde el año 1999 al 2009. En base a los que se quieran descargar, bastaría con acceder a la dirección de cada uno desde el programa externo de descargas. Para el desarrollo del proyecto únicamente se han descargado los ficheros del satélite RapidSCAT.

Los datos de la misión RapidSCAT siguen un proceso de descarga similar, accediendo al directorio apropiado [49], que contiene los datos con densidad 12,5km^2 entre los años 2014 y 2015.

3.2.3 Transformación de los Ficheros extraídos de PODAAC

El programa de transformación de los datos descargados de la NASA está implementado en Python 2.7 y se compone de un script principal que se apoya en siete librerías adicionales, también escritas en Python, además de disponer de un programa ejecutable.

El script principal se denomina **transformación.py** y contiene la funcionalidad necesaria que permite imprimir el menú de la aplicación en pantalla, además de la gestión de las selecciones del usuario. A su vez, realiza las llamadas a los métodos de transformación de los módulos ***lectura_L2B*** y ***lectura_L2B12***.

El módulo *lectura_L2B* se encarga de realizar las gestiones necesarias para la transformación de datos con densidad 25 km^2. Posee dos métodos: *GZ_L2B_2CSV_folder*, que es el encargado de organizar las acciones a realizar, y *obtener_csv*, que se ocupa de la ejecución del programa externo para convertir los datos a CSV.

El módulo *lectura_L2B* es el responsable de la transformación de datos con densidad 12,5 km2, siendo su método *GZ_L2B12_2_CSV_folder* el invocado desde el script principal. Asimismo, posee el método *ncf2csv* para conseguir la información de un fichero llamando al módulo *readnc*.

El script *readnc* se ha descargado desde los repositorios de la NASA [50] sin modificación alguna, siendo vital para la lectura de un fichero NetCDF.

El archivo *recorrerdir.py* tiene dos métodos que permiten la obtención de una lista con todos los ficheros que concuerden con una determinada extensión, ya sea sobre un directorio o sobre ese mismo, pero recursivamente.

Otra librería de apoyo es *unirficheros*, que posibilita concatenar diversos archivos CSV en uno solo que contenga todos los datos correspondientes.

Finalmente, la librería *gzhandler* sirve para comprimir o descomprimir ficheros con extensión “.gz”.

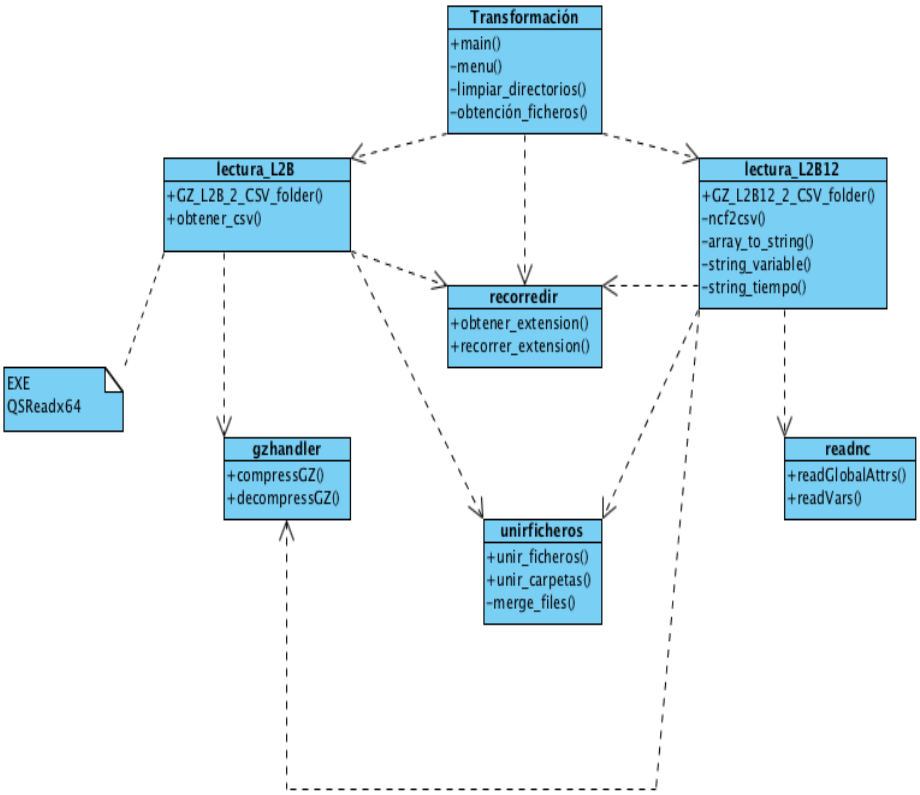

Figura 11-. Diagrama de clases del programa de transformación

Una vez hayan sido descargados los directorios deseados, se deben obtener tantas carpetas como años solicitados, conteniendo a su vez un directorio por cada día del año, cada uno de estos con varios ficheros comprimidos en base a la toma de datos por parte del satélite.

El proceso de transformación puede ser dividido en varias partes según el origen de datos. Sin embargo, los procesos iniciales son comunes a las dos fuentes, ya que es necesario realizar una descompresión previa de los datos organizados por año y día.

Una vez terminado el proceso anterior, se deben leer los ficheros cuyo formato es distinto, puesto que los archivos del satélite QuikSCAT son almacenados en formato HDF-4, mientras que los de RapidSCAT tienen formato netCDF-4; tras la lectura, deben filtrarse los campos deseados –en el caso que nos atañe, todos–, obteniendo a la salida un fichero con formato CSV.

Finalizada la filtración, se fusionan todos los ficheros que daten del mismo día en uno nuevo y terminaría el primer proceso de transformación.

Esta modificación de los datos ha sido desarrollada utilizando un programa en Python 2.7 en el que originalmente se selecciona la fuente de los datos, introduciendo la ruta absoluta en la que se encuentran los correspondientes a un año, así como la ruta de un directorio de salida en el que se almacenarán los ficheros de salida. Este programa se encarga de realizar la preparación de los datos para su posterior incorporación al sistema de bases de datos MongoDB. A continuación, en algoritmo 1, podemos encontrar una explicación detallada del algoritmo del módulo principal que soporta el peso del programa:

```
procedimiento obtención_ficheros (modo)
        ruta = obtener ruta de consola  si ruta es directorio correcto hacer
        lista = obtener_ficheros_gz(ruta)
        num_encontrados = tamaño(lista)
        si num_encontrados > 0 entonces
                ruta_final = obtener ruta de consola
                si ruta_final es directorio correcto entonces
                        si modo == 1 entonces
                                procedimiento GZ_L2B_2_CSV_folder(ruta, ruta_final)
                        fin si
                        si modo == 2 entonces
                                procedimiento GZ_L2B12_2_CSV_folder(ruta, rutafinal)
                        fin si
                        si no entonces
                                unir_ficheros(ruta,rutafinal,modo)
                        fin si no
                fin si
               si no entonces
                        imprimir("Ruta final no valida")
                fin si no
                        imprimir("Encontrados " + num_encontrados + "archivos gz")
        fin si
        si no hacer
                imprimir("Ruta no valida")
        fin si no
fin procedimiento
```

Algoritmo 1 -. Procedimiento de obtención de ficheros en el programa principal

El algoritmo 1 anterior comprueba la validez de las rutas introducidas por el usuario. Tras obtener los ficheros con extensión gz que hay en el directorio como ruta inicial, se realiza una llamada al método de transformación de los datos en ficheros CSV, dependiendo del modo que se haya introducido inicialmente (modo 1 para densidad 25 km^2, modo 2 para densidad 12,5km^2).

Debido a que en los sistemas operativos OS X se crea automáticamente un fichero ".DS_STORE" para mantener un control de versiones acerca de los datos que se están almacenando en el sistema de directorios, se añadió una funcionalidad al programa Python que permitiera, en atención a las rutas especificadas, la unión de los ficheros, dado que, al detectar los archivos ocultos con la extensión especificada anteriormente, la ejecución del algoritmo se detenía.

3.2.3.1 Datos de QuikSCAT (Densidad 25km^2)

Si el usuario decidiera utilizar los datos del satélite QuikSCAT, el programa principal realizaría una llamada al procedimiento *GZ_L2B_2_CSV_folder* del módulo *lectura_L2B*.

Este módulo se encarga del procesamiento de los archivos con formato HDF en base a la ruta especificada como inicial y almacenará los resultados en el directorio de la ruta final, en el que podremos encontrar un fichero CSV por cada día del año especificado.

El elemento más importante de este programa consiste en la reutilización de un programa en C que se encuentra a disposición de cualquiera en la dirección [50]. La compilación y llamada al mismo se realiza desde el procedimiento obtener_csv que podemos observar en el procedimiento explicado en el algoritmo 2 que sigue.

```
procedimiento GZ_L2B_2_CSV_folder (ruta, ruta_final)
        para carpeta en listar(ruta) hacer
                ruta_carpeta = unir(ruta, carpeta)
                lista_gz = obtener ficheros gz de ruta_carpeta
                para fichero_gz en lista_gz hacer
                        fichero_qs = descomprimir(fichero_gz)
                        procedimiento obtener_csv(ruta_carpeta, fichero_qs)
                fin para
        fin para
        procedimiento unir_carpetas(ruta, ruta_final, 1)
fin procedimiento
```

Algoritmo 2 -. Especificación del procedimiento que se encarga la conversión de ficheros QuikSCAT a CSV

El algoritmo 2 recorre cada una de las carpetas que hay en el directorio inicial correspondiente a un año determinado, descomprimiendo los ficheros GZ y convirtiéndolos en archivos CSV.

Los ficheros de salida contienen una línea inicial de información del fichero y cada uno de los datos observados con el formato específico diseñado para ellos. En la siguiente figura 12 se aprecia un ejemplo del fichero de datos obtenido tras la ejecución de este procedimiento.

```
(index) → (Time, Latitude, Longitude, Wind_speed, Wind_dir, Rain_rate, Flags, EFlags, Nudge_wind_speed, Nudge_wind_direct, Cross_track_wind_speed, Atmospheric_Speed, Num_ambiguities, Wind_Objetive, [ Ambiguity_Speed1, Ambiguity_Speed2, Ambiguity_Speed3, Ambiguity_Speed4], [ Ambiguity_Dir1, Ambiguity_Dir2, Ambiguity_Dir3, Ambiguity_Dir4], [ Ambiguity_Objetive1, Ambiguity_Objetive2, Ambiguity_Objetive3, Ambiguity_Objetive4], Num_in_fore, Num_in_aft, Num_out_fore, Num_out_aft)

2014276,-43.37,246.63,0.00,0.00,0.00,0.00,0.00,0.00,0.00,0.00,0.00,0.00,0.00,0.00,
[0.00;0.00;0.00;0.00],[0.00;0.00;0.00;0.00],[0.00;0.00;0.00;0.00],0.00,0.00,0.00,0.00

2014276,-43.48,246.63,0.00,0.00,0.00,0.00,0.00,0.00,0.00,0.00,0.00,0.00,0.00,0.00
,[0.00;0.00;0.00;0.00],[0.00;0.00;0.00;0.00],[0.00;0.00;0.00;0.00],0.00,0.00,0.00,0.00

2014276,-43.70,246.63,0.00,0.00,0.00,0.00,0.00,0.00,0.00,0.00,0.00,0.00,0.00,0.00,
[0.00;0.00;0.00;0.00],[0.00;0.00;0.00;0.00],[0.00;0.00;0.00;0.00],0.00,0.00,0.00,0.00
```

Figura 12 -. Formato de un fichero CSV de salida del programa de transformación

El algoritmo finaliza con la llamada al procedimiento “unir_carpetas” con modo 1, encargado de la unión y modificación final de los archivos CSV que pertenecen al mismo día, así como del traslado al directorio de salida de estos ficheros.

3.2.3.2 Datos de RapidSCAT (Densidad 12.5km^2)

Los datos del satélite RapidSCAT se obtienen mediante la llamada al procedimiento *GZ_L2B12_2_CSV_folder* del módulo *lectura_L2B12* desarrollado también en Python 2.7.

```
procedimiento GZ_L2B12_2_CSV_folder (ruta, ruta_final)
        para carpeta en listar(ruta) hacer
                ruta_carpeta = unir(ruta, carpeta)
                lista_gz = obtener ficheros gz de ruta_carpeta
                para fichero_gz en lista_gz hacer
                        fichero_qs = descomprimir(fichero_gz)
                        ruta_fichero_qs = unir(ruta_carpeta, fichero_qs)
                        procedimiento ncf2csv(ruta_fichero_qs)
                fin para
        fin para
        procedimiento unir_carpetas(ruta, ruta_final, 2)
fin procedimiento
```

Algoritmo 3 -. Especificación del procedimiento que se encarga la conversión de ficheros RapidSCAT a CSV

Este módulo implica la reutilización de un programa alojado en los servidores FTP de la NASA [51]. La modificación que se ha efectuado sobre este programa es el almacenamiento de todas las variables en un fichero CSV con el formato escogido por el programador.

Para proceder a esta conversión se hace uso del procedimiento *ncf2csv*, en el que, tras la apertura del fichero determinado, se obtienen las variables en distintas matrices de datos. Se recorren dichas matrices y, para cada una de las posiciones, se escribe una línea en el fichero de salida con el formato final.

Finalmente, se hace una llamada al procedimiento *unir_carpetas* con el argumento modo igual a 2, indicando que los datos procesados han sido los del satélite RapidSCAT en este caso.

```
procedimiento ncf2csv (ncfile)
        nc_file = abrir(ncfile)
        variables = procedimiento leer_variables(nc_file)

        tiempo = variables[0]
        latitud = variables[1]
        longitud = variables[2]
        wind_speed = variables[3]
        wind_dir = variables[4]
        rain_rate = variables[5]
        flags = variables [6]
        eflags = variables [7]
        nudge_wind_speed =  variables [8]
        nudge_wind_dir = variables [9]
        cross_track_windsp_bias = variables [10]
        atmospheric_speed_bias = variables [11]
        num_ambiguities = variables [12]
        wind_obj = variables [13]
        ambiguity_speed = variables [14]
        ambiguity_dir = variables [15]
        ambiguity_obj = variables [16]
        numb_in_fore = variables [17]
        numb_in_aft = variables [18]
        numb_out_fore = variables [19]
        numb_out_aft = variables [20]
        nombre_fichero = ncfile + ".csv"
        fout = abrir(nombre_fichero)
        para i en tamaño(latitud) hacer
                t = procedimiento obtener_tiempo(tiempo[i])
                para j en tamaño(latitud[i]) hacer
                        la = latitud[i][j]
                        lo = longitud[i][j]
                        ws = wind_speed[i][j]
                        wd = wind_dir[i][j]
                        rr = rain_rate[i][j]
                        flags_val=flags[i][j]
                        eflags_val=eflags[i][j]
```

```
                    nudgewsdp= nudge_wind_speed[i][j]
                    nudgewsdir= nudge_wind_dir[i][j]
                    cross_track_wdsp = cross_track_windsp_bias[i][j]
                    atmsp = atmospheric_speed_bias[i][j]
                    numamb= num_ambiguities[i][j]
                    wdobj = wind_obj[i][j]
                    ambsp = ambiguity_speed[i][j]
                    ambdir = ambiguity_dir[i][j]
                    ambobj = ambiguity_obj[i][j]
                    numinfore = numb_in_fore[i][j]
                    numinfaft = numb_in_aft[i][j]
                    numoutfore = numb_out_fore[i][j]
                    numoutaft = numb_out_aft[i][j]
                    linea = t + "," + la + "," + lo + "," + ws + ","
                    + wd + "," + rr + "," + flags_val + "," + eflags_val + ","
                    + nudgewdsp + "," + nudgewsdir + "," + cross_track_wdsp + ","
                    + atmsp + "," + numamb + "," + wdobj + "," + ambsp + ","
                    + ambdir + "," + ambobj + "," + numinfore + ","
                    + numinfaft + "," + numoutfore + "," + numoutaft  +"\n"
                    escribir línea en fout
                fin para
            fin para
        eliminar(ncfile)
        cerrar(fout)
    fin procedimiento
```

Algoritmo 4 -. Especificación del procedimiento que se encarga de la conversión de fichero netCDF a CSV

3.2.3.3 Unión de los ficheros

Al finalizar, el algoritmo de transformación hace una llamada al procedimiento de unión de carpetas del módulo **unirficheros.py**. Este procedimiento, además de unir los ficheros en uno de salida, realiza las modificaciones necesarias para que el forma de los ficheros transformados sea el mismo independientemente del origen de los datos

```
procedimiento unir_archivos(ruta, lista_ficheros, modo)
        fout = abrir("out.csv")
        para n en tamaño(lista_ficheros) hacer
                f = open(ruta + "/" + lista_ficheros[n])
                si modo == 1 entonces
                        saltar dos lineas
                        para linea en fichero hacer
                                linea = separar línea por “,”
                                linea[0] = formatear linea[0]
                                linea[1] = formatear linea[1]
                                linea[2] = formatear linea[2]
                                linea[3] = formatear linea[3]
                                linea[4] = formatear linea[4]
                                linea[5] = formatear linea[5]
                                linea[6] = formatear linea[6]
                                linea[7] = formatear linea[6]
                                …..
                                linea[20] = formatear linea[20]
                                nueva_linea = unir array linea
                                escribir en fout nueva_linea
                        fin para
                fin si
                si modo == 2 entonces
                        para linea en fichero hacer
                                escribir en fout línea
                        fin para
                fin si
                cerrar fichero
        fin para
        cerrar fout
fin procedimiento
```

Algoritmo 5 -. Especificación del algoritmo de unión de los archivos en una misma carpeta

Si la fuente de la que proceden los datos es del satélite QuikSCAT, es necesaria una alteración sobre cada campo de los datos, principalmente sobre el tiempo, convirtiéndolo a la nomenclatura de fecha juliana (aaaaddd, donde aaaa corresponde al año y ddd al día ordinal del año). Como los ficheros de esta fuente contienen dos líneas de información adicional en la cabecera, es necesario omitirlas en el resultado final.

Si, por lo contrario, la fuente de datos es la del satélite RapidSCAT, simplemente se han de copiar una a una las líneas del archivo, ya que todas las modificaciones han sido realizadas en el procedimiento explicado anteriormente.

3.2.4 Podaac2MongoDB

El programa que permite gestionar la transformación de los datos de salida del programa anterior y la carga de los datos en la plataforma se divide en varias clases:

- **Executable**: clase contenedora del main principal, junto con el programa que permite mostrar el menú de la aplicación por pantalla. Los métodos de esta clase son:
  - ***Main***
  - ***Menú***
- **MongoDBManager:** clase principal que lleva todo el peso de la ejecución del programa. Se compone de los siguientes métodos:
  - ***MongoDBManager***: constructor parametrizado para la creación del proceso de interacción Interfaz-Base de datos.
  - ***load***: método que permite la importación de los documentos JSON creados en la ejecución de este programa.
  - ***csv2json***: método que se encarga de convertir los ficheros recuperados de la transformación anterior y darles el esquema determinado para el almacenamiento.
- **ConverterHelper:** clase de apoyo a la anterior que permite realizar un conjunto de conversiones necesarias para la obtención del esquema a almacenar en MongoDB. Los métodos que la componen son:
  - ***longitudeConverter:*** transforma el formato longitud a uno legible para la plataforma.
  - ***ambiguityValuesConverter:*** método que convierte el String con los valores ambiguos en la toma de mediciones en un array con los valores numéricos predeterminados.
  - ***flagConverter & eflagConverter:*** métodos que se encargan de la conversión del valor numérico de las banderas de calidad

del viento en un array de booleanos en los que la posición indica la bandera activa.

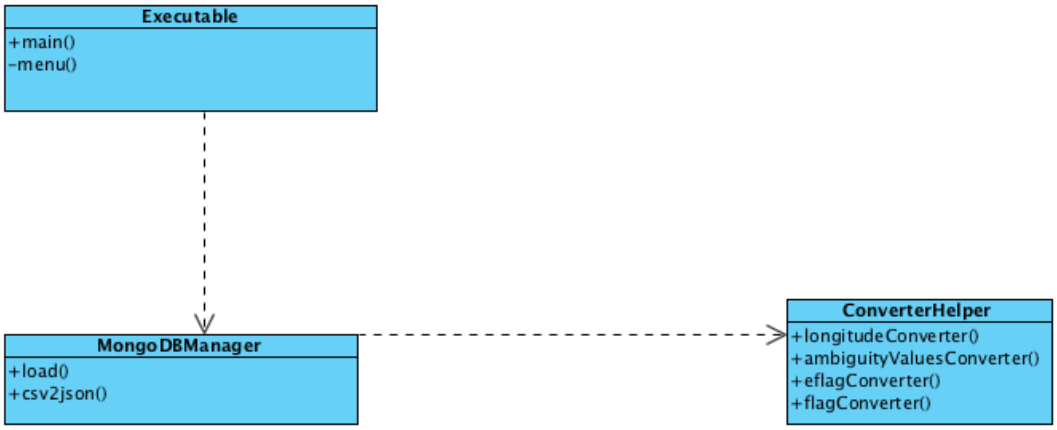

Figura 13 -. Diagrama de Clases del Proyecto Podaac2MongoDB

En la figura 13, podemos observar la relación entre las diferentes clases que permiten la conversión de los ficheros CSV a JSON y su posterior importación a la base de datos.

3.2.4.1 Preparación de MongoDB

Antes de poder cargar información dentro de nuestra base de datos, es necesaria la preparación de la misma. Esto es así debido a que nuestro sistema permite el almacenamiento de información geolocalizada, pero es necesario especificarle al sistema que los datos que va a guardar son de ese tipo.

Para ello, MongoDB provee de un mecanismo de indexación muy eficiente que permite reducir el tiempo de las consultas al rango de los segundos, si la asignación de índices se ha hecho correctamente.

```
{
        "_id" : ObjectId("568b0a6d6b317038f43fe9a5"),
        "time" : 2014276,
        "loc" : {
                "type" : "Point",
                "coordinates" : [-113.369995,-43.37]
        },
        "wind_speed" : 0,
        "wind_dir" : 0,
        "rain_rate" : 0,
        "flag_mask" : [false,false,false,false,false,false,false,false,false,false],
        "eflag_mask" : [false, false,false,false],
        "nudge_wind_speed" : 0,
        "nudge_wind_direct" : 0,
        "cross_track_wind_speed" : 0,
        "atmospheric_speed" : 0,
        "num_ambiguities" : 0,
        "wind_obj" : 0,
        "ambiguity_speed" : [ ],
        "ambiguity_dir" : [ ],
        "ambiguity_obj" : [ ],
        "num_in_fore" : 0,
        "num_in_aft" : 0,
        "num_out_fore" : 0,
        "num_out_aft" : 0
}
```


Figura 14 -. Ejemplo de documento Meteo Big Data

La idea de nuestro sistema es la de efectuar consultas a través de los campos de tiempo y el de localización, por lo que deberemos utilizar un índice individual para el tiempo y uno espacial del tipo *2dsphere* para las coordenadas (véase sección 1, apartado 3, subapartado 2). En la figura XX podemos observar un ejemplo del documento almacenado en MongoDB y una explicación de cada uno de los campos del mismo.

Para el proyecto desarrollado, todos los datos se han almacenado en una única colección, encargada de persistir en base al año en el cual se han tomado los datos, toda la información referente a los vientos marinos, en los que podremos encontrar los siguientes campos:

- **time**: fecha en la cual el satélite ha captado la información. El formato de almacenamiento ha sido YYYYDDD. Los cuatro primeros dígitos se corresponden con el año en el que se ha tomado el dato y los tres siguientes el día dentro del año, en orden secuencial, a partir del uno de enero.
- **loc**: localización geográfica del punto del cual se está captando la información. Es importante indicar que el primero de los valores de las coordenadas es el campo longitud variable entre -180 y 180, y el siguiente contiene la latitud, que oscila entre -90 y 90.
- **wind_speed**: velocidad del viento en esa zona.
- **wind_dir**: dirección del viento en esa zona.
- **rain_rate**: cantidad de lluvia caída en la zona geográfica determinada.
- **flag_mask**: banderas que recogen información acerca de la calidad del viento.
- **eflag_mask**: información adicional acerca de la calidad del viento.
- **nudge_wind_speed**: velocidad de empuje del viento en una zona.
- **nudge_wind_direct**: dirección de empuje del viento en una zona geográfica determinada.
- **cross_track_wind_speed**: velocidad perpendicular del viento en una zona determinada.
- **atmospheric_speed:** velocidad atmosférica de una determinada zona.
- **num_ambiguities:** número de valores erróneos en la toma de información de una determinada zona geográfica. Los valores asociados a este número son:
    - **ambiguity_speed**: velocidad errónea en una determinada zona geográfica.
    - **ambiguity_dir**: dirección errónea de una zona geográfica.

    - **ambiguity_obj**: características esperadas para los valores de ambigüedad en una zona determinada del planeta.
- **wind_obj**: características esperadas del viento en una zona determinada.
- **num_in/out_fore/aft**: número de mediciones realizadas para la recuperación del viento en una zona geográfica.

3.2.4.1.1 Creación de la base de datos

Para crear la base de datos que se modelará durante el desarrollo del proyecto, es necesario acceder a ***mongoshell***, un terminal que viene implementado en MongoDB y que se utiliza para realizar tareas de administración o consulta.

Inicialmente, es necesario levantar el servicio que actúa como controlador en segundo plano de todos los procesos que se lleven a cabo sobre nuestra base de datos. Para ello es necesario ejecutar el siguiente comando descrito en la figura 15:

```
mongod
```

Figura 15 -. Comando que permite levantar los servicios de MongoDB

Una vez haya terminado el proceso invocado, deberemos escribir el comando indicado en la figura 16:

```
mongod
```

Figura 16 -. Comando para acceder al terminal de MongoDB

Para comprobar la corrección en la ejecución del comando, ha de observarse la salida en el terminal mostrada en la figura 17:

```
MongoDB shell version: 3.2.0
connecting to: test
Server has startup warnings:
2016-01-17T10:31:48.081+0100 I CONTROL  [initandlisten]
2016-01-17T10:31:48.081+0100 I CONTROL  [initandlisten] ** WARNING: soft rlimits too low. Number of files is 256, should be at least 1000
>
```

Figura 17 -. Terminal de MongoDB

Una vez nos encontremos en este punto, podremos realizar las tareas de creación de la base de datos y de los índices que permitan realizar consultas, inserciones o modificaciones sobre nuestros datos.

Para crear la base de datos, se hará uso del comando *use* que genera automáticamente la base de datos en el caso de que esta no existiera, y le daremos el nombre de ***tfg***.

```
> use tfg
```

Figura 18 -. Comando para la creación de la base de datos

Que devolverá la respuesta mostrada en la figura 19, la cual indica que la creación ha sido realizada correctamente:

```
Switched to db tfg
```

Figura 19 -. Salida del comando de creación de bases de datos

Finalizado este proceso, podremos insertar con total normalidad los datos obtenidos de los satélites. Para salir del terminal de MongoDB bastará con ejecutar el comando ***exit***.

3.2.4.2 Programa de Gestión

Para la conversión de los ficheros CSV obtenidos en la transformación de los datos de los satélites (sección 3, apartado 2, subapartado 3) y la carga de los mismos en nuestro sistema gestor de base de datos MongoDB, se ha desarrollado un programa en Java que, haciendo uso de una sencilla interfaz gráfica, permite al usuario interactuar con el sistema.

El uso de este programa se especifica en el Anexo II: Manual de Usuario al final del documento.

3.2.4.2.1 Función principal

La gestión del menú se realiza desde la función principal del programa, que además, gestiona la llamada a los métodos correspondientes a cada entrada del menú.

```
función principal
        entero opcion = procedimiento menu
        cadena nombreColeccion, nombreBaseDeDatos
        archivo dir1, dir2
        mientras opcion distinto de 0 hacer
                según opcion hacer
                caso 1 hacer
                        dir1 = obtener directorio CSV desde consola
                        dir2 = obtener directorio JSON desde consola
                        si dir1 es directorio y dir2 es directorio entonces
                                procedimiento CSV2JSON(dir1, dir2)
                        fin si
        fin caso
```

```
                caso 2 hacer
                        dir1 = obtener directorio JSON desde consola
                        nombreColeccion = obtener cadena desde consola
                        nombreBaseDeDatos = obtener cadena desde consola
                        si dir1 es directorio entonces
                                procedimiento carga(dir1, nombreColeccion,
                        nombreBaseDeDatos)
                        fin si
                fin caso
        fin según
    fin mientras
fin función
```

Algoritmo 6 -. Procedimiento principal Podaac2Mongo

En la primera línea del algoritmo 6 se inserta una llamada al procedimiento menú. Este procedimiento, según se puede ver en el algoritmo 7, se encarga de imprimir por pantalla cada una de las opciones disponibles para el usuario, devolviendo la alternativa escogida por el mismo.

```
procedimiento menú retorna entero
        entero opción = -1
        mientras (opción menor que 0) o (opción mayor que 5) hacer
                imprimir("\n\n\n\t\t\tMENU")
                imprimir ("\t1. Traducir CSVs a JSON");
                imprimir ("\t2. Cargar ficheros JSON en MongoDB");
                imprimir ("\n\t0. Salir");
                opción = obtener entero desde consola
        fin mientras
        retornar opción
fin procedimiento
```

Algoritmo 7 -. Procedimiento menú Podaac2Mongo

3.2.4.2.2 Transformación de los datos a JSON

Para proceder a la carga de los datos en MongoDB, es necesaria una conversión previa de los ficheros CSV obtenidos en la transformación comentada en apartados anteriores (sección 3, apartado 2, subapartado 3), en ficheros JSON con el formato visto anteriormente (sección 3, apartado 2, subapartado 4).

La llamada al algoritmo descrito a continuación se efectúa en el momento en el que el usuario selecciona la opción 1 del menú del programa, cuya etiqueta es "*Traducir CSVs a JSON"* y tras la inserción por consola y comprobación de las rutas de trabajo.

En el algoritmo 8 se presenta el recorrido de todos los ficheros en el directorio que almacena los archivos CSV. Para cada uno de esos archivos se lee línea a línea guardando en el fichero de salida la traducción a JSON con el modelo escogido por el desarrollador.

```
procedimiento csv2json(directorioCSV, directorioJSON)
        ficheros = listarDirectorio(directorioCSV)
        ordenar(ficheros)
        para f en ficheros hacer
                si f acaba en ".csv" entonces
                        ficheroSalida = "directorioJSON"+"/f.json"
                        mientras (linea = leerlinea(f)) distinto de null hacer
                                campos = separar(linea, ",")
                                línea = "{time:" + campos[0] +
                                ", loc:{type: \"Point\", coordinates: [ " + (procedimiento
                                transformarLongitud) + " , " + campos[1] + "]}"
                                ", wind_speed:" + campos[3] +
                                ", wind_dir:" + campos[4] +
                                ", rain_rate:" + campos[5] +
                                ",flag_mask:"+procedimiento obtenerFlags+
                                ",eflag_mask:" + procdimiento obtenerEflags+
                                ",nudge_wind_speed:" + campos[8] +
                                ",nudge_wind_direct:" + campos[9] +
```

```
                                    “,cross_track_wind_speed:” + campos[10] +
                                    “,atmosspheric_speed:” + campos[11] +
                                    “,num_ambiguities:” + campos[12] +
                                    “,wind_obj:” + campos[13] +
                                    “,ambiguity_speed:” + campos[14] +
                                    “,ambiguity_dir:” + campos[14] +
                                    “,ambiguity_obj:” + campos[14] +
                                    “,num_in_fore:” + campos[14] +
                                     “,num_in_aft:” + campos[14] +
                                    “,num_out_fore:” + campos[14] +
                                    “,num_out_aft:” + campos[14] +
                                    +"}"
                            escribir línea en ficheroSalida
                            fin mientras
                            cerrar(ficheroSalida)
                    fin si
            fin para
fin procedimiento
```

Algoritmo 8 -. Procedimiento de conversión de CSV a JSON

3.2.4.2.3 Importación de los Datos a MongoDB

En la carga de la información en la plataforma, el usuario indica mediante la consola la ruta absoluta en la que se encuentran los ficheros JSON. Igualmente, indica el nombre de la base de datos y el de la colección a la cual importar la información.

```
procedimiento carga (directorioJSON, nombreBD, nombreColeccion)
        ficheros = listarDirectorio(directorioJSON) ordenar(ficheros)
        para f en ficheros hacer
                si f acaba en “.json” entonces
                        rutaFinal = directorioJSON + "/" + f
                        comando = "mongoimport --db " + nombreBD + " --collection  " +
                        nombreColeccion + " --file " + rutaFinal
                        ejecutarComando(comando)
                fin si
        fin para
fin procedimiento
```

Algoritmo 9 -. Procedimiento de importación de los datos en MongoDB

El algoritmo 9 se encarga de recorrer los ficheros de la ruta absoluta introducida por el usuario, comprobando que su extensión sea JSON. Una vez haya sido comprobada la extensión de dichos archivos, se prepara la ejecución del comando ***mongoimport*** que permite la carga de grandes cantidades de datos.

Finalmente, se realiza una llamada al sistema para la ejecución del comando y se procesa el siguiente fichero en el directorio.

3.2.5 Mongo2Hadoop

En este apartado se pretende realizar una descripción del proceso que permite al usuario la preparación de los datos almacenados en MongoDB para su procesamiento en Hadoop.

El único script que conforma el programa se denomina **mongo2hadoop.py** y contiene la funcionalidad necesaria que permite transformar toda la información comprendida en un rango de fechas al formato de archivos HDFS (sección 1, apartado 4) que utiliza Hadoop para trabajar.

Este programa ha sido desarrollado en Python 2.7 y ofrece, mediante una implementación sencilla una interfaz gráfica al usuario, la exportación de los datos que posteriormente se cargan en la plataforma anteriormente nombrada.

Se le muestra un formulario en el que tendrá que escribir la fecha de inicio de exportación, la fecha del final, el directorio de salida de los datos, el nombre del fichero de salida, el nombre de la base de datos de la cual se van a extraer los datos y, por último, el nombre de la colección en la que se encuentra los datos. También se le permitirá seleccionar qué datos exportar de los documentos pertinentes.

Finalmente, se realizará un proceso de exportación a un archivo temporal. Esto es debido a que la función de exportación de MongoDB añade un conjunto de cabeceras con los nombres de los campos seleccionados. Una vez terminada la exportación de los ficheros, se eliminarán las cabeceras anteriores y el fichero temporal creados.

3.2.5.1 Exportación de los archivos

Para poder realizar la carga de los datos en Hadoop es necesaria la exportación de la información almacenada en nuestro repositorio de datos. El usuario facilitará al programa una fecha de inicio y una fecha de fin que supondrá el rango de valores que serán exportados para su procesamiento en la plataforma anteriormente comentada.

El algoritmo 10 detalla la exportación de los datos. En él se comprueba la corrección de los campos del formulario y, si todos son correctos, comienza la exportación de los datos en un documento temporal.

```
procedimiento export ()
        fechaInicio = obtener cadena desde formulario
        fechaFin = obtener cadena desde formulario
        directorioSalida = obtener cadena desde formulario
        nombreFichero = obtener cadena desde formulario
        nombreBaseDeDatos = obtener cadena desde formulario
        nombreColeccion = obtener cadena dese formulario
        si campos del formulario son correctos entonces
                ejecutarComando ("mongoexport --db "+ nombreBaseDeDatos +"
                --collection "+ nombreColeccion +" --type=csv
                --fields "+ procedimiento obtenerCampos()
                +" --out "+ directorioSalida
                +"/temp.csv --query "
                +procedimiento obtenerConsulta(fechaInicio, fechaFin)
                , shell = True)
                ejecutarComando("tail +2 "+ directorioSalida
                +"/temp.csv > "+ directorioSalida +"/"+ nombreFichero +".csv", shell = True)
                ejecutarComando("rm -rf "+ directorioSalida +"/temp.csv", shell = True)
        fin si
fin procedimiento
```

Algoritmo 10 -. Procedimiento de Exportación de MongoDB a Hadoop

Una vez haya acabado la exportación, comienza el proceso de eliminación de las cabeceras con el comando ***tail +2***. A continuación, se borra el fichero temporal en el que se almacenan los datos de la exportación.

Para poder llevar a cabo esta funcionalidad, el programa hace una llamada a dos métodos implementados en el mismo script principal.

El primero de ellos permite la obtención de los campos que se desean exportar desde la base de datos. Estos campos se le muestran al usuario como Checkbox que podrá seleccionar a su antojo, obligando únicamente a la exportación de los campos de *tiempo, latitud y longitud*.
El código que permite obtener los atributos para la exportación es el que se incluye en el algoritmo 11.

```
procedimiento obtenerCampos () retorna cadena
        cadena camposSeleccion = “time, latitud, longitud”
        si velocidad_viento esta seleccionada entonces
                concatenar (camposSeleccion, “,wind_speed”)
        fin si
        si direccion_viento esta seleccionado entonces
                concatenar (camposSeleccion, “,wind_dir”)
        fin si
        si grado_precipitacion esta seleccionado entonces
                concatenar (camposSeleccion, “,rain_rate”)
        fin si
        si velocidad_empuje esta seleccionada entonces
                concatenar (camposSeleccion, “,nudge_wind_speed”)
        fin si
        si direccion_empuje esta seleccionado entonces
                concatenar (camposSeleccion, “,nudge_wind_dir”)
        fin si
        fin si
        si velocidad_perpendicular_viento esta seleccionada entonces
                concatenar (camposSeleccion, ",cross_track_wind_speed")
        fin si
        si velocidad_atmosferica esta seleccionada entonces
                concatenar (camposSeleccion, ",atmospheric_speed")
        fin si
        si numero_ambiguedades esta seleccionado entonces
                concatenar (camposSeleccion, ",num_ambiguities")
        fin si
        si objetivo_viento esta seleccionado entonces
                concatenar (camposSeleccion, ",wind_obj")
        fin si
```

```
        si mediciones_tomadas esta seleccionado entonces
                concatenar
                (camposSeleccion, ",num_in_fore,num_in_aft,num_out_fore,num_out_aft")
        fin si
        retornar camposSeleccion
fin procedimiento
```

Algoritmo 11 -. Procedimiento que permite obtener los campos para la exportación

No exportamos todos los datos por el miedo al formato de fichero de salida y por cómo puede ser tratado por Hadoop.

Por último, el método de exportación hace uso de un método que permite construir de forma sencilla y visible la consulta que permitirá realizar la exportación de los datos en base a la fecha de inicio y la del fin. Este procedimiento se detalla en el algoritmo 12.

```
procedimiento obtenerConsulta (fechaInicio, fechaFin) retorna cadena
        retornar '"{\"time\":{$gte:"+ fechaInicio +"},\"time\":{$lte:"+ fechaFin +"}}"'
fin procedimiento
```

Algoritmo 12 -. Procedimiento que permite obtener la consulta para la exportación

3.2.6 Hadoop

Se trata del último de los procesos que se acomete en este trabajo de Fin de Grado. Por limitación del tiempo atribuido a un TFG, se ha llegado únicamente a su planteamiento y diseño, que serán descritos en la presente sección.

El proceso se ha denominado ***ETLHadoop*** y consiste en la carga del fichero obtenido en el apartado anterior dentro del ecosistema Hadoop, realizando un conjunto de tareas en paralelo Map y Reduce que refinarán los datos hasta prepararlos para montar sobre ellos futuros procesos de análisis de datos y Data Mining.

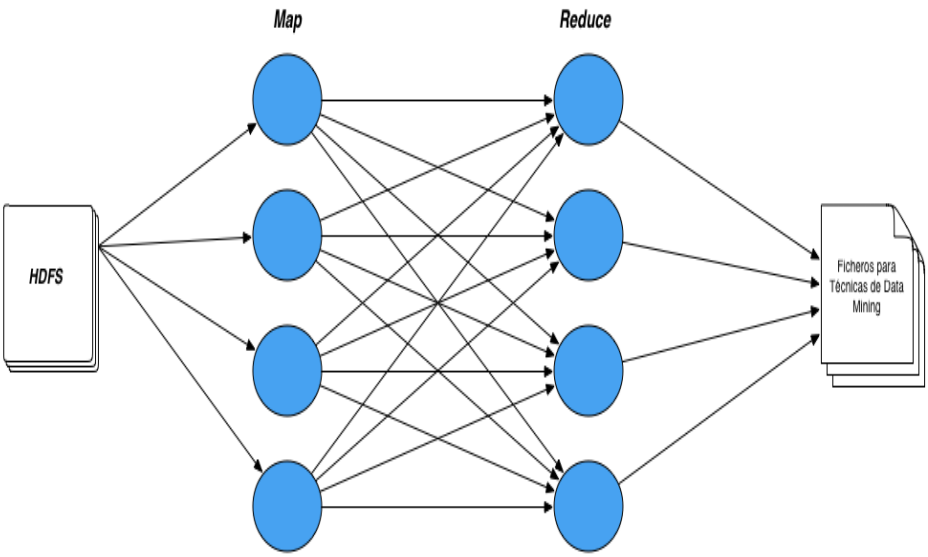

Figura 20 -. Proceso de MapReduce MeteoBigData

El algoritmo recibirá como entrada un archivo CSV cargado previamente en sistema de ficheros HDFS de Hadoop (sección 1, apartado 4). Durante el procesamiento se seleccionarán un conjunto de regiones geográficas, como se puede observar en la Figura 24, generando un archivo que contendrá únicamente información acerca de dichas regiones.

La razón de escoger un conjunto de zonas con inestabilidad climática durante ciertas épocas del año es, por una parte, evitar tener que procesar miles de millones de puntos registrados por los satélites; por otra, poder observar la evolución del cambio climático en zonas que son propensas a los desastres meteorológicos.

A su salida, se obtendrá un fichero de ***firmas/huellas*** [53], que no es más que un archivo que caracteriza la información distinguiéndola del resto sin la necesidad de utilizar toda la que proporciona el satélite diariamente.

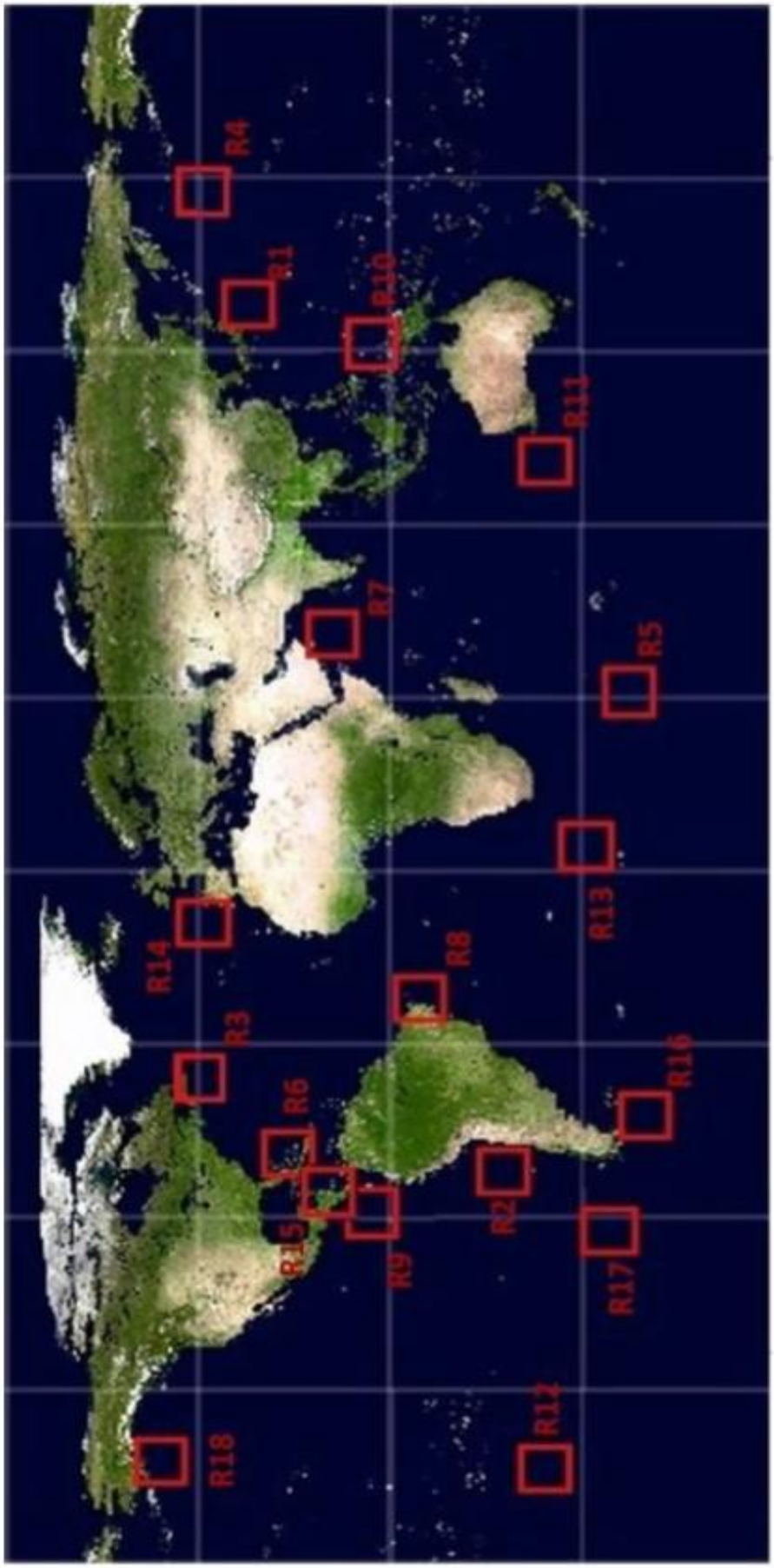

Figura 21 -. Regiones Seleccionadas [52]

Las regiones geográficas referenciadas en la figura anterior serán las utilizadas para la creación del archivo final del sistema. Son las siguientes:

- Región 1 → latitud [-1|-10] y longitud [140|149]
- Región 2 → latitud [-20|-29] y longitud [95|104]
- Región 3 → latitud [-30|-39] y longitud [280|289]
- Región 4 → latitud [-30|-39] y longitud [332|341]
- Región 5 → latitud [-40|-49] y longitud [180|189]
- Región 6 → latitud [-45|-54] y longitud [80|89]
- Región 7 → latitud [-50|-59] y longitud [220|229]
- Región 8 → latitud [-54|-63] y longitud [110|119]
- Región 9 → latitud [0|9] y longitud [310|319]
- Región 10 → latitud [0|9] y longitud [85|94]
- Región 11 → latitud [10|19] y longitud [235|244]
- Región 12 → latitud [10|19] y longitud [90|99]
- Región 13 → latitud [20|29] y longitud [110|119]
- Región 14 → latitud [30|39] y longitud [320|329]
- Región 15 → latitud [40|49] y longitud [130|139]
- Región 16 → latitud [40|49] y longitud [160|169]
- Región 17 → latitud [40|49] y longitud [350|359]
- Región 18 → latitud [50|59] y longitud [20|29]

Los datos correspondientes a las regiones anteriormente especificadas generarán una “firma” o “huella” (signature o fingerprint en inglés), consistente en el valor medio de la relación entre la velocidad del viento y su dirección en dichas regiones. Con estos puntos se consigue discriminar alrededor de un millón de datos en las observaciones diarias del planeta.
A continuación, en el apartado 3.2.6.1, se describe en qué consisten las funciones Map y Reduce.

3.2.6.1 Función Map

La función Map recibirá las latitudes y longitudes de las regiones que se han elegido para el almacenamiento de todos los datos y el descarte del resto. Cada una de las regiones estará formada por 10 latitudes y 10 longitudes, y

los datos comprendidos entre esos valores serán los que se utilicen en el procesamiento.

Para hacer distinciones entre las diferentes zonas se utilizará siempre la latitud y después la longitud de cada una. La clave (key) es compuesta y está formada, en primer lugar, por la fecha de toma de datos en formato juliano; después, la primera de las latitudes de la región y, por último, la primera de las longitudes. Con esto, quedan bien diferenciadas las regiones entre sí.

Como ya vimos en la sección 1, apartado 4, subapartado 3, MapReduce trabaja con pares de clave valor para realizar sus transformaciones. En este caso, el valor (value) contendrá los datos escogidos por el usuario en el programa de extracción de MongoDB, tal y como podemos encontrarlos en el fichero para realizar varias operaciones Reduce. Los datos con los que trabaja la aplicación son la velocidad del viento, su dirección y el grado de precipitaciones. En futuras aproximaciones se enviarán todos los datos que haya en el fichero. El algoritmo 13 describe el proceso de selección de las regiones.

```
procedimiento Map
        escoge según latitud y longitud hacer
                Escribir (Fecha, Latitud_Inicial_Región, Longitud_Inicial_Región,
                Velocidad_viento, Dirección_Viento, Grado_Precipitación)
fin procedimiento
```

Algoritmo 13 -. Procedimiento Map ETLHadoop

3.2.6.2 Función Reduce

En la función Reduce se calcularán diferentes operaciones. En primer lugar, para cada una de las regiones se calculará la media de la velocidad del viento, de la dirección del viento, y la relación existente entre ambas.

Esta operación retorna como salida un fichero en el que cada línea del mismo almacena la fecha juliana de observación, latitud inicial de la región, longitud inicial, la media de la velocidad del viento, la media de la dirección del viento y la relación entre ellas.

Normalmente habrá 18 valores de atributo para cada uno de los días, ya que se han escogido 18 regiones. Puede darse la posibilidad de que haya días

con un número de líneas inferior al normal, lo cual puede ser debido a que en ese día el satélite no llegue a pasar por esa zona o que dicho satélite haya sufrido tareas de mantenimiento.

A la salida de esta función obtendremos el fichero de firmas que permitirá discernir cuáles son las regiones geográficas que nos interesan para su posterior procesamiento.

Por último, con los datos obtenidos anteriormente, se realizarán un conjunto de tareas de análisis y minado de datos para entender la evolución del cambio climático en el planeta a lo largo del tiempo.

3.2.7 Flujo de Ejecución del Sistema

La figura 23 muestra un diagrama de actividad que representa el flujo de trabajo del sistema Meteo Big Data a través de un conjunto de acciones. La totalidad de las acciones han de ser realizadas por el usuario; es decir, no se automatiza ninguno de los procesos que se van a ejecutar.

El diagrama no es más que una explicación no detallada del flujo de ejecución normal del sistema desarrollado en este Trabajo Fin de Grado. El flujo de ejecución comenzaría con la descarga de los datos desde los repositorios utilizando la aplicación Filezilla.

Terminada esta actividad, nos encontramos con una bifurcación, lo que significa que el flujo de ejecución tomará un camino u otro dependiendo del origen de los datos en este caso.

Si los datos proceden del satélite QuikSCAT, el flujo tomará el camino de la izquierda, realizando la transformación explicada en el apartado 3.2.3.1. En el caso de que el origen de los datos sea el satélite RapidSCAT, se ejecutará el programa explicado en el apartado 3.2.3.2.

Una vez hayamos convertido los datos, nos encontramos con una segunda bifurcación que permite escoger el sistema gestor de bases de datos a utilizar. Sucede de este modo por la posibilidad de añadir otra base de datos en futuras aproximaciones.

A continuación, se realiza el proceso de carga en MongoDB, para lo que necesitaremos convertir los ficheros a JSON y cargarlos en el sistema.

Tras la carga, el usuario realizaría la exportación de los documentos almacenados en MongoDB, obteniendo como salida un único fichero CSV que cargará en la plataforma Hadoop.

Por último, se aplicará el algoritmo indicado en el apartado 3.2.6, que devolverá un archivo de firmas/huellas y finalizará la ejecución del sistema Meteo Big Data.

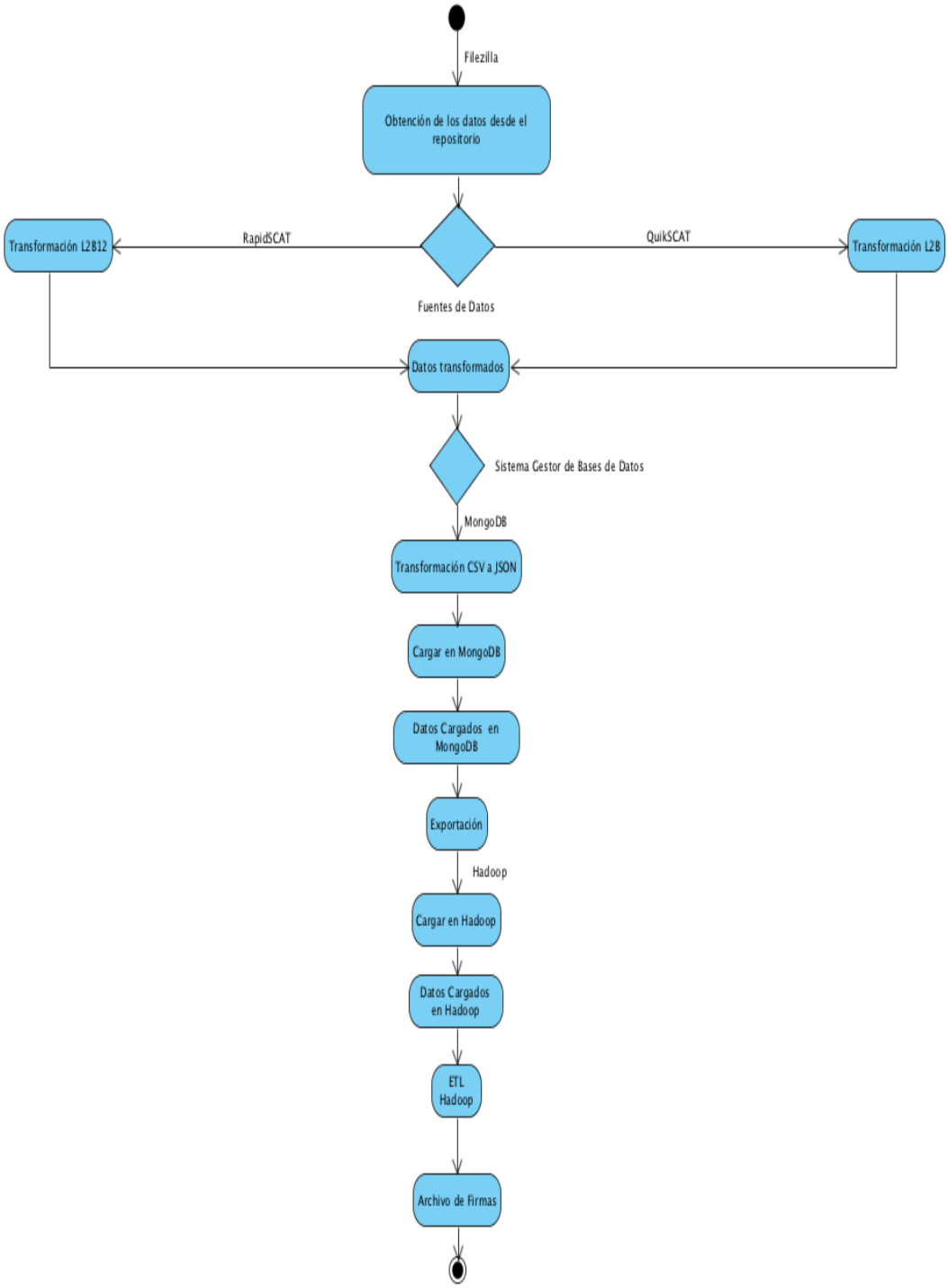

Figura 22 -. Ciclo de Ejecución del Sistema Meteo Big Data

4 Resultados

En el transcurso de este capítulo se discuten los resultados obtenidos durante el desarrollo del proyecto, centrándonos primordialmente en la eficiencia de ejecución. Para ello se procede a una confrontación entre los tiempos de importación de datos en una base de datos MongoDB indexada con otra a la que no se le ha añadido mecanismo de indexación alguno.

De la misma manera, se han medido los tiempos de respuesta que se obtienen al lanzar una consulta temporal a las bases de datos con las características anteriormente enunciadas.

Las mediciones temporales que han supuesto la obtención de estos resultados han sido realizadas contabilizando el tiempo de creación de la consulta, el tiempo de ejecución de la misma y, por último, la obtención de los resultados.

La razón de haber realizado esta elección radica en el hecho de que MongoDB retorna un iterador sobre los resultados en el momento en el que se ejecuta una consulta sobre ella, añadiéndose al tiempo real de consulta el necesitado por el iterador que recorre todos los elementos retornados por la consulta. Si considerásemos únicamente el tiempo de obtención de resultados, éste oscila en el orden de los milisegundos, siendo no muy significativo en nuestro orden de magnitud en el proceso de consulta.

Para realizar las pruebas de velocidad que se explican en este apartado se han utilizado los 15 primeros ficheros del satélite RapidSCAT para el año 2014. Ocupan un tamaño aproximado de 90 GB, conteniendo alrededor de 200 millones de observaciones.

En los siguientes apartados se reflejan las comparativas de uso de las distintas soluciones.

4.1 Análisis de los resultados de importación de datos

El primero de los análisis llevado a cabo hace referencia al tiempo de importación que ha consumido la carga de información en la base de datos. Para este análisis se ha realizado la carga de quince días en MongoDB utilizando una base de datos con indexación temporal y geoespacial y sin indexación.

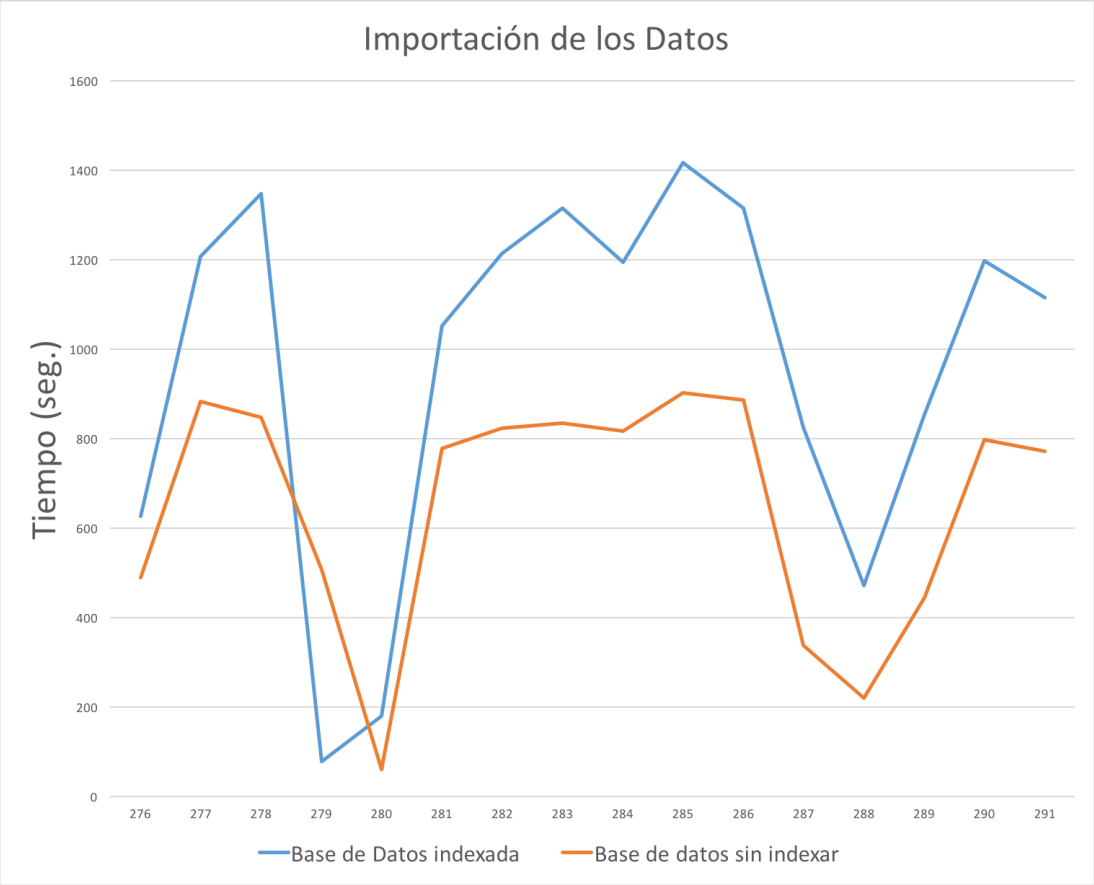

Figura 23 -. Comparación de tiempos de carga de datos

Como se puede apreciar en la figura 23, la carga de datos en una base de datos que no utiliza mecanismos de indexación es más rápida que en una que hace uso de ellos. Esto es debido a que los datos no tienen que ser parejas de elementos; es decir, no es necesario guardar en una estructura de datos el elemento que se desea indexar y su posición en la base de datos para agilizar su acceso.

No obstante, aunque se utilice más tiempo en la carga de la información en nuestro repositorio, se obtienen una ganancia considerable en las consultas por los campos indexados, tal y como se puede observar en el siguiente apartado.

4.2 Análisis de los resultados de consultas temporales

Este análisis se ha realizado con las consultas temporales, que tienen como objetivo recuperar íntegramente los datos de cada uno de los días almacenados en la base de datos.

Por cuestiones de disponibilidad de espacio en la máquina elemental con la que se ha trabajo (descrita en el capítulo 2, sección 2.2 de esta memoria), se ha tomado la decisión de recuperar entre uno y quince días de datos almacenados, obteniendo como resultado la gráfica indicada en la figura 24.

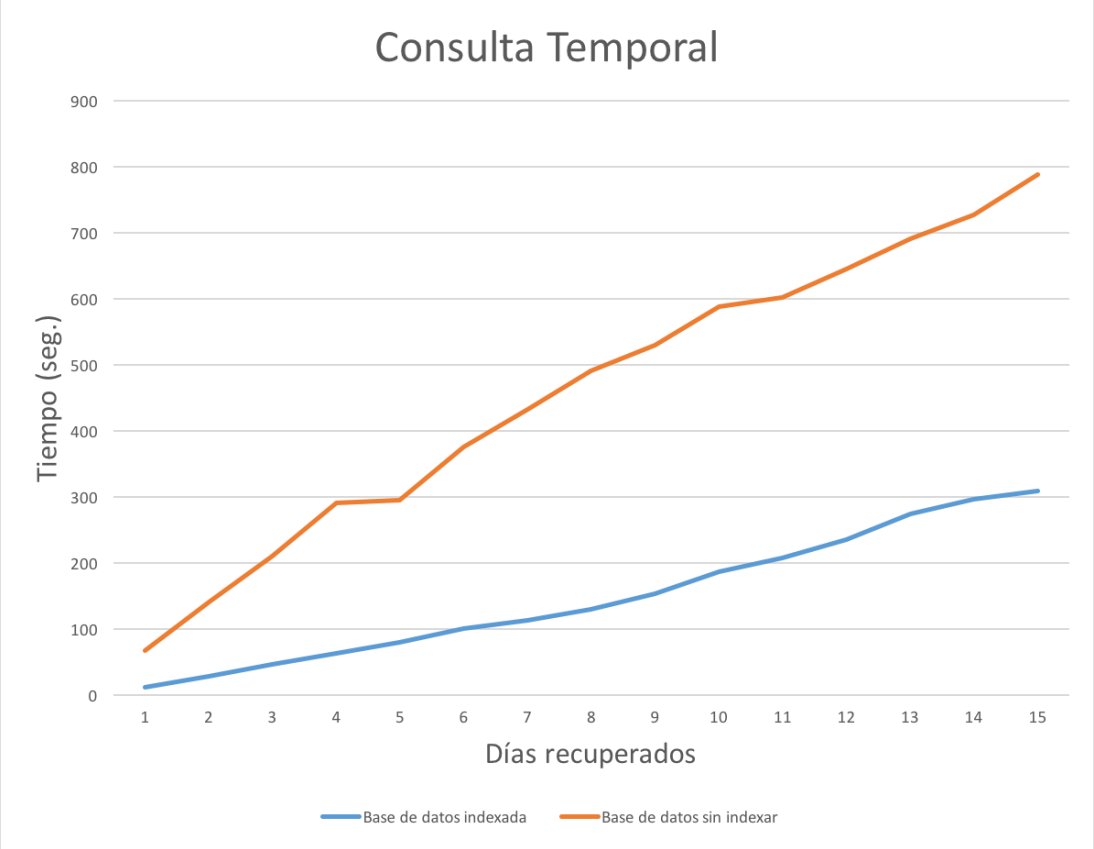

Figura 24 -. Comparación de tiempos en consultas temporales

La figura 24 recoge la gráfica que compara los tiempos de obtención de consultas espacio-temporales en dos bases de datos: una con mecanismos

de indexación y otra sin ellos, de todos los datos correspondientes a cada día de observación almacenado y solicitado en ventanas de consulta de 1 hasta 15 días.

Como se puede observar, una consulta sobre una base de datos correctamente indexada supera con creces el rendimiento que ofrece una sobre la que no se ha implantado dicho mecanismo. Esto es debido a que la plataforma almacena en un especie de diccionario de acceso rápido, los valores de los campos por los cuales se ha establecido la indexación y también las direcciones a los documentos asociados a dichos valores.

Si en la consulta que se está realizando, se detecta que el objeto de dicha consulta es un valor indexado, la plataforma accede al diccionario, y si el valor de consulta existe, devuelve automáticamente todos los campos solicitados en la consulta sin tener que recorrer toda la colección comprobando si el valor de búsqueda coincide con el valor del documento almacenado.

Esta forma de trabajar, agiliza en gran medida el acceso a la información y la obtención de los resultados.

5 Conclusiones

El desarrollo de este Trabajo Fin de Grado ha servido, en primer lugar, para la toma de contacto con datos de científicos, cuyo formato es necesario conocer, estudiar y transformar si deseamos hacer uso de ellos. Para su obtención se ha utilizado un programa gestor de descargas y se ha mejorado una aplicación [47] que permite la descompresión y modificación de estos datos.

Se logra con ello la integración de enormes volúmenes de datos provenientes de distintas fuentes en un único repositorio con capacidad de despliegue distribuido, pero que mantiene las peticiones de uso de datos en cualquiera de las máquinas utilizadas. Esta capacidad la proporciona el uso de la BD NoSQL MongoDB.

Unido a lo anterior, las aplicaciones de análisis y minado de datos de observación geográfica, meteorológicos, climáticos, sobre ecosistemas, vegetación, etc., tan importantes en múltiples disciplinas científicas, pueden ser mucho más sencillas a la hora de alimentarse de datos tan heterogéneos.

Enfrentarse a un reto de estas características supone llevar a cabo una toma de decisiones exhaustiva y constante, lo que significa actuar como un verdadero ingeniero. Esto implica ser capaz de adoptar soluciones sobre la marcha diferentes a la planteada inicialmente en base a las dificultades encontradas durante el desarrollo de las mismas.

Ha sido necesario el estudio del paradigma NoSQL, puesto el sistema gestor de bases de datos utilizado sigue esta filosofía. Esto ha permitido cubrir un rango de soluciones mayor dentro de los posibles problemas que se pueden llegar a plantear.

El almacenamiento NoSQL posibilita la ejecución en paralelo de diversas herramientas de análisis y minado de geodatos.

Bien es cierto que se había realizado un trabajo previo con la plataforma MongoDB, pero ha sido durante la implementación de las soluciones de este proyecto cuando más se ha investigado y descubierto los entresijos que provee este sistema gestor de bases de datos NoSQL.

Una de las metas más importantes, solventada de modo satisfactorio, pasaba por conseguir unificar un conjunto de datos heterogéneo, evitando el acceso a los diferentes repositorios de almacenamiento cada vez que se quisiera trabajar con uno de los tipos de datos estudiados u otro, hecho que agiliza el proceso de interacción y trabajo con la información en gran medida.

Por otra parte, ha de resaltarse que, aunque se conocían los conceptos básicos de programación en Python 2.7, siempre es difícil adecuarse a una lenguaje de programación totalmente distinto a las estudiados durante el Grado en Informática.

Este proyecto también ha permitido conocer y estudiar una plataforma tan potente y con una trayectoria tan amplia como puede ser Hadoop, además de llevar a la práctica un nuevo paradigma como la computación paralela mediante el uso de técnicas Map y Reduce.

Para finalizar, me gustaría indicar que este proyecto ha sido el resultado de muchos meses de estudio, análisis, desarrollo y documentación. Poco a poco, mediante mimo y brega, se ha ido materializando en la forma que puede observarse ahora.

7 Anexo I: Manual de Programador

En este apartado se especifica cómo ha de ser preparado el entorno para llevar a cabo sin problemas cada una de las diferentes tareas que componen el proyecto.

Es necesaria la instalación previa del ***Java Development Kit***. Para el desarrollo de este proyecto se ha utilizado la versión 7.

7.1 Instalación de Python

Para hacer uso de la plataforma Python en MacOS X basta con acceder a su página de descargas [37] y escoger el indicado, según se muestra en la figura 25.

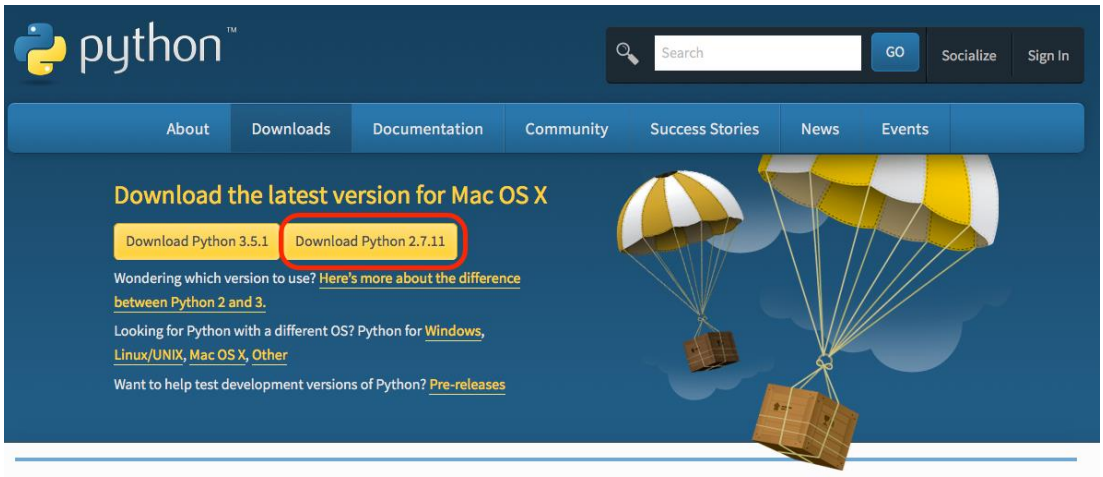

Figura 25 -. Página de descarga de Python 2.7

Una vez haya finalizado la descarga, bastará con ejecutar el instalador con formato .pkg y seguir los pasos que indica el mismo.

7.2 Instalación de Filezilla

En MacOS X accederemos desde nuestro navegador a la página oficial de Filezilla [46] y descargaremos el cliente. Una vez haya finalizado la descarga, abriremos el instalador con formato .dmg y arrastraremos su contenido a la carpeta de Aplicaciones de nuestro sistema.

7.3 Instalación de MongoDB

Para instalar MongoDB en MacOS X podemos utilizar dos tipos de instalación, en base a los deseos del programador. En este apartado se muestra una descripción de los formatos de instalación posibles.

7.3.1 Instalación de MongoDB utilizando HomeBrew

En primer lugar, es necesario tener instalado previamente el software de resolución de dependencias ***Homebrew [39]***, para lo que escribiremos en la ventana de terminal:

```
ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

Una vez haya terminado el proceso de instalación, o bien en el caso en el que hubiéramos instalado Homebrew, ejecutaremos el siguiente comando para la actualización de la base de datos de paquetes del software:

```
brew update
```

Por último, escribiremos en la ventana de terminal el comando descrito a continuación para instalar la plataforma:

```
brew install mongodb
```

7.3.2 Instalación de MongoDB utilizando los binarios oficiales

En primer lugar, es necesario acceder a la página de descargas de la plataforma utilizada [38]. Desde allí escogeremos la versión que más se adapte a las características de la máquina y descargaremos el archivo como puede observarse en la figura 26.

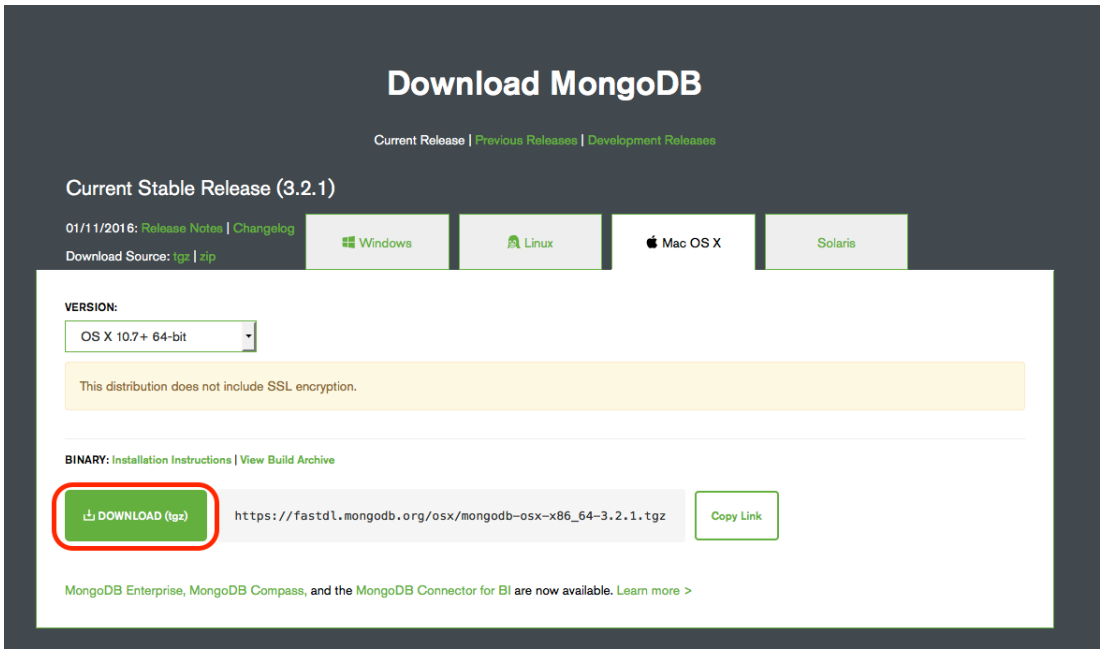

Figura 26 -. Página de descargas de MongoDB

Una vez finalizada la descarga, accederemos al directorio en el cuál se almacenó el archivo con los binarios oficiales de nuestra plataforma, y los descomprimiremos con el gestor de descompresión que el programador crea oportuno. Debemos observar un formato de directorios parecido al que se muestra en la figura 27.

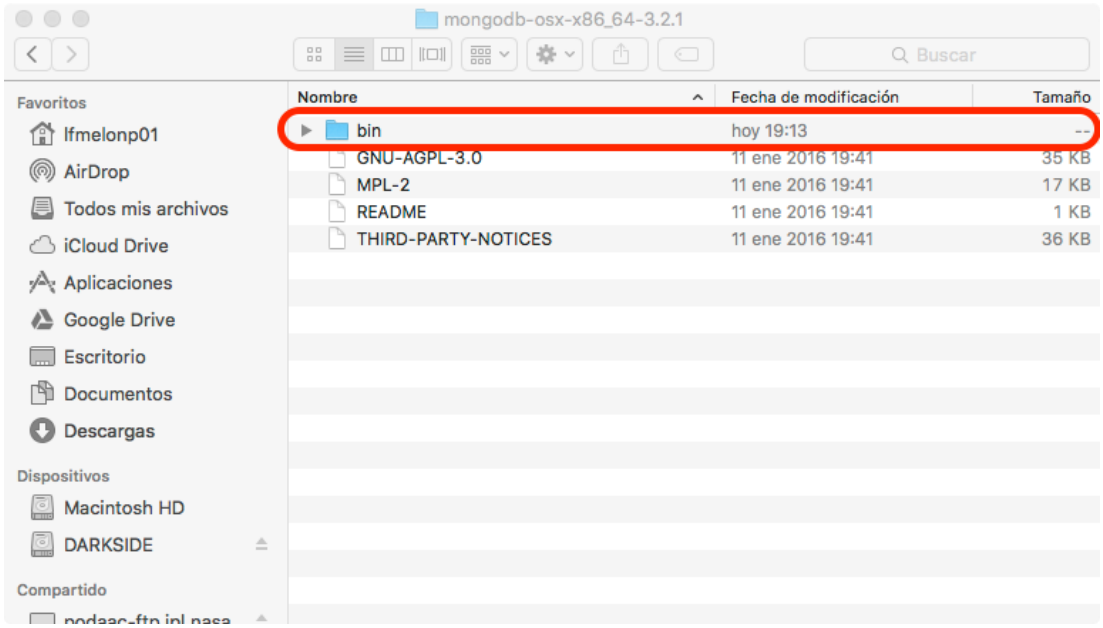

Figura 27 .- Contenido del archivo descargado de MongoDB

Mediante la ventana de terminal abierta, accederemos a dicha carpeta y listaremos los archivos contenidos mediante la orden ***ls –la***, según se puede ver en la figura 28.

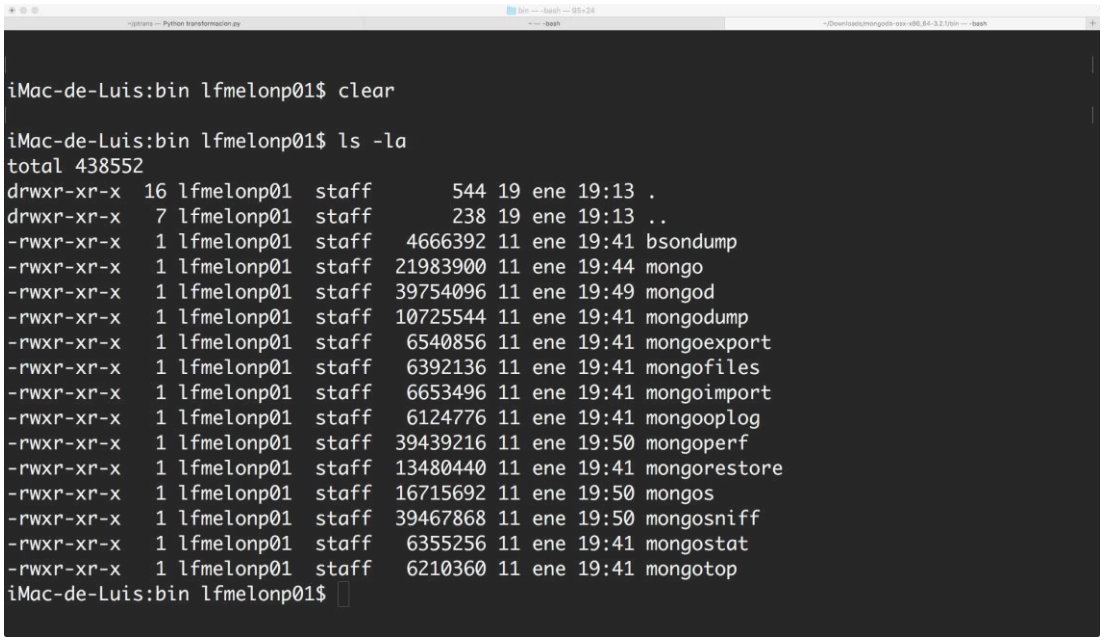

Figura 28 -. Archivos binarios de MongoDB

Si la salida que se obtiene del comando anterior es la observada en la figura 28, estaremos en disposición de preparar los directorios en los que se van a almacenar las bases de datos y las colecciones de información. En el caso contrario, se debe repetir la descarga de los ficheros y repetir el proceso.
Para crear los directorios en los que MongoDB va a almacenar las bases de datos y las colecciones ejecutaremos los siguientes comandos:

```
sudo mkdir –p /data/db
sudo chmod 777 /data
sudo chmod 777 /data/db
```

Para evitar realizar un constante acceso a la carpeta de instalación para poder levantar los servicios de MongoDB y acceder a la consola de la plataforma, en la ventana de terminal, y dentro de la carpeta ***bin***, escribiremos:

```
sudo cp * /usr/local/bin
```

Con el comando anterior le indicamos al sistema operativo que los diferentes ejecutables se encuentran almacenados en la carpeta *bin* del directorio de instalación, y como tal deben ser entendidos por la consola del sistema.

Comprobación del funcionamiento y creación de los índices

Para comprobar que el proceso de instalación ha ido correctamente, se escribirá en una consola el siguiente comando:

```
mongod
```

Si no hay ningún fallo, se obtendrá como salida el conjunto de mensajes indicado en la figura 29.

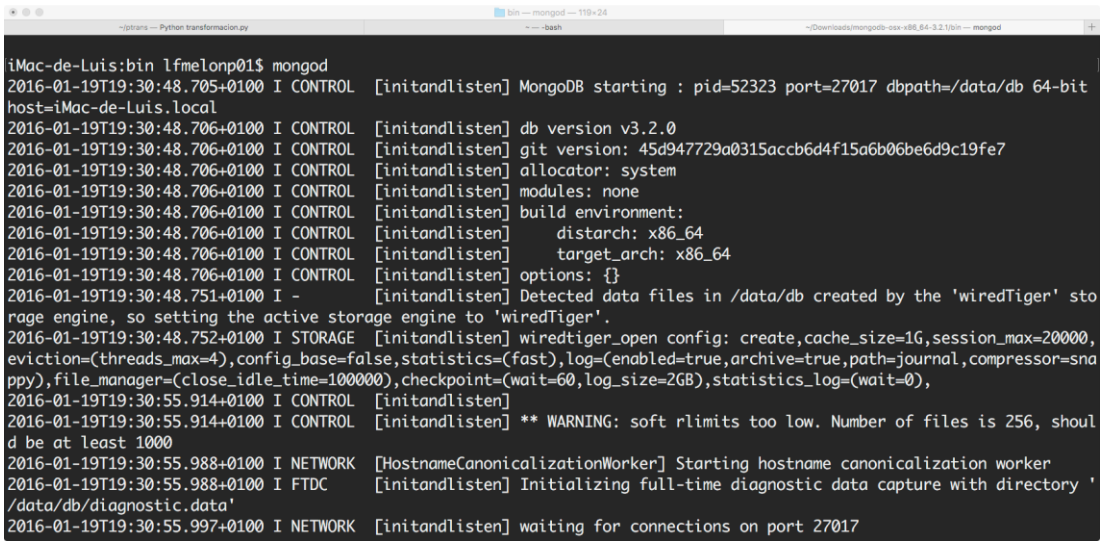

Figura 29 -. Comprobación de la correcta instalación de MongoDB

En este punto se abrirá una nueva pestaña en el terminal del sistema y podremos acceder a la consola de MongoDB con el comando ***mongo***. Se debería obtener la salida indicada en la figura 30.

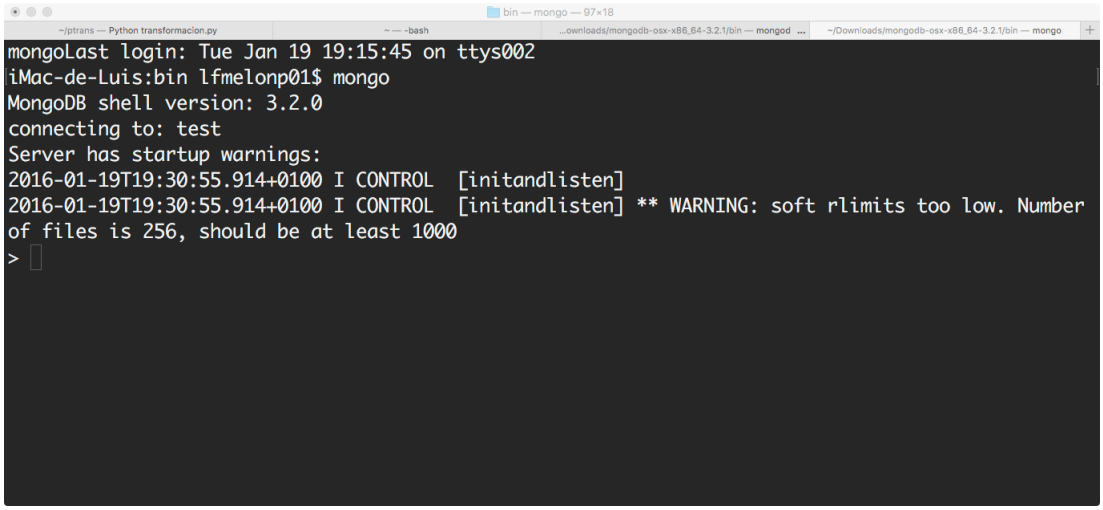

Figura 30 -. Acceso a la consola de MongoDB

Finalmente, escribiremos ***use tfg*** (base de datos creada en el apartado 3.2.4.1.1) y la consola devolverá el siguiente mensaje:

```
Switched to db tfg
```

Una vez haya sido seleccionada la base de datos, comenzaremos con la creación de los índices unidimensionales para que el acceso a los datos a través de la fecha sea eficiente. Para ello, es necesario indicar que el campo "time" sufrirá una indexación y que este índice tendrá una sola dimensión. La creación de los índices necesita que se le especifique a MongoDB la colección sobre la que trabajará el índice, que en nuestro caso se ha elegido la colección "a2014".

La creación del índice se realiza al invocar el comando siguiente en la ventana de terminal:

```
> db.a2014.createIndex({"time":1})
```

Si todo ha ido bien, el sistema indicará que la colección ha sido creada correctamente y mostrará información acerca de los índices existentes, antes y después de la invocación del comando, seguidos de un campo *ok* que especificará si la creación del índice no ha generado ningún problema, según se presenta en la figura 31.

```
{
        "createdCollectionAutomatically" : true,
        "numIndexesBefore" : 1,
        "numIndexesAfter" : 2,
        "ok" : 1
}
```

Figura 31 -. Resultado de la creación del índice temporal

Terminado este proceso, se pasará a la creación del índice más importante, aquél que va a optimizar los accesos a los datos a través de la localización geográfica de los mismos.

Para la creación de este tipo de índices es necesario indicarle a la consola cuál será el campo que albergará las coordenadas, en este caso ***loc***, además del tipo de índice, que en nuestro caso es ***2dsphere*** (sección 1, apartado 3, subapartado 2), en la forma que sigue:

```
> db.a2014.createIndex({loc:"2dsphere"})
```

De la misma manera que en la creación del índice anterior, obtendremos un mensaje parecido al de la figura 32, con la salvedad de que la generación automática de la colección esta vez no será cierta, ya que fue creada con la invocación al comando anterior:

```
{
        "createdCollectionAutomatically" : true,
        "numIndexesBefore" : 2,
        "numIndexesAfter" : 3,
        "ok" : 1
}
```

Figura 32 -. Resultado de la creación del índice geospacial

7.4 Instalación de Hadoop

Habiendo instalado previamente Homebrew, escribiremos en un terminal el siguiente comando:

```
brew install hadoop
```

Si todo ha ido correctamente, Hadoop se habrá instalado correctamente y la consola devolverá el siguiente mensaje:

```
Hadoop installed in the following directory
/usr/local/Cellar/hadoop
```

7.4.1 Configuración de Hadoop

Para poder utilizar Hadoop en nuestro desarrollo ha de procederse a una modificación previa de los archivos de configuración de la plataforma. Inicialmente, el programador debe acceder al siguiente directorio (utilizando el comando open en un terminal):

```
/usr/local/Cellar/hadoop/2.7.1/libexec/etc/hadoop
```

Dentro de este directorio podremos encontrar todos los archivos de configuración de Hadoop. En primer lugar, se deberá modificar el archivo **hadoop-env.sh** que contiene la información de configuración del ecosistema Hadoop. El programador debe encontrar la siguiente línea:

```
export HADOOP_OPTS="$HADOOP_OPTS -Djava.net.preferIPv4Stack=true"
```

Y cambiarla por la siguiente:

```
export HADOOP_OPTS="$HADOOP_OPTS -Djava.net.preferIPv4Stack=true -Djava.security.krb5.realm= -Djava.security.krb5.kdc="
```

Se guardarán los cambios en este archivo y se editará el fichero **core-site.xml**, que contiene todos los aspectos relacionados con la configuración del sistema de ficheros HDFS de la plataforma. Dentro de la etiqueta **<configuration></configuration>** se debe añadir:

```
<property>
    <name>hadoop.tmp.dir</name>
    <value>/usr/local/Cellar/hadoop/hdfs/tmp</value>
    <description>A base for other temporary directories.</description>
  </property>
  <property>
    <name>fs.default.name</name>
    <value>hdfs://localhost:9000</value>
  </property>
```

La siguiente de las modificaciones implica modificar el fichero **mapred-site.xml**, que contiene toda la información referente a los elementos de configuración de MapReduce. El programador deberá añadir las siguientes líneas al fichero que por defecto viene vacío.

```
<configuration>
        <property>
                <name>mapred.job.tracker</name>
                <value>localhost:9010</value>
        </property>
</configuration>
```

El último de los ficheros que debe ser alterado para el correcto funcionamiento de la plataforma es **hdfs-site.xml**, el cual contiene información acerca de la configuración particular de HDFS. Deben añadirse las siguientes líneas:

```
<configuration>
        <property>
                <name>dfs.replication</name>
                <value>1</value>
        </property>
</configuration>
```

En este punto, y para facilitar la ejecución de las órdenes que permiten conectar y desconectar los servicios que ofrece Hadoop, se abrirá un editor de texto y se escribirá:

```
alias hstart="/usr/local/Cellar/hadoop/2.7.1/sbin/start-
dfs.sh;/usr/local/Cellar/hadoop/2.7.1/sbin/start-yarn.sh"
alias hstop="/usr/local/Cellar/hadoop/2.7.1/sbin/stop-
yarn.sh;/usr/local/Cellar/hadoop/2.7.1/sbin/stop-dfs.sh"
```

Este fichero deberá ser guardado como un fichero oculto **.profile** y se ejecutará en el terminal la orden **source .profile** para que el sistema entienda que *hstart* y *hstop* son comandos ejecutables por consola.
Para poder ejecutar Hadoop han de facilitarse conexiones remotas en nuestro sistema operativo; para ello, se accederá a **Preferencias del Sistema → Compartiendo** y se marcará la opción **Sesión Remota**. Una vez hecho esto, se procederá a la creación de claves **ssh** para permitir el acceso remoto. Esto se realiza en la ventana de terminal escribiendo los siguientes comandos:

```
ssh-keygen -t rsa
cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

Por último, debemos darle forma a HDFS utilizando el comando:

```
hdfs namenode -format
```

Si todo ha ido bien, la consola devolverá lo indicado en la figura 33.

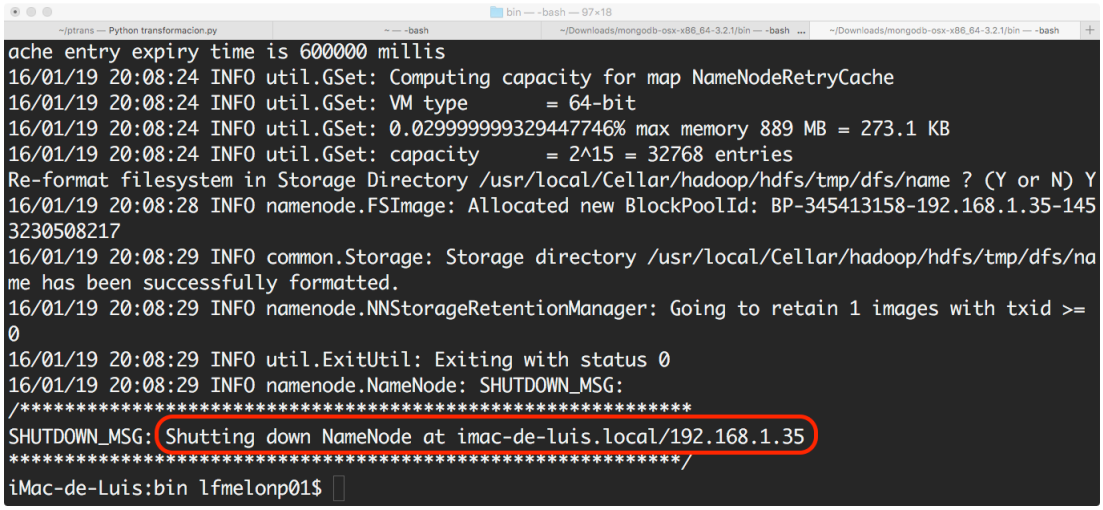

Figura 33 -. Resultado comando de formato de HDFS

Para comprobar la correcta ejecución de Hadoop, el programador puede probar a escribir en otra pestaña del terminal, o en otra ventana de terminal, el comando **hstart** y comprobar su correcta ejecución como muestra la figura 34.

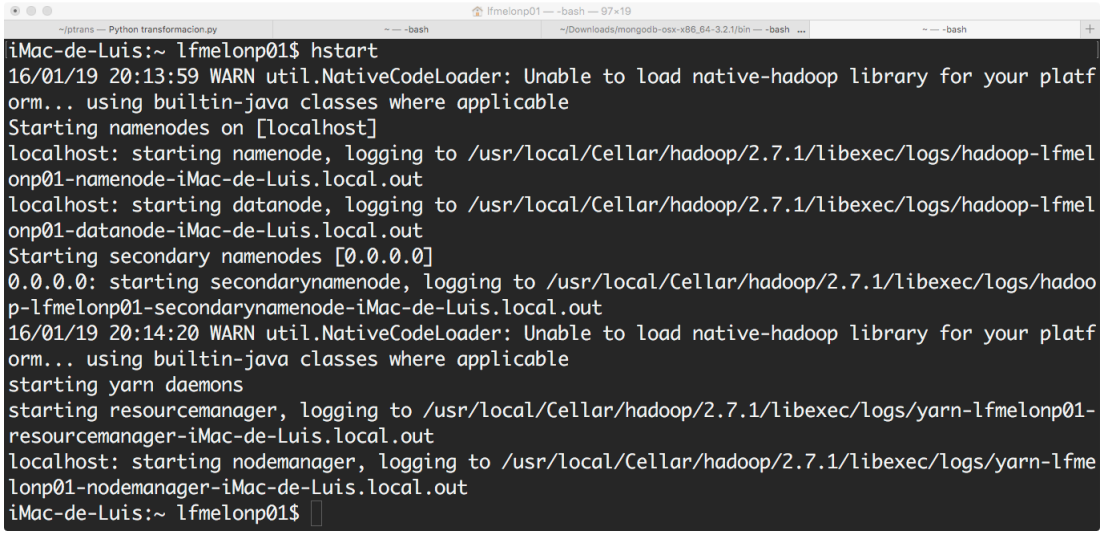

Figura 34 -. Resultado de invocar hstart

7.5 Instalación de Eclipse

7.5.1 Eclipse para MongoDB

Para obtener el IDE Eclipse que permita interactuar con la plataforma MongoDB, accederemos a la página [40] y se descargará la versión indicada en la figura 35.

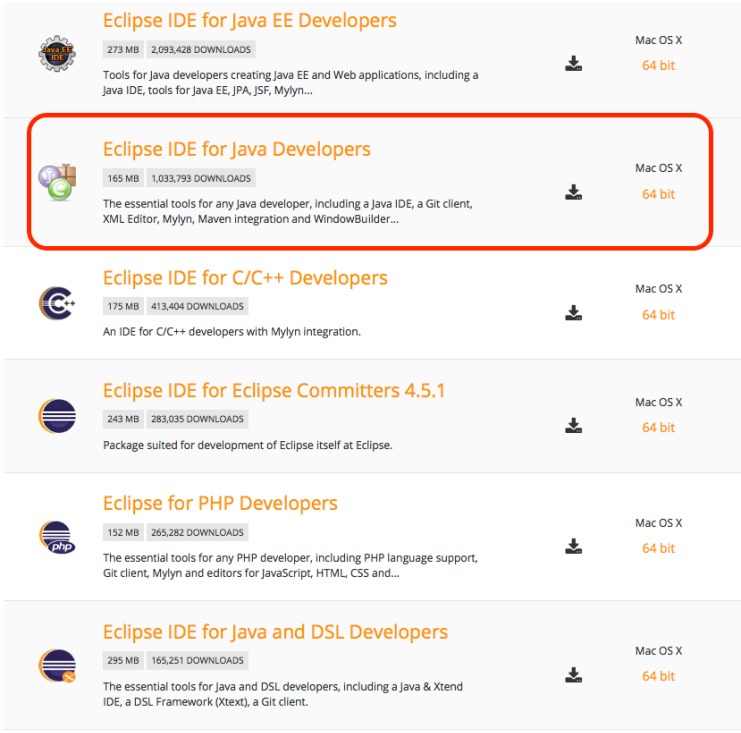

Figura 35 -. Eclipse Mars IDE Packages

Una vez terminada la descarga, bastará con descomprimir el archivo en el directorio que se crea oportuno.

7.5.1.1 Solucionando Errores: Importación de las librerías

Si a la hora de añadir un proyecto al entorno eclipse el programador se encuentra con problemas de importación de librerías, significa que falta añadir los archivos *JAR* que permiten trabajar con dichas librerías. En este caso, dentro de la barra de herramientas de Eclipse, deberemos acceder a **Project → Properties** y seleccionar la opción **Java Build Path** que podrá

verse en la parte izquierda del cuadro de diálogo. Con ello, llegaremos a una ventana como la indicada en la figura 36.

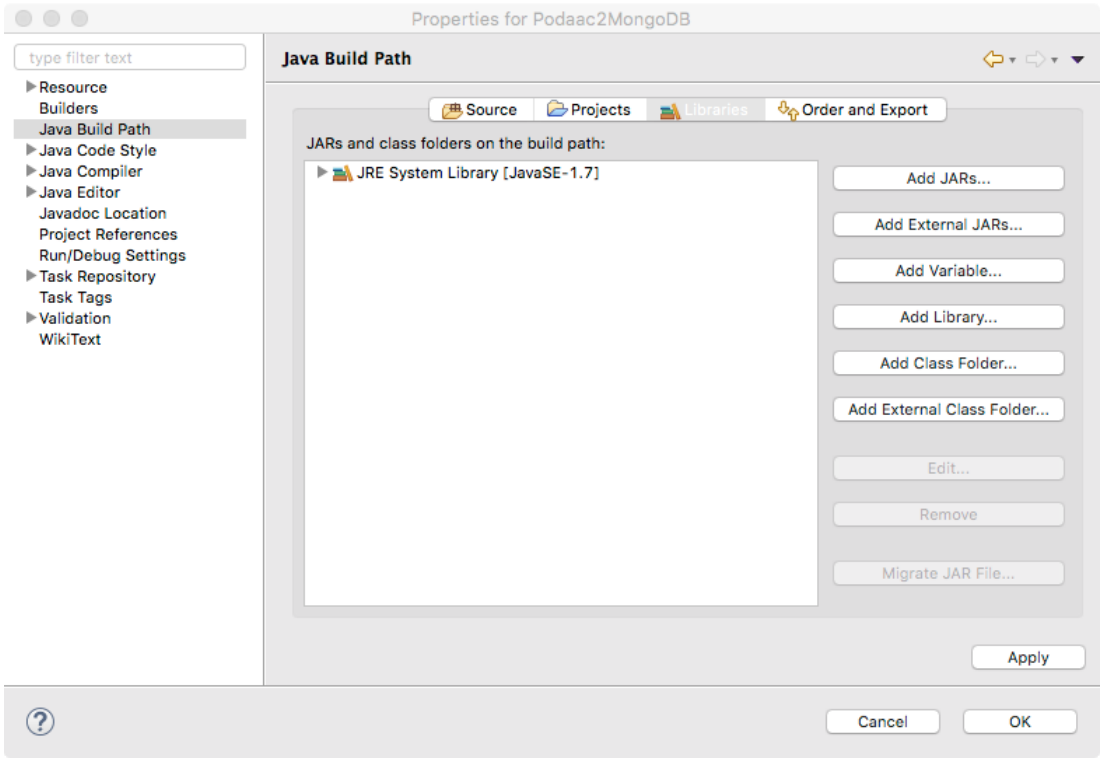

Figura 36 -. Ventana de propiedades del proyecto Podaac2Mongo en Eclipse

Pulsando el botón **AddJars…** aparecerá un cuadro de diálogo igual que el de la figura 37.

El programador debe seleccionar todos los JAR dentro de la carpeta **lib**, acepta la inserción de estos archivos y aplicar los cambios al proyecto, lo que deberá solventar los problemas de importación.

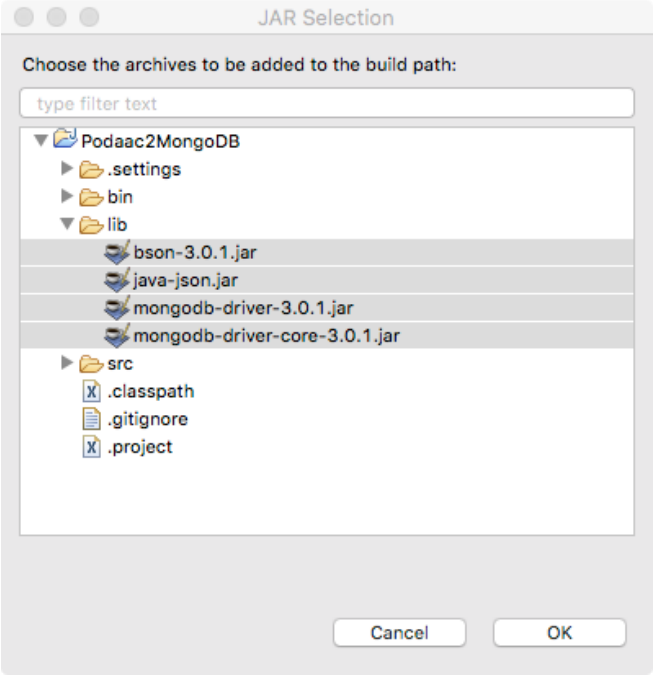

Figura 37 -. Cuadro de diálogo de selección de JARs

7.5.2 Eclipse para Hadoop

En primer lugar, ha de descargarse el entorno de desarrollo integrado en su versión ***Kepler***, ya que el *plugin* que se va a utilizar para conectar con Hadoop únicamente funciona en esta versión. Para ello se debe acceder a la página de descargas de *Eclipse Kepler* [54] y escoger el indicado en la figura 38.

Una vez finalizada la descarga, se descomprimirá el archivo en el directorio deseado por el programador.

Para configurar Eclipse Kepler y poder crear, editar y ejecutar proyecto de la plataforma Hadoop, también es necesario descargar otro *plugin*, que inicialmente se encontraba en la dirección [55], pero que durante el desarrollo del proyecto estos archivos fueron reubicados a la dirección [56] por decisión de sus desarrolladores.

Cuando termine la descarga del *plugin*, deberán copiarse todos los archivos del directorio *release* de la carpeta contenedora del *plugin* en el directorio de *plugins* de la carpeta de Eclipse Kepler.

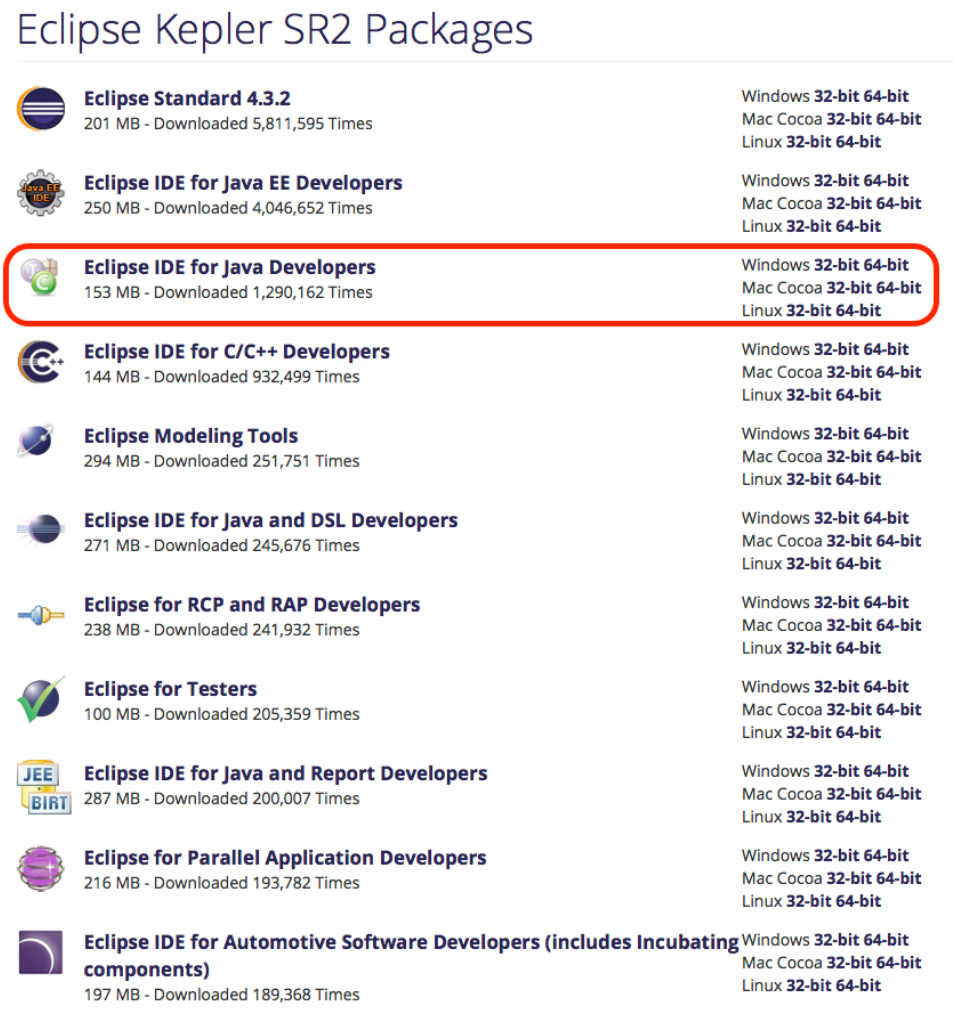

Figura 38 -. Eclipse Kepler IDE Packages

7.5.2.1 Configuración del Entorno Eclipse para Hadoop

Abrimos Eclipse Kepler y en la barra de herramientas se seleccionará **Window → Open Perspective -> Other… -> Map/Reduce**. Tras ello, en la parte inferior se podrá observar la pestaña **Map/Reduce Locations**; en este punto, debería verse algo parecido a lo que la figura 39 muestra.

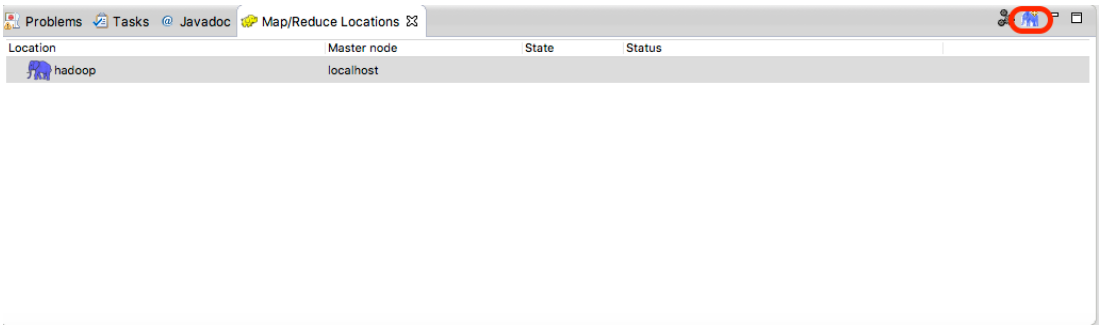

Figura 39 -. Ventana de Map/Reduce Locations en Eclipse

Si es así, estaremos en disposición de crear una nueva conexión Map/Reduce. Para ello, pulsamos el botón **New Hadoop Location**, apareciendo entonces el siguiente cuadro de diálogo, tal y como indica la figura 40.

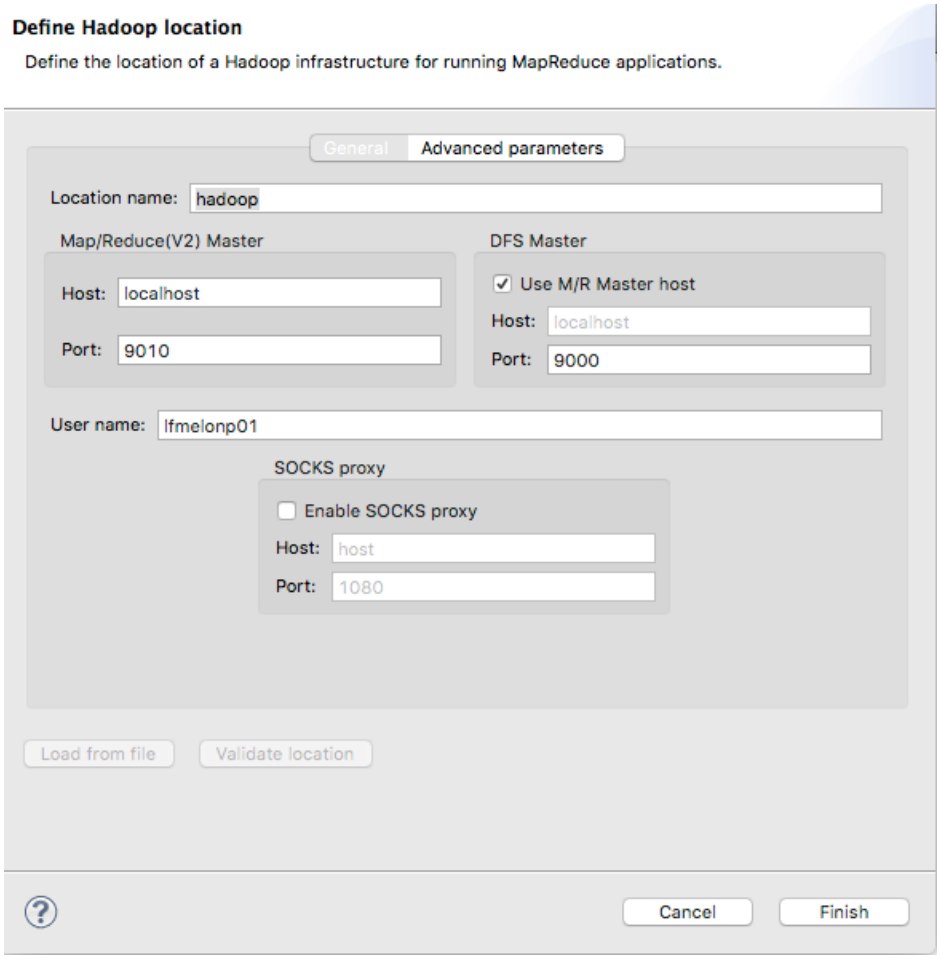

Figura 40 -. Creación de localización Hadoop

El programador añadirá los valores para cada uno de los campos seleccionables, según se muestra en la figura XX anterior. Posteriormente aceptará pulsando *finish*, tras lo cual se podrá acceder al sistema de ficheros de la plataforma, *una vez lanzados los procesos de Hadoop*.

La creación de proyecto se realizará a través de la opción **Create New Project → Map/Reduce Project**. En el cuadro de diálogo resultante, en el caso de que por defecto no aparezca, se tomará la opción *Use default Hadoop* y se indicará el directorio de instalación de la plataforma. Posteriormente se debe pulsar ***Configure Hadoop Install Directory*** y escribir la ruta en la que se encuentra la instalación de Hadoop; en nuestro caso: ***/usr/local/hadoop/2.7.1***.

Una vez finalizados los pasos indicados anteriormente, podremos implementar aplicaciones Map/Reduce sin ningún tipo de problemas añadido.

8 Anexo II: Manual de Usuario

8.1 Descarga de los Ficheros

Abierta la aplicación Filezilla, para la descarga de los ficheros mediante el uso del protocolo FTP, ha de introducirse la dirección del servidor en el cuadro de texto indicado como **Servidor** que aparece en la parte superior izquierda del programa, tal como indica la figura 41.

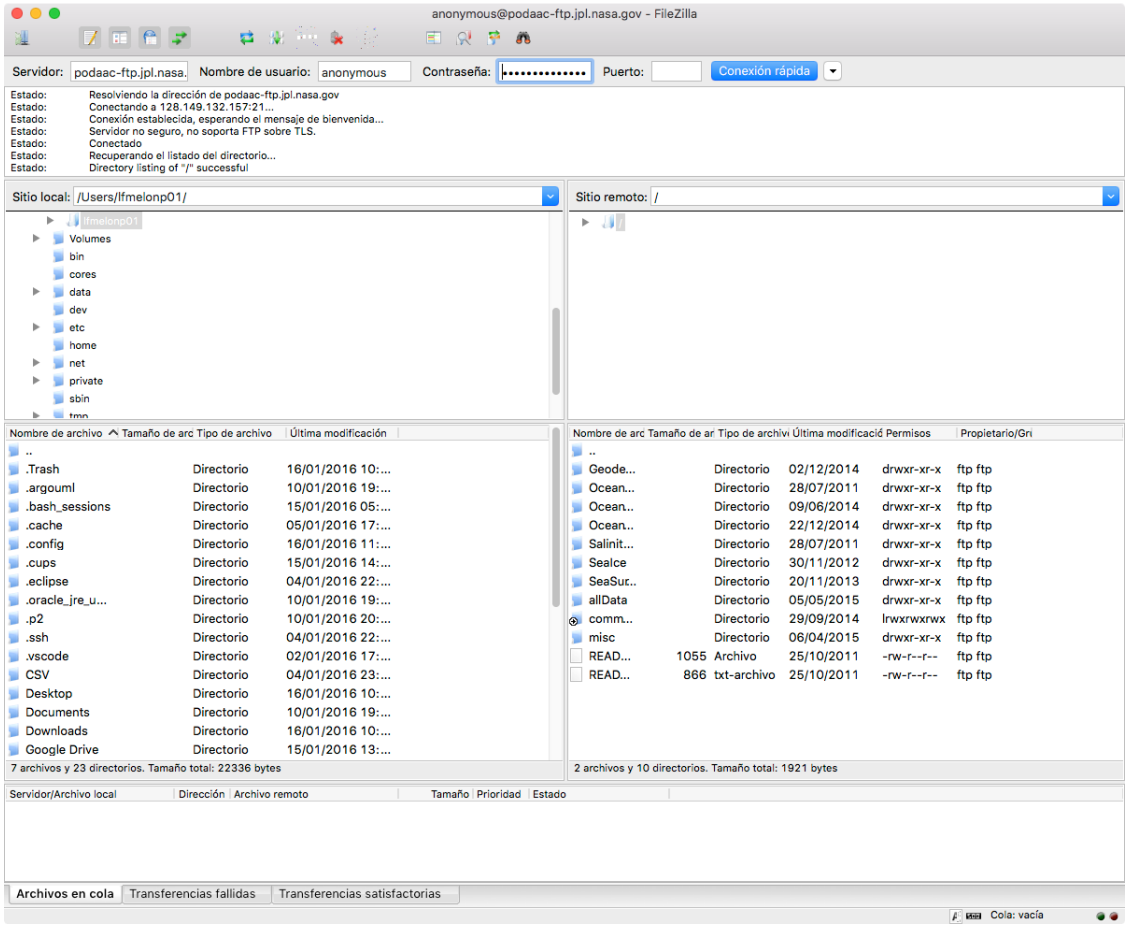

Figura 41 -. Aplicación Filezilla

Una vez se haya escrito la dirección del servidor, será necesario ingresar un usuario en su respectivo cuadro de texto; en este caso ***anonymous***. También es necesaria la dirección de correo electrónico como contraseña, según exige el protocolo PODAAC, que generalmente suele utilizarse ***blah@blah.blah***, como ha sucedido en nuestro caso. Posteriormente se deberá pulsar el botón ***Conexión Rápida***.

En la pestaña de *Sitio Remoto* podrá escribir la ruta para la descarga de los archivos deseados para ejecutar el sistema. Debido a que para este proyecto se han utilizado únicamente los del satélite RapidSCAT, la dirección de descarga será:
ftp://podaac-ftp.jpl.nasa.gov/allData/rapidscat/L2B12/v1.1/2014/

En el caso de querer utilizar los datos del satélite QuikSCAT podrá acceder a ellos desde la dirección de ejemplo:
ftp://podaac-ftp.jpl.nasa.gov/allData/quikscat/L2B/v2/2005/

8.2 Uso del programa de transformación

Para comenzar a utilizar el programa de transformación ha de estar instalada la versión 2.7 del lenguaje de programación Python.
El directorio que contenga el programa de transformación debe incluir también la carpeta “librerías”, que contiene los módulos adicionales en Python que se van a usar para la ejecución del *script* principal. Si se desea utilizar la conversión de datos procedentes del satélite QuikSCAT, además de las librerías indicadas anteriormente, esta carpeta debe contener el programa QSreadx64, que será ejecutado en caso de querer transformar datos cuya fuente es el satélite QuikSCAT.

8.2.1 Ejecución y menú principal

Para ejecutar el programa de transformación deberá acceder a la carpeta que contenga dicho programa y ejecutar en consola ***python transformacion.py***, o pulsar dos veces sobre el fichero.
Como se puede observar en la figura 42, la manera de escoger cada una de las opciones del menú es escribir el número de la orden y pulsar Enter.

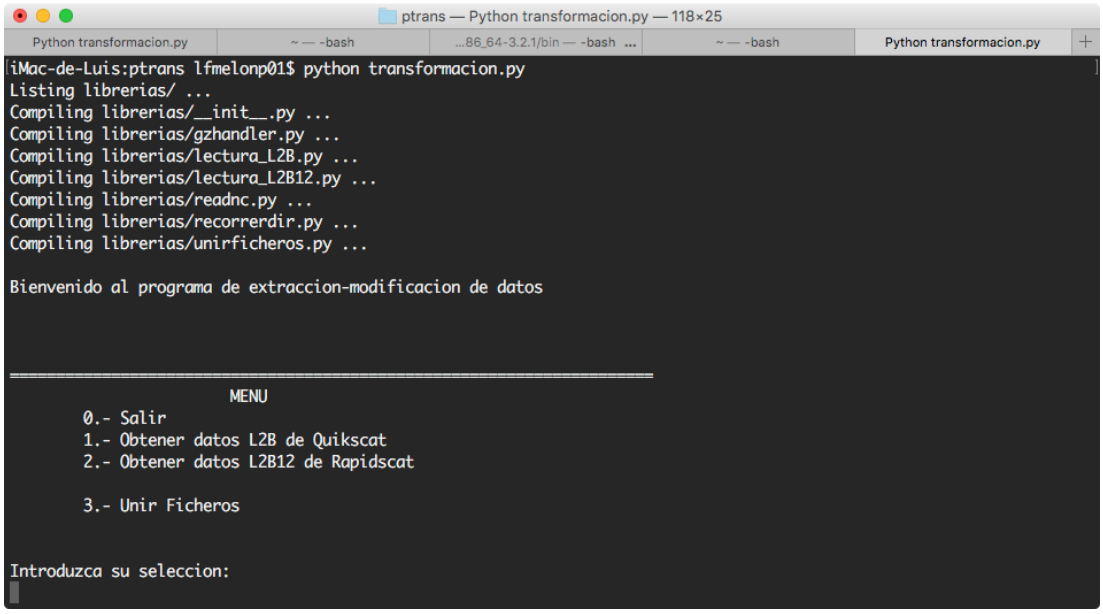

Figura 42 -. Menú del programa de transformación

8.2.2 Obtención de los datos de RapidSCAT

La transformación de los datos comprimidos provenientes del satélite RapidSCAT necesita tener los ficheros correspondientes a cada día en directorios separados dentro de la misma raíz; así, para la transformación de todos los datos de un año en RapidSCAT debe existir un directorio con tantas carpetas como días del año se quieran obtener, cada carpeta con el día juliano correspondiente, y en las cuales se almacenan los ficheros proporcionados por la PODAAC-NASA.

Tras acceder a la opción número 2 del menú principal, se le pedirá al usuario que introduzca la ruta de los ficheros comprimidos gz, así como la ruta en la que desea obtener lo ficheros finales. Si se introducen las carpetas correctamente, se realizará el proceso de transformación, en el que se imprimen mensajes informativos mientras el proceso se lleva a cabo. Al finalizar se informa del número total de ficheros procesados. El resultado debe ser un fichero CSV en la carpeta destino por cada fichero diario existente en la carpeta contenedora de los datos iniciales.

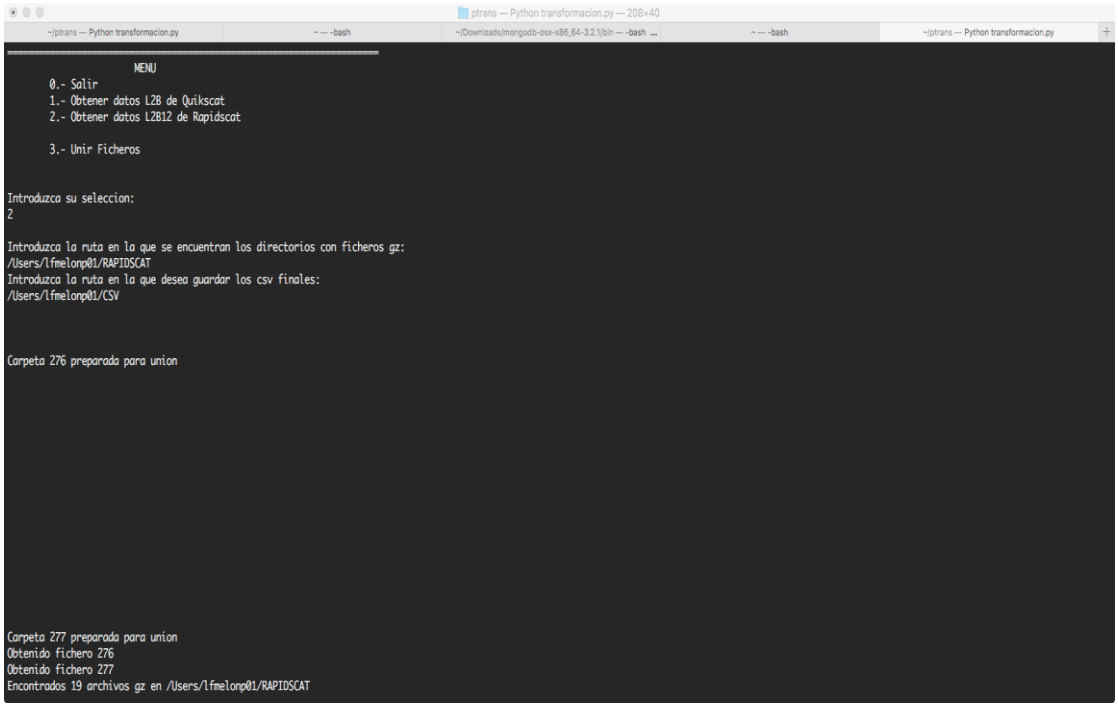

Figura 43 -. Resultado del programa de transformación para datos procedentes de RapidSCAT

8.2.3 Unión de los ficheros de datos

La opción número 3 del programa de transformación ha sido implementada por la problemática de la creación automática de ficheros ***.DS_STORE*** que cortan la ejecución normal del script bajo entornos de desarrollo Mac OS X.

Esta alternativa ofrece a los usuarios de este sistema operativo que sufran este problema la posibilidad de especificar el directorio en el cual están almacenados los ficheros parciales de la transformación, efectuar una unificación de éstos, y una copia al directorio de salida determinado. La salida que se obtendrá será igual que la de la figura 43 del apartado anterior.

8.3 Uso del programa para MongoDB

El programa desarrollado para interactuar con la base de datos MongoDB debe importarse como un proyecto de eclipse, según se especifica a continuación.

8.3.1 Importación de Proyecto a Eclipse

Para realizar una correcta importación de un proyecto eclipse es necesario, en primer lugar, haber descomprimido en la carpeta de *workspace*, creada por el entorno de desarrollo de manera automática, el proyecto en cuestión. Además, también es necesario mantener una versión de Eclipse ejecutándose.

Tras abrir el entorno de desarrollo, debe pulsar la opción ***File*** en la barra de herramientas del sistema y escoger la opción ***Import*** como puede verse en la figura 44.

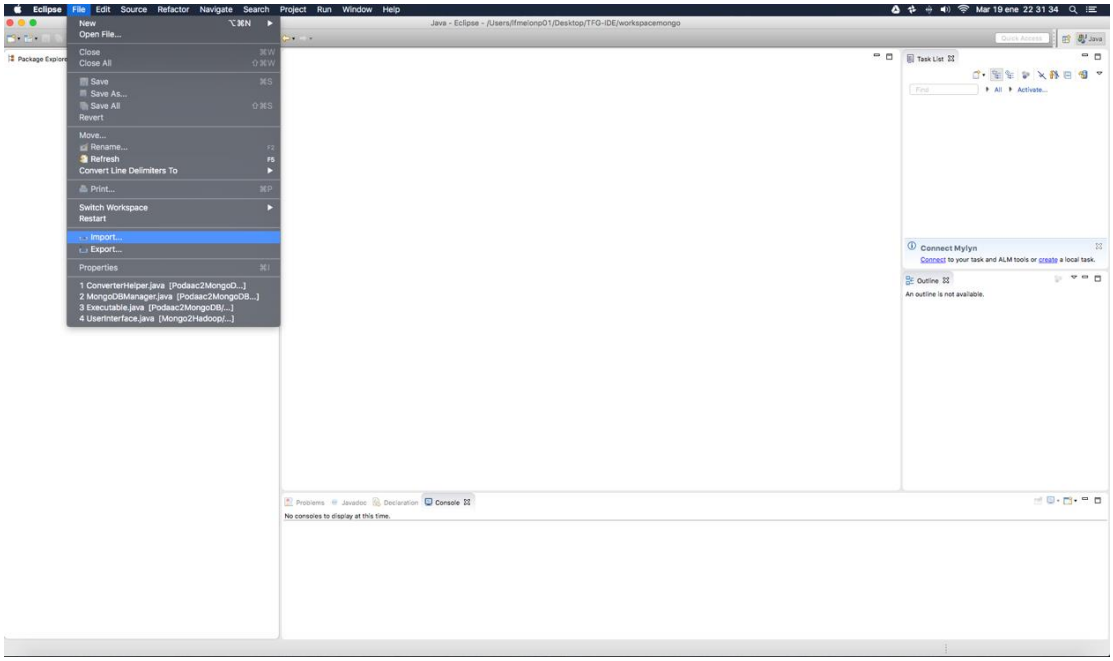

Figura 44 -. Importación de un proyecto a Eclipse

A continuación, se abrirá una ventana, como muestra la figura 45, en la que deberá escoger la opción ***Existing projects into Workspace*** y tendrá que pulsar ***Next***.

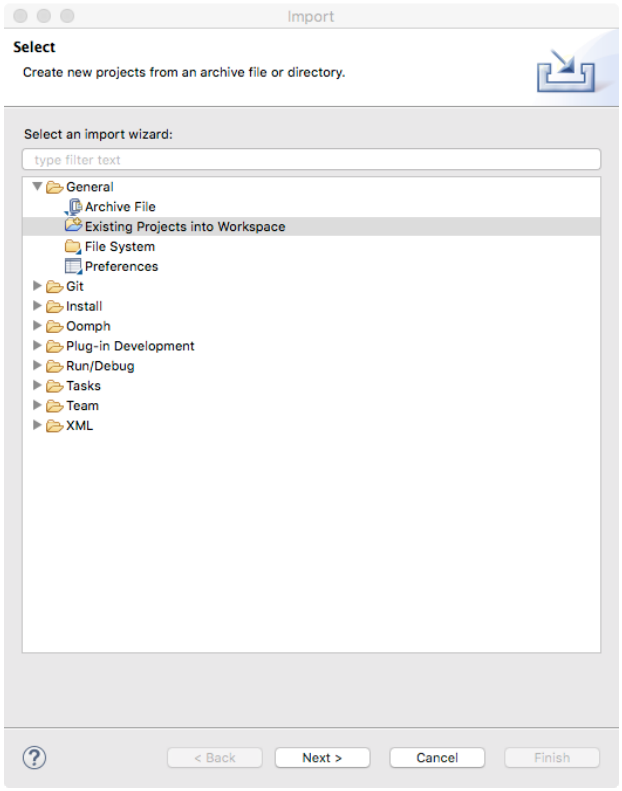

Figura 45 -. Elección del tipo de proyecto a importar a Eclipse

En la siguiente ventana deberá escoger la opción ***Browse*** para elegir el proyecto ***Podaac2Mongo*** existente en el *workspace* de la aplicación. Esta acción puede verse en la figura 46.

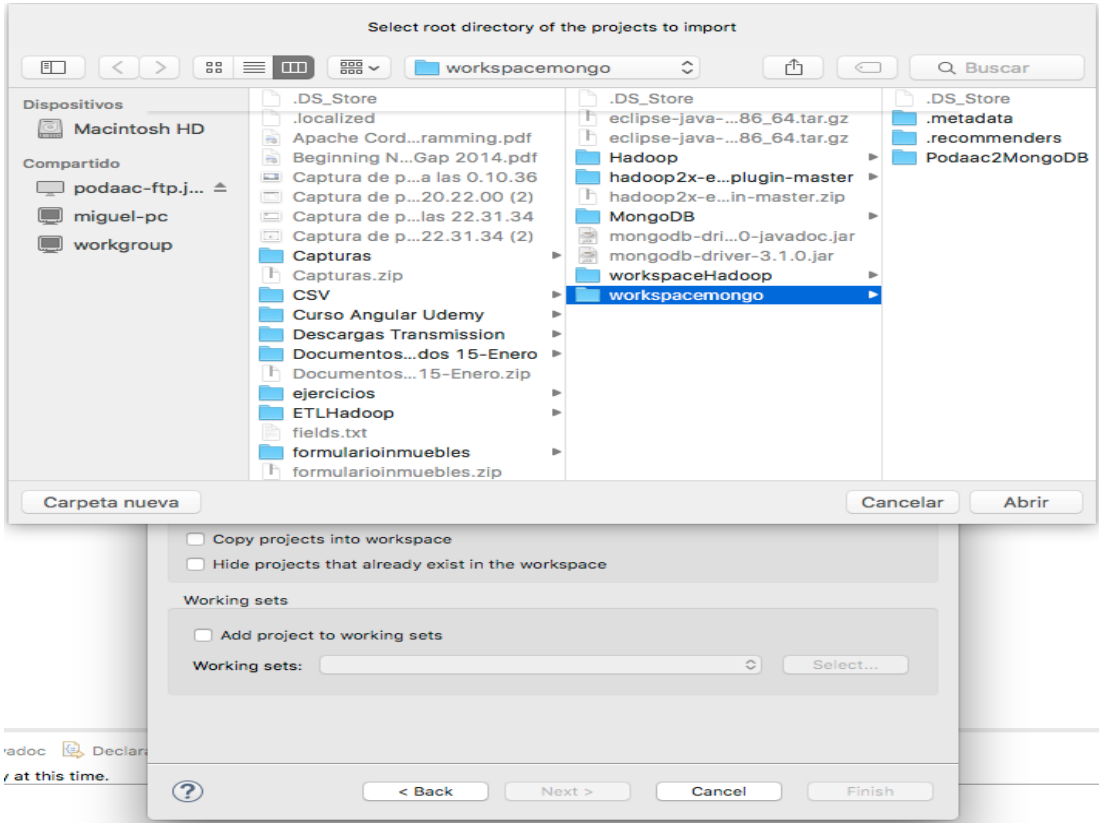

Figura 46 -. Selección del proyecto a importar

Finalizada la elección del proyecto, únicamente deberá pulsar el botón ***Finish*** y la importación habrá finalizado; entonces podrá ejecutarlo sin más problemas de los indicados en apartados anteriores.

8.3.2 Ejecución de la aplicación

Una vez el proyecto haya sido importado correctamente al entorno de desarrollo, deberá pulsar el botón de ejecución para arrancar el programa. Una alternativa es pulsar con el ***Click Derecho*** encima del proyecto y seleccionar la opción ***Run As.. → Java Application***.

Para facilitar la interacción de usuario con la base de datos, se ha diseñado un menú simple que aparece en la consola (véase la figura 47) y en el que se puede seleccionar distintas acciones, como cargar datos o hacer consultas, introduciendo el número correspondiente a la acción y pulsando la tecla Intro.

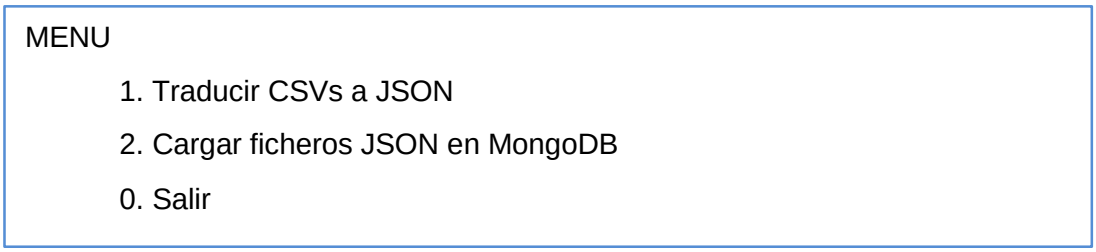

Figura 47 -. Menú de aplicación Podaac2Mongo

8.3.2.1 Traducción CSV a JSON

Antes de poder realizar la carga de datos en MongoDB es necesario transformar los ficheros que se desean importar y que han sido obtenidos como salida al programa de transformación. Esto es debido a la necesidad del sistema de tener un formato específico en los datos que van a ser indexados espacialmente por el campo “loc”. La figura XX nos presenta la bienvenida al asistente de traducción.

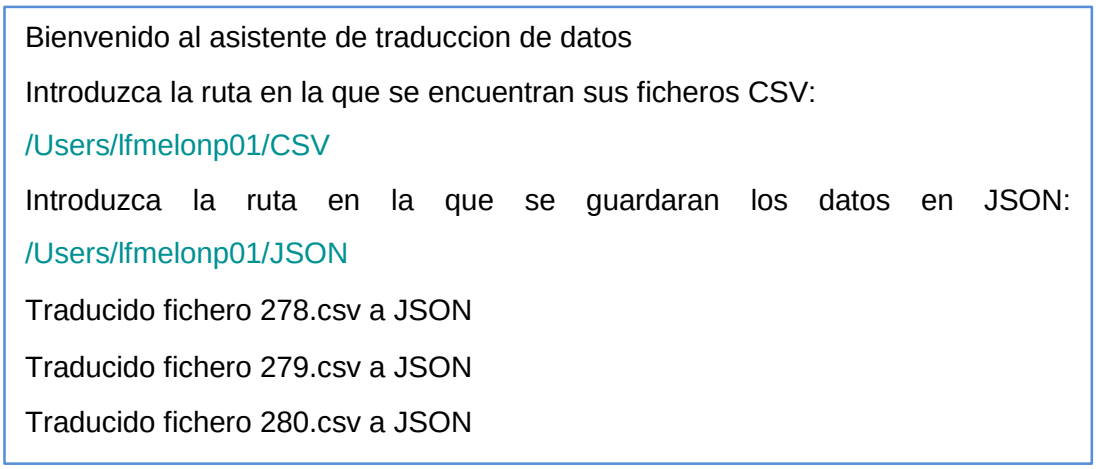

Figura 48 -. Asistente de traducción de datos Podaac2Mongo

Tras seleccionar la primera opción y pulsar el botón Intro, deberá indicar cuál es la ruta en la que se encuentra los ficheros CSV para cargar (en los que deberán estar todos los archivos que desee importar), así como la ruta de destino final.

Cuando haya introducido las rutas específicas, el programa traducirá cada uno de los ficheros de forma individual, mostrando un mensaje para cada uno de los ficheros traducidos.

8.3.2.2 Carga de ficheros JSON en MongoDB

El requisito previo para que esta parte del programa funcione correctamente consiste en que tenga almacenado en un directorio todos los ficheros con formato JSON que desee importar a la base de datos.

Tras seleccionar la opción 2 del menú principal del programa, deberá introducir la ruta en la que se encuentran los ficheros y, si ésta es correcta, deberá especificar el nombre de la base de datos a la que desea importar los archivos, así como el nombre de la colección que albergará la información.

Una vez hechos estos pasos, se realiza la inserción de tantos ficheros como haya en el directorio, imprimiéndose un mensaje informativo por cada uno de ellos y otro al finalizar el procedimiento, según puede verse en la figura 49.

```
Bienvenido al asistente de carga de datos en MongoDB
Introduzca la ruta en la que se encuentran sus ficheros JSON:
/Users/lfmelonp01/JSON
Introduzca el nombre de la Base de Datos en la que va a insertar los datos
tfg
Introduzca el nombre de la Coleccion de la Base de Datos en la que va a insertar los datos
a2014
Insertado 277.json en 1207 segundos
Insertado 278.json en 1347 segundos
Finalizada insercion, 3 ficheros insertados.
```

Figura 49 -. -. Asistente para la carga de datos en MongoDB

8.4 Uso del programa para la exportación de documentos en MongoDB

La utilización del programa de transformación de datos necesita la versión 2.7 del lenguaje de programación Python instalada.

Para utilizar el programa de transformación, deberá acceder a la carpeta que contenga dicho programa y ejecutar en consola ***python mongo2hadoop.py*** o pulsar dos veces sobre el fichero.

Se le mostrará el formulario que aparece en la figura 50, que deberá rellenar completamente para poder exportar los datos. Debe incidirse en la circunstancia de que los únicos campos que no tienen por qué ser seleccionados son los que aparecen bajo la etiqueta ***Valores a exportar (por defecto (time, longitude,latitude))***

Figura 50 -. Programa de exportación de MongoDB para Hadoop

Una vez se hayan insertado correctamente todos los valores, comenzará el proceso de exportación desde la base de datos.

Bibliografía

[1] IBM developerWorks. ¿Qué es Big Data?. https://www.ibm.com/developerworks/ssa/local/im/que-es-big-data/ (visitado en Diciembre de 2015)

[2] ADEOS II. Winds, Measuring Ocean Winds from Space. https://winds.jpl.nasa.gov/missions/seawinds/ (visitado en Agosto de 2015)

[3] Nasa Science. http://science.nasa.gov/ (visitado en Agosto de 2015)

[4] National Oceanic and Atmospheric Administration. http://www.noaa.gov/ (visitado en Agosto de 2015)

[5] Teaching and Training Resources for the Geoscience Community, “Ciclo de vida de las olas: generación”. http://www.meted.ucar.edu/marine/mod2_wlc_gen_ed1/print1.html (visitado en Diciembre de 2015)

[6] Nasa Science. Missions. QuikSCAT. http://science.nasa.gov/missions/quikscat/ (visitado en agosto de 2015)

[7] NSCAT. NASA Scatterometer. https://podaac.jpl.nasa.gov/NSCAT (visitado en Agosto de 2015)

[8] NASA. Imagen sintética del Satélite QuikSCAT. https://winds.jpl.nasa.gov/files/winds/QuikSCAT_high.jpg (visitado en Diciembre de 2015)

[9] Jet Propulsion Laboratory. California Institute of Technology. Measuring Ocean Winds from Space. QuikSCAT. https://winds.jpl.nasa.gov/missions/quikscat/ (visitado en Agosto de 2015)

[10] Wikipedia. The Free Encyclopedia. Scatterometer. https://en.wikipedia.org/wiki/Scatterometer (visitado en Agosto de 2015)

[11] PODAAC. Physical Oceanography DAAC. http://podaac.jpl.nasa.gov/ (visitado en Agosto de 2015)

[12] The HDF Group. The HDF levels of interaction. https://www.hdfgroup.org/products/hdf4/whatishdf.html (visitado en Septiembre de 2015)

[13] Jet Propulsion Laboratory. California Institute of Technology. Measuring Ocean Winds from Space. RapidSCAT. https://winds.jpl.nasa.gov/missions/RapidScat/ (visitado en Agosto de 2015)

[14] Unidata NetCDF. What is NetCDF? http://www.unidata.ucar.edu/software/netcdf/docs/ (visitado en Septiembre de 2015)

[15] Why NoSQL? Planet Cassandra. http://www.planetcassandra.org/what-is-nosql/ (visitado en Julio de 2015)

[16] Wikipedia. La enciclopedia libre. SQL. https://es.wikipedia.org/wiki/SQL (visitado en Enero de 2016)

[17] Wikipedia. La enciclopedia libre. NoSQL. https://es.wikipedia.org/wiki/NoSQL (visitado en Enero de 2016)

[18] Eric A. Brewer. Berkeley Education. https://www.cs.berkeley.edu/~brewer/ (visitado en Diciembre de 2015)

[19] InfoQ. CAP Twelve Years Later: How the “Rules” Have Changed. http://www.infoq.com/articles/cap-twelve-years-later-how-the-rules-have-changed (visitado en Julio de 2015)

[20] CAP. Figura del Teorema CAP. http://i.stack.imgur.com/a9hMn.png (visitado en Enero de 2016)

[21] MongoDB for GIANT IDEAS. NoSQL Database Types. https://www.mongodb.com/nosql-explained (visitado en Julio de 2015)

[22] MongoDB for GIANT IDEAS. MongoDB Fundamentals. https://docs.mongodb.org/manual/faq/fundamentals/ (visitado en Agosto de 2015)

[23] MongoDB for GIANT IDEAS. Data Modeling Introduction. https://docs.mongodb.org/v3.0/core/data-modeling-introduction/ (visitado en Agosto de 2015)

[24] Ilustración. Modelo de datos MongoDB vs Modelo de Datos Relacional. http://www.aaronstannard.com/images/mongo%20vs%20sql%20differences.png (visitado en Enero de 2016)

[25] MongoDB for GIANT IDEAS. Index Concepts. https://docs.mongodb.org/manual/core/indexes/ (visitado en Agosto de 2015)

[26] ReadWrite. Hadoop: What It Is And How It Works. http://readwrite.com/2013/05/23/hadoop-what-it-is-and-how-it-works (visitado en Agosto de 2015)

[27] Apache Software Fundation. Apache Hadoop. https://hadoop.apache.org/ (visitado en Septiembre de 2015)

[28] Hadoop. Figura del Ecosistema Hadoop. http://thebigdatablog.weebly.com/uploads/3/2/3/2/32326475/87375_orig.jpg (visitado en Enero de 2016)

[29] The Architecture of Open Source Applications. The Hadoop Distributed File System. http://www.aosabook.org/en/hdfs.html (visitado en Septiembre de 2015)

[30] Hadoop. Figura de la Arquitectura del Sistema de Ficheros Distribuidos Hadoop. http://bigdataweek.com/wp-content/uploads/2013/11/HDFS-Architecture.jpg (visitado en Enero de 2016)

[31] Reutter, J. *El modelo detrás de MapReduce*, Universidad Católica de Chile, https://www.ing.puc.cl/ciencia-de-la-computacion/wp-content/uploads/2015/01/juan-reutter_mapreduce.pdf (visitado en Septiembre de 2015).

[32] White, T. *Hadoop The Definitive Guide .* O'Reilly Media, Inc. (Abril 2015, 4th edition) (visitado en Septiembre de 2015)

[33] Rodríguez, F. *Mínería de Datos y Almacenes de Datos*. Grado en Ingeniería Informática en Ingeniería del Software. Universidad de Extremadura. (2014/2015) (visitado en Agosto de 2015)

[34] Wikipedia. The Free Encyclopedia. Comma-separated values. https://en.wikipedia.org/wiki/Comma-separated_values (visitado en Enero de 2016)

[35] Java. http://java.com/es/ (visitado en Agosto de 2015)

[36] Wiki Python. Tkinter. https://wiki.python.org/moin/TkInter (visitado en Diciembre de 2015)

[37] Python. Downloads. https://www.python.org/downloads/ (visitado en Agosto de 2015)
[38] MongoDB. Downloads. https://www.mongodb.org/downloads#production (visitado en Septiembre de 2015)
[39] Homebrew. The missing package manager for OS X. http://brew.sh/ (visitado en Septiembre de 2015)
[40] Eclipse. The Eclipse Fundation. Downloads. https://www.eclipse.org/downloads/ (visitado en Septiembre de 2015)
[41] Wikipedia. The Free Encyclopedia. NetCDF. https://en.wikipedia.org/wiki/NetCDF (visitado en Septiembre de 2015)
[42] Netcdf4-python. Python/numpy interace to netCDF. Download page. https://code.google.com/p/netcdf4-python/downloads/list (visitado en Septiembre de 2015)
[43] NumPy Scientific computing package. http://www.numpy.org/ (visitado en Septiembre de 2015)
[44] Sourceforge. Numerical Python. A package for scientific computing with Python. http://sourceforge.net/projects/numpy/files/NumPy/1.9.2/ (visitado en febrero de 2015)
[45] Wikipedia. La Enciclopedia Libre. File Transfer Protocol. https://es.wikipedia.org/wiki/File_Transfer_Protocol (visitado en Septiembre de 2015)
[46] Filezilla. The free FTP solution. https://filezilla-project.org/ (visitado en Agosto de 2015)
[47] Teomiro, D. Trabajo Fin De Grado. Integración de datos de vientos marinos en Oracle NoSQL. Universidad de Extremadura. (visitado en Septiembre de 2015)
[48] Misión QuikSCAT vía FTP. ftp://podaac-ftp.jpl.nasa.gov/OceanWinds/quikscat (visitado en Agosto de 2015)
[49] Misión RapidSCAT vía FTP. ftp://podaac-ftp.jpl.nasa.gov/OceanWinds/rapidscat (visitado en Agosto de 2015)
[50] Physical Oceanography Distributed Active Archive Center (PODAAC). Seawinds on RapidSCAT. Python Software. Módulo readnc. ftp://podaac-

ftp.jpl.nasa.gov/allData/rapidscat/L2B12/sw/Python/readnc.py (visitado en Agosto de 2015)

[51] Physical Oceanography Distributed Active Archive Center (PODAAC). Seawinds on QuikSCAT. C Software. ftp://podaac-ftp.jpl.nasa.gov/allData/quikscat/L2B/v2/sw/C/ (visitado en Agosto de 2015)

[52] Physical Oceanography Distributed Active Archive Center (PODAAC). Seawinds on RapidSCAT. Tools. http://podaac-tools.jpl.nasa.gov/hitide (visitado en Diciembre de 2015)

[53] Broder, A., Glassman, S.C., Manasse, M.S., & Zweig, G. “Syntactic clustering of the Web”. Computer Networks and ISDN Systems, 29(8-13), 1157-1166, 1997.

[54] Eclipse Kepler. The Eclipse Fundation. Downloads. https://eclipse.org/kepler/ (visitado en Octubre en 2015)

[55] Eclipse Kepler. Plugin Hadoop MapReduce (Deshabilitado). https://github.com/winghc/hadoop2x-eclipse- plugin/archive/master.zip (visitado en Octubre de 2015)

[56] Eclipse Kepler. Plugin Hadoop MapReduce (Habilitado). https://github.com/Luisfermp18/hadoopeclipesplugin (visitado en Enero de 2016)