UNIVERSIDAD DE EXTREMADURA

Escuela Politécnica

Grado en Ingeniería Informática en Ingeniería del Software

Trabajo Fin de Grado

Almacenamiento NoSQL de datos geoespaciales con Apache Cassandra

Alberto Caso Agúndez

Convocatoria: Enero, 2016

UNIVERSIDAD DE EXTREMADURA

Escuela Politécnica

Grado en Ingeniería Informática en Ingeniería del Software

Trabajo Fin de Grado

Almacenamiento NoSQL de datos geoespaciales con Apache Cassandra

Autor: Alberto Caso Agúndez

Fdo.:

Director: Félix Rodríguez Rodríguez
Fdo.:

Tribunal Calificador

Presidente: Antonio Polo Márquez

Fdo.:

Secretario: Mª Luisa Durán Martín-Merás

Fdo.:

Vocal: Miryam J. Salas Sánchez

Fdo.:

CALIFICACIÓN:

FECHA:

Índice

Índice Figuras

Índice Algoritmos

Resumen del trabajo

El objetivo de este Trabajo Fin de Grado consiste en almacenar datos espacio-temporales en una base de datos NoSQL para otorgar mayor eficiencia a las consultas realizadas por cualquier aplicación de análisis y minado de datos. En concreto, se ha seleccionado la base de datos Apache Cassandra (2015). Los datos espacio-temporales provienen de los capturados entre 1999 y 2009 por el satélite geoespacial QuikSCAT de la NASA (PODAAC.JPL.NASA, 2015c), ya fuera de uso, y de los proporcionados a partir de octubre de 2014 por un instrumento denominado RapidSCAT, también de la NASA (PODAAC.JPL.NASA, 2015e), actualmente en funcionamiento, alojado en la Estación Espacial Internacional (ISS) en sustitución del anterior satélite. Ambas fuentes nos aportan información meteorológica de la superficie de los océanos y mares de todo el planeta.

La información ofrecida por ambos satélites está determinada por la localización geográfica (latitud y longitud) y el momento concreto de la toma de datos: esta información es la que nos sitúa en el espacio y en el tiempo. Junto a la información espacio-temporal se incluye datos relevantes como son la dirección y velocidad del viento, y el ratio de precipitaciones, entre otros, proporcionando un total de 14 variables, en el caso del satélite QuikSCAT, y 22 con RapidSCAT. La información proporcionada es muy voluminosa.

Los datos se encuentran en formato netCDF4 (*Network Common Data Form*), un tipo de formato de datos científicos binarios organizados en matrices (UNIDATA.UCAR, 2015). Para permitir que las herramientas de análisis y minado exploten eficientemente estos datos geoespaciales, se extraen del repositorio FTP de la NASA y se transforman adecuadamente para luego ser almacenados en la base de datos NoSQL Apache Cassandra. La transformación, tanto en formato como en la selección de atributos de interés, genera ficheros intermedios CSV (*Comma Separated Values*) organizados por semanas de observación orbital, con idea de proporcionar una mayor sencillez en el proceso carga. La transformación se ha realizado en Python 2, versión 2.7 (Python, 2015).

Una vez obtenidos los diferentes archivos intermedios CSV, se organizan según la estructura que va a tener la base de datos NoSQL, de tal modo que las operaciones de consulta sean ágiles, computacionalmente hablando, y nos proporcionen la información lo antes posible. La carga de datos en la base de datos NoSQL Apache Cassandra se lleva a cabo mediante una aplicación específicamente construida en Java 8 (ORACLE, 2014).

La eficiencia de las consultas se mejora, además, con el uso de indexación externa. Con la ayuda de una estructura de indexación espacial R-tree (árbol R en su denominación en español) se asigna muy rápidamente un identificador único a cada una las diferentes zonas o regiones espaciales en las que se ha dividido el planeta. Es, pues, una transformación de dos dimensiones en una. Disminuyendo la dimensionalidad del espacio la base de datos NoSQL Apache Cassandra trabaja con mejor rendimiento, ya que por el momento esta base de datos no permite búsquedas eficientes basadas en más de una dimensión o clave.

Por último, en este trabajo también se ha incluido la visualización de las consultas realizadas sobre los datos geoespaciales almacenados en Apache Cassandra. Para ello, se ha construido mediante Java Servlets una página Web de acceso, consistente en un formulario para consultas espacio-temporales que considera latitudes, longitudes y tiempo de recogida de observaciones meteorológicas. La visualización de las respuestas proporcionadas por cada consulta utiliza los servicios Google Maps (2015) y Google Earth (2015), permitiendo al usuario interactuar con las salidas proporcionadas. Finalmente, la aplicación desarrollada ofrece la posibilidad de generar documentos de salida a partir de la cada consulta, tanto JSON (*JavaScript Object Notation*) como CSV, facilitando, así, trabajar con los datos consultados en infinidad de aplicaciones, ya que la mayoría soportan alguno, o ambos, de estos formatos.

Capítulo 1: Introducción

En el siglo en el que nos encontramos, la ciencia, y en especial las tecnologías, avanzan de una forma extraordinaria. Surgen, por tanto, nuevas técnicas de desarrollo, ya que existen otras necesidades; tal es así, que en la actualidad tiene una gran fuerza la domótica o también temas relacionados con el cambio climático, entre otros. Es por ello, que cada vez se recogen una mayor cantidad de datos para su posterior estudio.

Desde hace tiempo se ha intentado predecir los fenómenos meteorológicos en base a informaciones que puedan ser manejadas por criterios matemáticos y de manejo estadísticos. Todos los días tenemos ejemplos en los medios de esas predicciones que nos permite realizar las actividades cotidianas conforme al clima y la meteorología.

Este trabajo se ciñe en proporcionar técnicas adecuadas de almacenamiento y recuperación de datos cuya importancia es primordial para las predicciones de fenómenos futuros: los datos geoespaciales marinos. En concreto, se ha utilizado datos provenientes de las observaciones realizadas por satélites orbitales terrestres QuikSCAT y RapidSCAT sobre el viento cerca de la superficie de mares y océanos. La idea conlleva la formación de una base de datos adecuada que permita la obtención de manera precisa y rápida de toda la información existente relacionada con el viento marino y sus propiedades. El acceso adecuado y eficiente a esta información permitiría que muchas aplicaciones de análisis y minado de datos utilicen eficazmente estos datos para lograr su interpretación casi en tiempo real del comportamiento de estos fenómenos.

Los datos seleccionados son de gran precisión y aportan una amplia cobertura en todo momento para el estudio de fenómenos meteorológicos, como huracanes, tifones o ciclones (PODAAC.JPL.NASA, 2015d). El análisis de estos datos (y muchos otros existentes) lo que pretenden es evitar, en la medida de lo posible, las consecuencias devastadoras que producen, tanto pérdidas materiales como humanas. Es fácil comprender su importancia y utilidad si nos fijamos en ejemplos claros de

devastación reciente de estos fenómenos, como los ocurridos entre julio y octubre de 2005 con los huracanes Katrina, Emily, Rita o Wilma (Wikipedia, 2015d).

Proporcionar un almacenamiento y recuperación adecuado de estos datos para facilitar lo máximo posible el trabajo de las aplicaciones de análisis no es sencillo. Los datos geoespaciales no sólo provienen de fuentes distintas (en nuestro caso son dos homogéneas, ya que RapidSCAT deriva de QuikSCAT, pero podrían ser diferentes), sino que alcanzan volúmenes muy elevados, que no dejan de crecer y crecer, pues el satélite RapidSCAT está aún en funcionamiento, y cada día genera más observaciones. Y las aplicaciones de análisis y minado de estos datos pueden necesitar acceder a cualquiera de todo el volumen creciente de los datos existentes.

En la actualidad, existen múltiples bases de datos para el almacenamiento de la información. Las bases de datos son recursos que recopilan todo tipo de información para atender las necesidades de un amplio grupo de usuarios. Pero la alta especialización y las necesidades que se han ido produciendo en distintos campos han permitido desarrollar esta herramienta con unas especificaciones más concretas y definidas. Entre ellas, las más adecuadas para la gestión de los datos que nos ocupan en el presente proyecto, son las bases de datos NoSQL, de muy reciente aparición, que son aquellas las que por el momento están demostrando que mejor se adaptan al hecho de almacenar ingentes cantidades de datos de forma eficaz y sencilla, también para su recuperación. Son las bases de datos surgidas por la necesidad de tratar con **V**olúmenes enormes de datos, de gran **V**ariedad (diversidad) de fuentes, cuya **V**elocidad de recuperación es el factor primordial de consideración para que cualquier aplicación que las use. Son las 3 de las 4 uves básicas de la tecnología más reciente para el análisis de datos, tecnología conocida como Big Data. La cuarta uve es el **V**alor de la información generado por el resultado del análisis realizado sobre los datos almacenados.

A lo largo del resto del capítulo profundizaremos sobre la evolución de los sistemas de almacenamiento de la información, desde su origen hasta la aparición de las actuales bases de datos NoSQL. Primeramente, en la sección 1.1, analizaremos las bases de datos relacionales, y los problemas que plantean a los datos como los que en este proyecto se utilizan. Seguidamente, y con el fin de comprender mejor por

comparación la elección del almacenamiento NoSQL para este trabajo, en la sección 1.2 se describen los aspectos fundamentales de la tecnología NoSQL, además de perfilar el por qué las bases de datos relacionales no son las más adecuadas para este tipo de datos voluminosos y variables, como son los datos geoespaciales, a la hora de accederlos y analizarlos velozmente. En la sección 1.3, centramos la visión en Apache Cassandra, la base de datos NoSQL seleccionada para el almacenamiento de los geodatos. A continuación, en la sección 1.4, se explica más pormenorizadamente la naturaleza de los datos QuikSCAT y RapidSCAT, su origen y formato. Para finalizar el capítulo, la sección 1.5 presenta el diseño del índice espacial R-tree (árbol R) utilizado como acelerador del acceso a los datos sobre Apache Cassandra.

1.1. Bases de datos relacionales vs bases de datos NoSQL

1.1.1. Bases de datos relacionales

En los años 1980 surgen las bases de datos relacionales, que utilizan un modelo relacional, formado por una serie de tablas en las cuales se almacenan los datos. Las bases de datos relacionales han sido utilizadas, y lo siguen siendo, por la mayoría de empresas, ya que podemos acceder a los datos de una forma rápida y relativamente sencilla mediante el lenguaje llamado SQL (*Structured Query Language*). Estos accesos se realizan a partir de transacciones, que nos ofrecen la capacidad de no poder modificar los datos hasta que la transacción no haya finalizado (RIBW1314, 2015). Para llevarlas a cabo, las transacciones tienen que cumplir una serie de características conocidas por las siglas ACID, que se desglosa en (MDAD1415, 2015):

- Atomicidad: si una operación tiene más de un paso se ejecutan en su totalidad o no.
- Consistencia: está relacionado con la integridad, y nos asegura que solo se empieza aquello que se puede acabar, de tal manera que los datos no se cambien ni se deformen.
- Aislamiento: asegura que una operación no puede afectar a otras.

- Durabilidad: asegura que una vez realizada la operación, ésta persistirá y no se podrá deshacer.

1.1.2. Carencia de las bases de datos relacionales

Las bases de datos relacionales nos presentan una serie de problemas que son muy difíciles de resolver. Por ejemplo, si dos personas intentan acceder al mismo dato en una consulta podemos tener problemas, ya que si estamos consultando un dato que justamente se está modificando por otro usuario estaríamos obteniendo una reseña errónea. Esto, al principio de surgir las bases de datos relacionales, no era un problema porque era muy difícil que dos personas coincidieran a la vez y además realizaran la misma consulta, pero con el uso de Internet que ocurra esto ya no es tan difícil. Dicho problema está relacionado con la concurrencia de la información.

Por otro lado, utilizar un lenguaje específico para realizar el acceso a los datos supone un problema porque las bases de datos están incrustadas en programas y éstos, a su vez, utilizan otro lenguaje de programación: es el problema de la impedancia.

Y por último, las bases de datos relacionales usualmente se encuentran alojadas en máquinas "*Main Frame*"; es decir, centralizadas y ubicadas en una misma localización. Cuando surgieron estas bases de datos tampoco era un problema, ya que no estábamos hablando de una gran cantidad de información; pero, por ejemplo, en el caso de este trabajo, en el que estamos utilizando grandes series de datos de observación meteorológica (concretamente 58 millones de puntos de observación – registros de datos – cada semana, y durante más de 10 años), nos supone que la cantidad de información a almacenar es ingente y progresivamente creciente. Si todos estos datos los tuviéramos que almacenar en un único equipo, necesitaríamos un dispositivo con unas características muy definidas y de altas prestaciones que provocaría una gran inversión económica. Por ello, el crecimiento en el sentido de escalabilidad de esta unidad o "*Main Frame*" sería vertical, y conllevaría un aumento de la potencia, el almacenamiento y la memoria del equipo (Fowler, M., 2015).

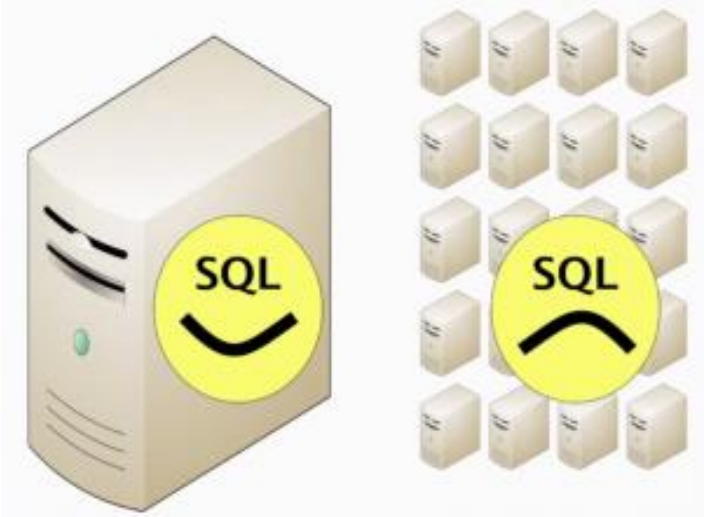


Figura 1: Bases de datos relacionales MainFrame (Fowler, M., 2015)

Otra de las carencias presentadas por las bases de datos relacionales es que los datos almacenados se tienen que encontrar completamente estructurados, es decir, los datos están fuertemente "tipados", y poseen un estilo rígido. Para el caso de la realización de este proyecto, la estructuración homogénea de los datos es un gran problema, ya que debe preverse el almacenamiento de los datos de fuentes u orígenes muy diversos (provienen de distintos satélites orbitales, o naves de reconocimiento, cada uno con su diseño de datos y fichero diferenciado); es decir, de naturaleza heterogénea. La integración de datos heterogéneos en una base de datos relacional sigue siendo un problema abierto al cual los investigadores siguen dedicándole esfuerzo y tiempo. Así pues, el uso de una base de datos relacional no sería, por tanto, adecuada para nuestros datos geoespaciales.

1.1.3. Bases de datos NoSQL

Debido a los problemas expuestos anteriormente con el uso de bases de datos relaciones, y dada la expansión creciente y exponencial de Internet, con un aumento progresivo de la cantidad de tráfico de datos y de su producción, empresas como Google, Amazon, Facebook o Twitter crean sus propios sistemas de almacenamiento, dando origen a las denominadas bases de datos NoSQL.

La sencillez es una de las premisas de las bases de datos NoSQL. Generalmente, no llevan a cabo un modelo para su diseño como era el caso de las bases de datos relacionales, sino que el diseño del almacenamiento se basa en la forma de las respuestas a consultas. El diseño de los datos no está fuertemente estructurado; es más bien semi-estructurado e incluso carente de estructura. Este hecho facilita poder integrar datos de provenientes de diferentes fuentes. Así, por ejemplo, en este proyecto se utiliza la misma base de datos para los datos pertenecientes a dos satélites diferentes, QuikSCAT y RapidSCAT, aunque de diseño muy cercano, pero distintos. Y lo que es más importante, también permite almacenar otros datos de otras fuentes de diferente formato y contenido.

Otra característica relevante de las bases de datos NoSQL es el crecimiento horizontal en relación con la escalabilidad; es decir, permiten utilizar diferentes equipos con unas características medias de potencia, almacenamiento y memoria, formando un "*cluster*". NoSQL son, pues, bases de datos distribuidas. Al utilizar equipos pequeños conectados, los caros sistemas de altas prestaciones ya no son una exigencia.

Y por último, el problema de la impedancia se mitiga un poco con la elección de lenguajes de consulta muy sencillos, de poco peso sintáctico, para las bases de datos NoSQL. No utilizan el lenguaje SQL, sino que algunas implementaciones utilizan llamadas a procesos de recuperación y almacenamiento propio, o bien, caso de otras, utilizan un lenguaje muy reducido de sintaxis similar a SQL con el fin de facilitar su aprendizaje, como ocurre en el caso de Apache Cassandra con el lenguaje CQL (*Cassandra Query Language*). Al hablar de lenguaje de acceso sencillo o reducido, nos referimos a que no permiten consultas de complejidad elevada, como ocurre con las uniones o *Join* de tablas y las subconsultas de SQL.

Estas bases de datos NoSQL son el resultado, como ya hemos introducido anteriormente, de la aplicación de técnicas dirigidas a dar respuesta al surgimiento del "*Big Data*"; en concreto, para dar respuesta al enorme volumen de almacenamiento. En contraposición a las propiedades ACID de las bases de datos (sección 1.1.1), las bases de datos NoSQL promueven una serie de características

con las siglas BASE (contraposición hasta en el nombre dato) que consisten en (Dataversity, 2015):

- Basic Availability: quiere decir que se centra en la disponibilidad de los datos incluso con la presencia de fallos; es decir, la gestión de las base de datos está altamente distribuida. Por lo tanto, al existir la replicación, los datos se propagan en diferentes nodos y si algún nodo está caído, otro contendrá su información.
- Soft-State: las bases de datos con esta característica abandonan los requisitos de coherencia de las bases de datos relacionales, ya que se considera que la consistencia de los datos es un problema de los desarrolladores y no de las bases de datos.
- Eventual Consistency: en las bases de datos NoSQL la consistencia se realizará en algún momento del futuro pero no se sabe cuándo ocurrirá.

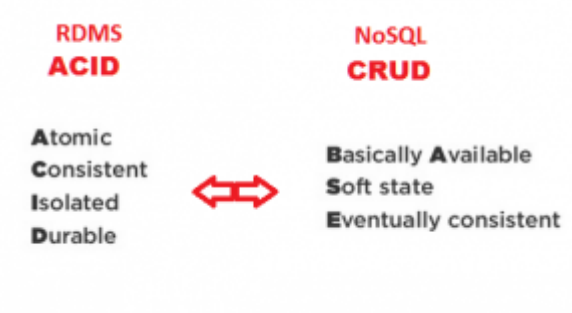


Figura 2: ACID vs BASE (Media.Licdn, 2015)

1.1.3.1 Teorema CAP

Este teorema presentado por Eric Brewer, de la Universidad de Berkeley, nos enuncia que es imposible garantizar la consistencia, la disponibilidad y la tolerancia a fallos, a la vez en un sistema distribuido. Las siglas CAP corresponden a:

- Consistencia (Consistency): se refiere a que cuando se realiza una consulta siempre se tiene que recibir la misma información.
- Disponibilidad (Availability): todos los clientes pueden escribir y leer aunque algún nodo de la base de datos este caído.

- Tolerancia a Fallos (Partition Tolerance): el sistema gestor de base de datos tiene que funcionar aunque exista algún tipo de fallo o alguno de sus nodos este caído.

Como consecuencia de este teorema, podemos dividir las bases de datos NoSQL en 3 tipos:

* En primer lugar podemos cumplir las condiciones CA (Consistencia y disponibilidad) provocando así que no exista tolerancia a fallos, este es el caso de las bases de datos relacionales como pueden ser MySQL, Oracle, SQL Server…etc.

* En segundo lugar, cuando las condiciones son CP (Consistencia y tolerancia a fallos), excluye la disponibilidad, en este caso podemos hablar de bases de datos NoSQL como MongoDB o HBase.

* Y por último, cuando se cumplen las condiciones AP (Disponibilidad y tolerancia a fallos) descartan la opción de consistencia, como es el caso de Apache Cassandra, que es la herramienta que vamos a utilizar durante en el proyecto (LNDS, 2015).

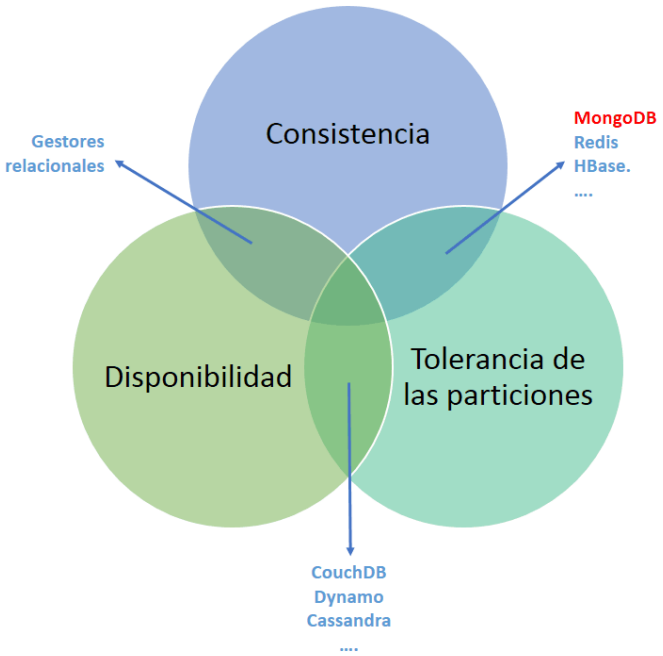


Figura 3: Teorema CAP (CAMPUSMVP, 2015)

La decisión de utilizar Apache Cassandra es porque se obtienen una gran cantidad de datos próximos en distancia (12 kilómetros y medio), dando más importancia a la disponibilidad y a la tolerancia a fallos, con respecto a la consistencia.

Como podemos observar, dependiendo de las necesidades de nuestro proyecto podemos usar una herramienta u otra. Por ejemplo, si se trata de un banco siempre se tiene que cumplir la consistencia y la disponibilidad, por ello tenemos que utilizar las bases de datos relaciones, en cambio si queremos realizar una página web de ventas de productos podríamos utilizar cualquiera de las otras dos opciones.

1.1.3.2. Tipos de bases de datos NoSQL

Aparte de las diferencias de las que ya hemos hablado con el teorema CAP, existe otro tipo de clasificación de las bases de datos NoSQL, que se pueden dividir en cuatro grupos basándonos en el modelo de datos utilizado (Acens, 2015):

- Clave-valor: es la base de dato más sencilla y popular. En este sistema cada elemento se encuentra identificado por una clave única, lo que nos permite obtener los datos de una forma muy rápida. Dentro de este grupo se encuentra Apache Cassandra, con la cual realizamos el presente proyecto.
- Documentales: en ellas la información se almacena en estructuras denominadas documentos, los cuales son estructuras JSON o XML y también se utiliza una clave única para cada registro; por lo que podemos afirmar que se trata de una subclase de las bases de datos clave-valor.
- Grafos: en este tipo de bases de datos la información se representa en los nodos de un grafo y sus relaciones con las aristas de dichos grafos, de tal forma que se pueden utilizar algoritmos para grafos a la hora de recorrerlos. Son muy útiles para guardar información con muchas relaciones, como en las redes y conexiones sociales.
- Orientadas a columnas: funcionan de forma parecida a las bases de datos relacionales, pero almacenando columnas de datos en lugar de registros, de tal forma que están organizadas en columnas en lugar de por filas.

1.2. Apache Cassandra

Apache Cassandra es de una base de datos NoSQL de código abierto, distribuida y construida para el manejo de un gran volumen de datos. Se utiliza cuando se necesita escalabilidad y alta disponibilidad sin comprometer el rendimiento. La disponibilidad se produce gracias al diseño de anillo que tienen los nodos, en los cuales se encuentran los datos. Los nodos pueden estar repartidos por diferentes zonas geográficas.

En el diseño en anillo todos los nodos tienen el mismo papel; es decir, no existe un nodo maestro, por lo que todos los nodos se comunican entre sí por igual. Gracias a ello cumple el teorema CAP, en concreto AP (Disponibilidad y tolerancia de las Particiones o fallos), provocando que todos los nodos pueden atender a diferentes peticiones.

Figura 4: Arquitectura Apache Cassandra en anillo (GoogleUserContent, 2015)

Cassandra se creó mediante un proyecto de ingenieros de la red social Facebook (Wikipedia, 2015c) en el año 2008, debido a los problemas encontrados a la hora de realizar búsquedas en su plataforma. Posteriormente la han añadido bajo licencia Open Source como uno más de los proyectos Apache (PlanetCassandra.org, 2015).

1.2.1. Características fundamentales

Apache Cassandra ofrece una serie de características destacables que explicaremos a continuación (Academy Datastax, 2015a):

- En su diseño todos los nodos son iguales, de tal forma que todos pueden leer y escribir, proporcionando simplicidad operativa y una escalabilidad horizontal sencilla.
- Posibilidad de añadir nuevo nodos sin tener que hacer caer el *cluster* primero.
- Nodos que han fallado pueden ser fácilmente restaurados o reemplazados.
- Modelo de datos flexible y dinámico que nos permite realizar rápidas escrituras y lecturas.
- Utilización del lenguaje CQL (Cassandra Query Language) para realizar las consultas, actualizaciones, borrados e inserciones de los datos. Es similar al lenguaje SQL de las bases de datos relacionales.
- Replicación múltiple de los datos, que consiste en realizar copias de los datos en los diferentes nodos. Esta replicación se lleva a cabo mediante el usuario que indicará el número de réplicas que necesite.

1.2.2. Modelo de datos

Como ya hemos comentado en el apartado 1.1.2, en las bases de datos NoSQL no existe un modelo de datos fijo, sino que es flexible y dinámico, y, cómo no iba a ser de otra forma, ocurre lo mismo con Apache Cassandra. Conlleva que la estructura creada para una base de datos depende del tipo de consultas que se vayan a realizar.

También hemos comentado en el apartado 1.1.3.2 que Apache Cassandra podemos catalogarla como una base de datos Clave-valor; por ello utiliza los siguientes conceptos:

- *Column*: es el nivel más bajo que hay, se trata de una estructura con 3 campos, que son: el nombre de la columna, el valor que tiene y el

timestamp. El *timestamp* contiene el momento (fecha, hora en milisegundos) en la cual se ha realizado la inserción.

- *Supercolumn*: es un conjunto de *Column*s.
- *ColumnFamily*: se trata de un conjunto de *Column*s ordenadas, de tal manera que cada fila contiene una clave. Las *ColumnFamily* se pueden definir como las tablas de las bases de datos relacionales.
- *Keyspace*: es el nivel más alto en el modelo de datos. La *keyspace* contiene todas las familias de columnas. Para que nos resulte más sencillo de entender, podemos compararlo con la base de datos en el modelo relacional.

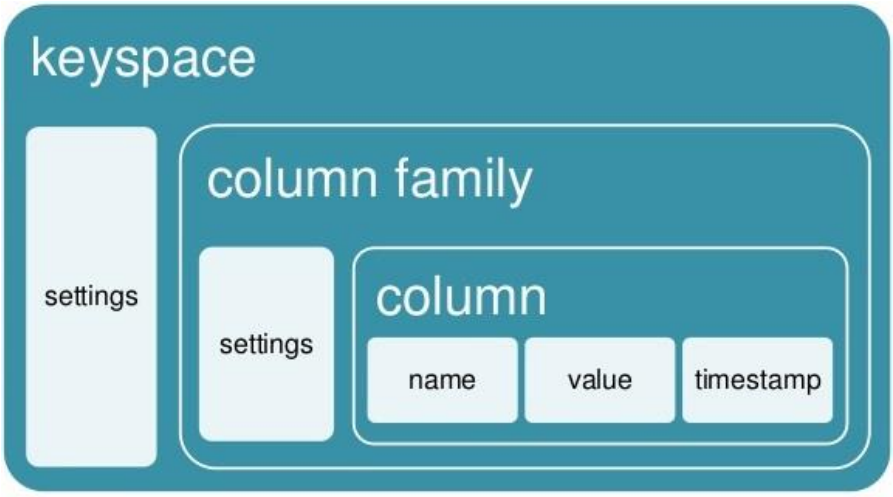


Figura 5: Modelo de datos en Apache Cassandra (Saxena, U., 2015)

Como podemos observar en la figura 5, las keyspace están formadas por ColumnFamily y estas a su vez por Column (Hewitt, E., 2011).

1.2.3. CQL

CQL (*Cassandra Query Language*) es el lenguaje de comunicación con la base de datos Apache Cassandra. Para poder interactuar con él accedemos al terminal (*shell*) CQL que recibe el nombre *cqlsh*, y que se encuentra en la carpeta *bin* de la carpeta de Cassandra.

Con este lenguaje podemos realizar diferentes operaciones como pueden ser insertar datos en una *ColumnFamily*, crear una *Keyspace* o *ColumnFamily* con sus diferentes *Column*. También podemos modificar datos, borrarlos o consultarlos.

Aunque es un lenguaje bastante parecido a SQL en sintaxis, la ejecución de las consultas no se parece en nada (Doc.datastax, 2015).

1.3. OceanWinds con QuikSCAT

El satélite *Quik Scatterometer,* o QuikSCAT, construido por un consorcio entre la NASA y las agencias espaciales alemana e italiana, se ha encargado durante algo más de una década a recoger datos de vientos marinos. Fue lanzado el 19 de junio de 1999 mediante el cohete Titan II (Wikipedia, 2015e) desde California, con la misión de ocupar el vacío que había dejado anteriormente NSCAT (NASA Scatterometer) (PODAAC.JPL.NASA, 2015b), que se encargaba también de las observaciones del viento de la superficie marina; pero falló de forma inesperada en junio de 1997. El satélite se diseñó como una misión de rápida recuperación y se pensaba que iba a durar entre dos y tres años. Actualmente, éste satélite ya no se encuentra en funcionamiento ya que el 22 de noviembre de 2009 tuvo un fallo mecánico del motor de su antena.

El satélite QuikSCAT ascendió en el espacio hasta alcanzar una órbita elíptica con una altitud máxima de 800 kilómetros sobre la superficie terrestre, consiguiendo así dar la vuelta a la Tierra alrededor de 14 veces al día, con el siguiente objetivo (WINDS.JPL.NASA, 2015c):

- Adquirir toda la información sobre las condiciones meteorológicas con mediciones de alta resolución de los vientos cerca de la superficie terrestre.
- Determinar los mecanismos y la interacción que hay entre el aire y el mar en varias escalas espaciales y temporales.
- Estudiar los cambios anuales y semestrales de la vegetación de la selva tropical.
- Estudiar los movimientos de hielo marino del Ártico.

- Combinar datos de viento con las mediciones de otra disciplina para ayudar sobre el cambio climático global.
- Mejorar la advertencia de tormentas y su correspondiente seguimiento.
- Mejorar pronósticos del tiempo cerca de las costas mediante el uso de los datos viento, así como la predicción de las olas.

Figura 6: Satélite QuikSCAT (EOSPSO.NASA, 2015)

El satélite utiliza la banda de frecuencia de microondas para obtener los datos de las observaciones: las microondas ayudan a traspasar las nubes y las precipitaciones llegando a la superficie de los océanos y respondiendo rápidamente al movimiento del aire, dependiendo de la rugosidad de la superficie marina debido al tiempo, la intensidad del eco de la microonda, la longitud de onda variará y podremos obtener la velocidad del viento en ese instante (PODAAC.JPL.NASA, 2015c).

Este satélite tenía una antena con 1 metro de diámetro con un plato de giro que producía dos haces puntuales, provocando un barrido de forma circular como podemos observar en la siguiente figura 7. También contiene un radar de alta

frecuencia de microondas que nos permite ofrecer aproximadamente 400.000 medidas, obteniéndose así la información de un 90% de la superficie terrestre en un día.

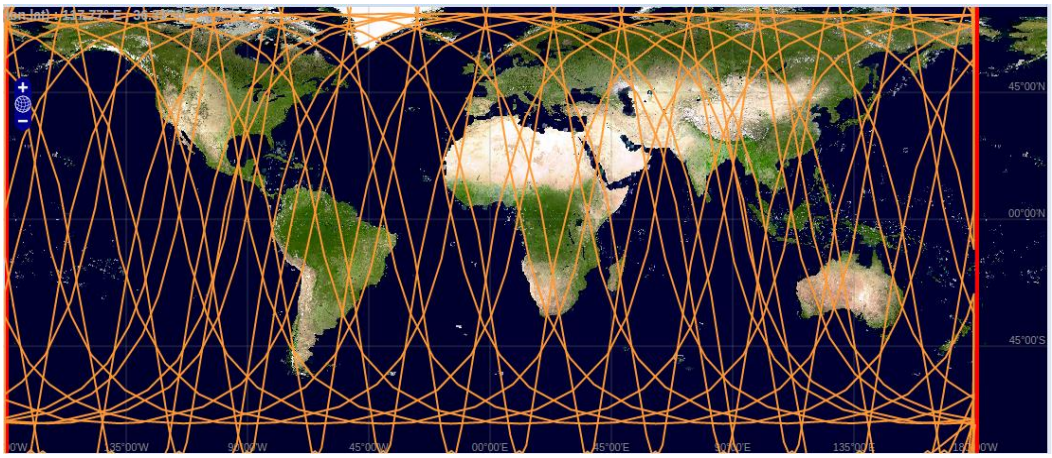


Figura 7: Movimiento del satélite QuikSCAT (PODAAC-TOOLS.JPL.NASA, 2015)

QuikSCAT nos ofrece dos tipos de resoluciones: una resolución gruesa en donde se recoge una observación cada 25 kilómetros cuadrados, almacenándose los datos en ficheros con formato HDF4 (HDFGroup, 2015); y una resolución más fina con una observación recogida cada 12,5 kilómetros cuadrados, guardándose los datos en ficheros con formato netCDF4.

1.4. OceanWinds con RapidSCAT

Debido a la utilidad del satélite QuikSCAT, que acabó por dejar de funcionar en 2009, la NASA relanza el proyecto de recogida de datos sobre vientos marinos y desarrolla RapidSCAT con el fin de reemplazarlo. RapidSCAT ya no es un satélite, sino un instrumento (toda la tecnología que residía en QuikSCAT para la recogida de datos con una serie de mejoras) que se encuentra actualmente a bordo de la Estación Espacial Internacional (ISS). Este instrumento fue lanzado el 20 de septiembre de 2014, desde Cabo Cañaveral, Florida, y actualmente sigue utilizándose.

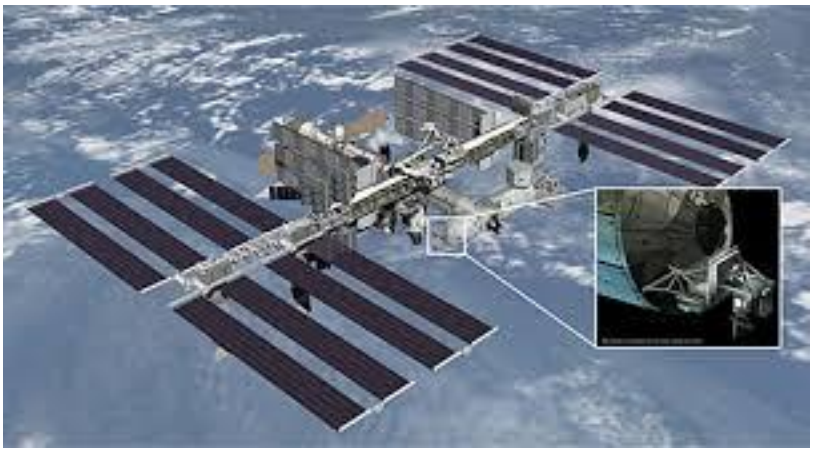


Figura 8: Satélite RapidSCAT (WINDS.JPL.NASA, 2015b)

Con esta misión, que reutilizó de manera ágil el hardware diseñado para QuikSCAT, añadiéndole una serie de mejoras, se persiguieron nuevos objetivos adicionales a los propuestos en QuikSCAT: aumentar la validez de los datos, realizar estudios sobre el ciclo diurno y semi-diurno de los vientos y la evapotranspiración sobre la tierra, entre otros. Hay que destacar que el movimiento de RapidSCAT es diferente al movimiento circular de QuikSCAT, ahora son ciclos diurnos o semi-diurnos con el objetivo de que, en aproximadamente dos meses, podremos tener datos de todos los puntos marinos del planeta. (WINDS.JPL.NASA, 2015d).

La obtención de estos datos se encuentra en el portal Web PODAAC (PODAAC.JPL.NASA, 2015a), estos datos tienen una resolución de 12,5 kilómetros cuadrados, y se encuentran en ficheros de formato netCDF4. En este portal Web existe un apartado con diferente software para poder tratar este tipo de datos.

1.5. Árbol R

El árbol R (o R-tree) es una estructura de datos desarrollada a partir del árbol B (Wikipedia, 2015a), propuesta por Antonin Guttman en 1984. La estructura se utiliza para almacenar datos espaciales; es decir, datos de 2 a 4 dimensiones, como es el caso que nos atañe donde utilizamos las coordenadas de una zona geográfica (longitud y latitud, dos dimensiones), o el caso de los datos geométricos (Wikipedia, 2015b).

La idea fundamental de esta estructura es agrupar los objetos más cercanos y los representa con un rectángulo mínimo de delimitaciones, que está definido por dos puntos: el inferior, que es el menor de todos los componentes que forman dicho rectángulo, y el superior, que es el mayor de todos los componentes; de tal forma que, a la hora de realizar una consulta, si los datos consultados no cortan dicho rectángulo, indica que no se encuentran en dicha zona. Este rectángulo recibe el nombre de MBB (*Minimum Bounding Box*, o Caja Mínima Contenedora). Los rectángulos mínimos también pueden solaparse e intersecarse con otros (Asto, P.C. y Apaza, Y. G., 2015).

El árbol R está formado por dos elementos. En primer lugar tenemos los nodos hojas, que contienen un identificador y su correspondiente MBB; y también tenemos los nodos no hojas, que mediante un puntero indican a la dirección de los nodos hojas, éstos últimos también contienen la MBB global de los nodos hojas a los que apunta. Por último, añadir que existe un nodo raíz similar a los nodos hojas pero que cumple con unas características diferentes.

El árbol R organiza los datos en páginas: cada nodo es una página. Es un árbol de búsqueda equilibrada, de tal forma que todos los nodos hojas se encuentran a la misma altura del nodo raíz. Cada página contiene un número máximo de datos, siendo M este número máximo, aunque también garantiza una cantidad de datos mínimos, a excepción del nodo raíz que no tiene un mínimo de descendentes (Árbol R, 2015). Por lo que podríamos decir que:

$$m_{min} = \left\lfloor \frac{M+1}{2} \right\rfloor$$

1.5.1. Inserción

El tipo de inserción que utilizaremos en nuestro árbol R es el de carga máxima, consistente en insertar datos en el árbol R desde cero intentando mantener las páginas de datos lo más llenas posible para una mayor utilización de la estructura, menor número de nodos existente y buscando, por tanto, una mejor eficiencia en la búsqueda. Para explicar el método, utilizaremos un ejemplo con nodos de capacidad

máxima 5 coordenadas. Las inserciones se realizan en el plano bidimensional, ya que es la dimensionalidad que este proyecto necesita.

Las 5 primeras inserciones de datos se realizan sin problemas debido a que es la capacidad de la página de datos. El árbol quedaría como se muestra en la figura 9.

Figura 9: Inserción de los cinco primeros datos en el Árbol R

Al insertar más datos se provoca la división de la página de datos llena en dos: la página de datos anteriormente llena y una nueva página de datos generada a tal fin. Los datos se trasladan a partes iguales en la medida de lo posible entre ambas páginas, intentando buscar la mayor cercanía espacial entre los datos contenidos en cada página. Se calcula la MBB de los datos ubicados en cada página y se trasladan hacía una nueva página de índice padre que contendrá el espacio de búsqueda contenido en cada página de datos. Este proceso se lleva a cabo siempre y cuando el nodo que sufra una inserción se encuentre lleno, impidiendo albergar el nuevo dato por desbordamiento. La figura 10 muestra cómo quedaría el árbol R de la figura 9 tras insertar un nuevo elemento F. Asimismo, la figura 10 presenta por encima del árbol R el espacio bidimensional de dónde se ubican cada uno de los elementos insertados en la estructura de indexación.

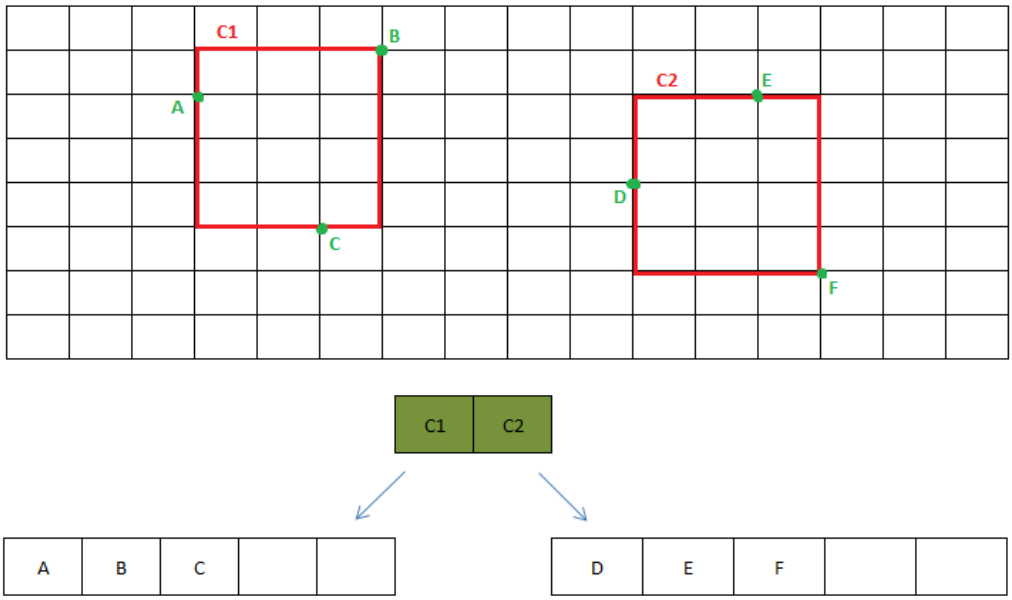


Figura 10: Insertar seis datos en el árbol R

Si ahora nuevamente queremos realizar una nueva inserción de datos, tendríamos dos nodos hojas con espacio, la inserción se realiza en el nodo con menor número de datos y en el orden de izquierda a derecha. La figura 11 representa este hecho.

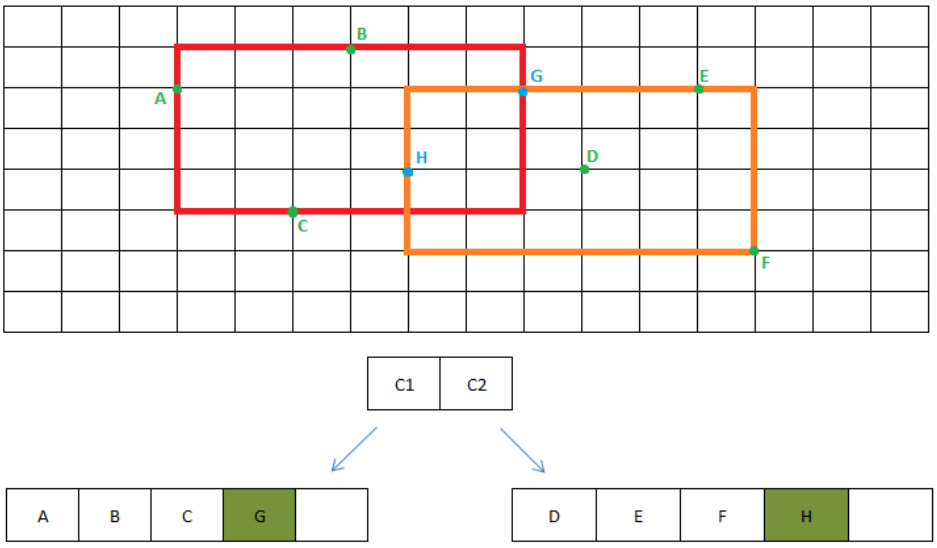


Figura 11: Insertar ocho datos en el árbol R

1.5.2. Búsqueda

A la hora de realizar la búsqueda en un árbol R, vamos a buscar los datos que forman un rectángulo, en este caso sería por el rectángulo "Q", a partir del árbol R que tenemos en la imagen siguiente.

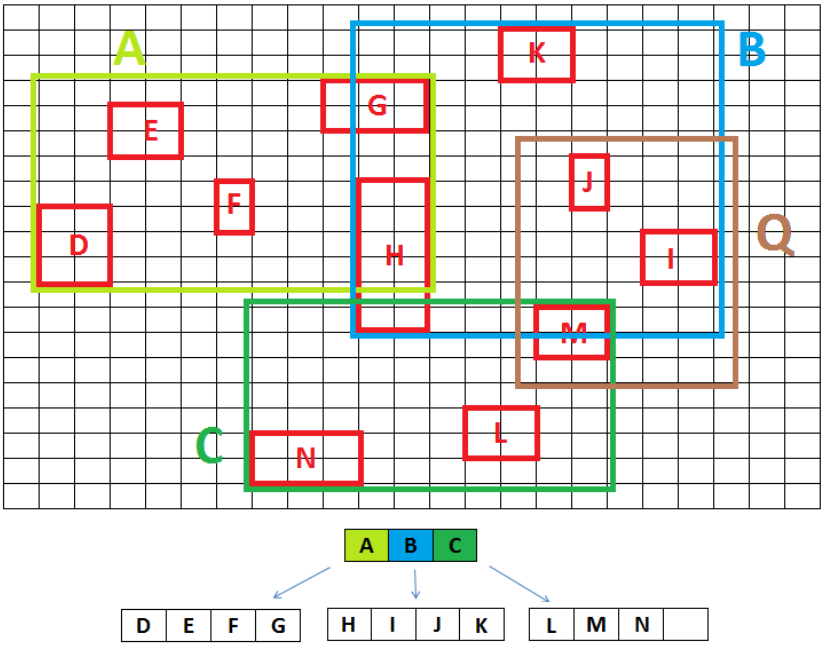


Figura 12: Realizar búsqueda en el árbol R

El proceso de búsqueda consiste en ir recorriendo el árbol R desde la raíz hasta los nodos hoja. Durante este proceso se irá comprobando si la MBB (Caja Mínima Contenedora) intersecta con los datos de consulta.

El primer paso es acceder al nodo raíz y empezar a recorrer cada uno de los campos y comprobar si en la primera región se encuentran los datos de búsqueda. Como podemos observar la región "A" no contiene los datos por lo que no descendemos por esta zona del árbol.

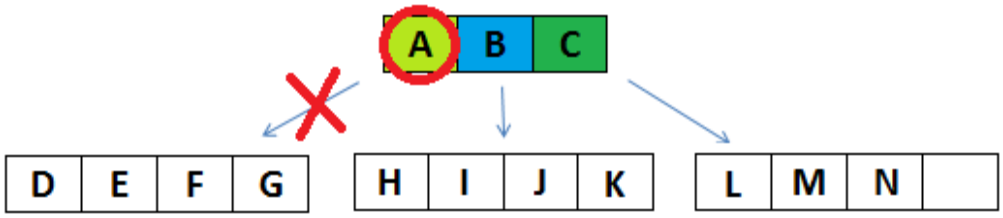


Figura 13: Búsqueda en la primera región

A continuación, comprobamos la región "B", en este caso sí contiene los datos de búsqueda, por lo que descendemos a las zonas que lo forman. Como podemos observar las zonas que contienen los datos de consulta serian la "I" y la "J".

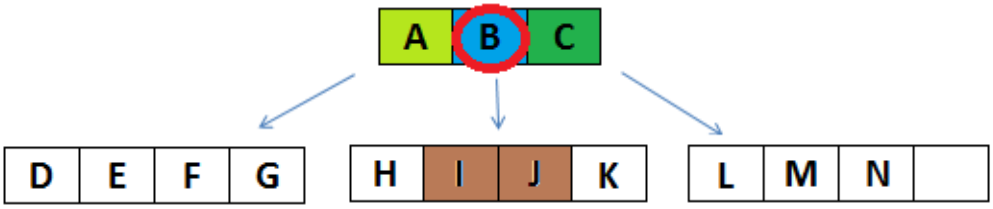


Figura 14: Búsqueda en la segunda región

Por último, accedemos a la última región, en este caso la "C" y volvemos a comprobar como en los casos anteriores. Nuevamente tiene datos de la consulta por lo que descendemos en el árbol y comprobamos que la zona "M" forma parte de la consulta.

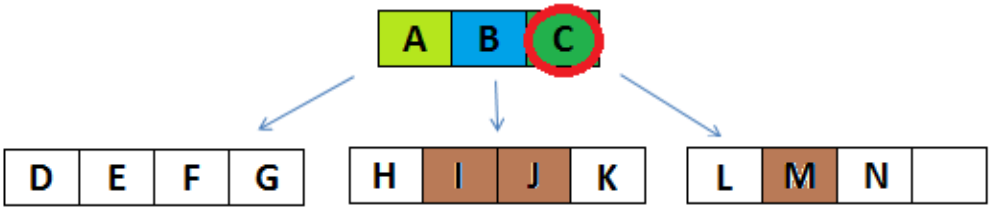


Figura 15: Búsqueda en la tercera región

El árbol R funciona eficazmente si no existen (muchos) solapamientos entre las regiones MBB, porque cuanto mayor solapamiento exista, mayor número de ramas hay que descender en cada consulta que se realice. Una manera de evitarlos y de aumentar, por tanto la eficacia de la estructura, es procurar que los datos que se

alojen en una hoja sean lo más próximos posible, construyendo MBB mínimas y, por tanto, con menor opción de solaparse con otras regiones o MBB. La mejor opción es cargar las páginas hojas con los elementos próximos es el espacio, siguiendo un orden de cercanía. La cercanía, de manera tradicional, se ha basado en el concepto de distancia: un punto es más cercano a otro cuanta menos distancia exista entre ellos.

El propósito de utilizar un árbol R en este proyecto es el de transformar una clave bidimensional (determinada por su latitud y longitud) en una clave unidimensional, que es la utilizada por Apache Cassandra para acceder a los elementos almacenados (acceso clave-valor). La idea es establecer un orden entre los objetos almacenados en la base de datos y asignarle un identificador en virtud de este orden. El problema radica en que el orden en dos o más dimensiones no existe como tal.

Como los objetos son bidimensionales, el árbol R los organizará espacialmente basándose en el concepto de cercanía, que utiliza la distancia para establecer la mayor o menor proximidad entre dos objetos o puntos del espacio. El identificador de acceso a la base de datos Apache Cassandra se asigna en orden según la cercanía entre sus objetos. Una de las formas de establecer un orden basado en la cercanía es utilizando las Curvas de Hilbert, que se describen a continuación en el apartado 1.5.3.

1.5.3. Curvas de Hilbert

Si queremos recorrer una distancia desde x_1 a x_2, podemos saber perfectamente la distancia que hay de un punto a otro, el problema viene cuando queremos establecer un orden entre dos puntos bidimensionales. . Durante toda la historia ha habido diferentes propuestas a la hora de abordar la distancia.

En 1891, David Hilbert diseñó una variación sobre la curva continua y recibió el nombre de Curva de Hilbert (Matesmates, 2015). La Curva de Hilbert tiene la propiedad de ser una curva continua que pasa por todos los puntos de un espacio cuadrado, de tal forma que el inicio de la curva se encontraría en la esquina inferior izquierda y el final en la parte superior izquierda. La construcción de esta Curva de Hilbert se lleva a cabo mediante diferentes pasos. En primer lugar se hace una

división de un espacio cuadrado en 4, de manera que dos números consecutivos se asocien en base a su proximidad y continuidad de la curva, como podemos ver en la figura 16.

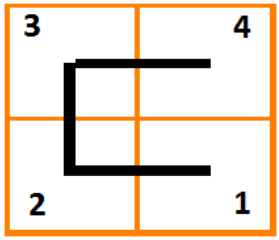


Figura 16: Curva de Hilbert paso primer (AbreojosEnsamblador, 2015)

Si volvemos a realizar otra subdivisión de los diferentes cuadrados y realizamos el mismo proceso tendríamos algo parecido a lo mostrado en la figura 17.

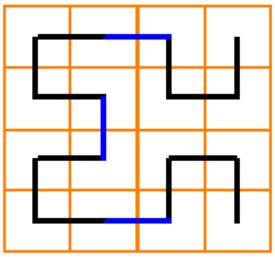


Figura 17: Curva de Hilbert paso segundo (AbreojosEnsamblador, 2015)

Este proceso se va realizando un número de veces hasta que las diferentes subdivisiones tienden a cero. (ECURED, 2015)

En este proceso se realiza la Curva de Hilbert en el árbol R para poder diferenciar varias regiones para que el acceso a la búsqueda en el árbol sea la más rápida posible.

1.6. Alcance y Objetivos del proyecto

Este proyecto tiene como objetivo principal ser capaz de almacenar información relacionada con datos geoespaciales sobre una base de datos NoSQL; en concreto, sobre Apache Cassandra, y poder realizar consultas sobre dichos parámetros.

Además, se ha realizado un estudio para que las consultas sean lo más rápidas posibles; para ello, se han diseñado diferentes estructuras en la base de datos para comparar unas con otras y poder elegir la de mejor comportamiento.

También se ha realizado el estudio e implantación del árbol R para mejorar aún más la eficiencia de las consultas de los datos geoespaciales almacenados.

Para llevar a cabo este proyecto, se han realizado diferentes módulos de aplicaciones. Los módulos son los siguientes:

- Una aplicación, realizada con el lenguaje de programación Python, que nos transforma los datos publicados por la NASA en archivos con extensión CSV, un formato de datos fácil de trasladar a cualquier aplicación de tratamiento, análisis o visualización de datos.
- Una aplicación Java que permite generar diferentes estructuras para la inserción de los datos en la base de datos NoSQL Apache Cassandra.
- Una aplicación Python para realizar las consultas sobre la base de datos Apache Cassandra.
- Un módulo de integración con el árbol R para agilizar las consultas.
- Por último, un módulo de consultas y visualización Web de los datos meteorológicos almacenados.

Todas las implementaciones de los distintos módulos de aplicaciones se han desarrollado y unificado bajo el sistema operativo Linux con su distribución Ubuntu

Capítulo 2: Materiales: entorno, herramientas y lenguajes de programación

En este capítulo vamos a hablar sobre el software y hardware utilizado durante el progreso del proyecto para llevarlo a cabo, así como los entornos utilizados, lenguajes de programación y sistema operativo y el hardware utilizado.

2.1. Software

A continuación, vamos a ir describiendo todos los elementos de tipo software utilizados en este proyecto.

2.1.1. Apache Cassandra

Esta herramienta ha sido el gestor de base de datos que hemos utilizado durante el proyecto para almacenar todos los datos procedentes de los satélites QuikSCAT y RapidSCAT.

La versión utilizada de este gestor base de datos ha sido 2.2.0, esta versión es la última que nos proporciona Apache Cassandra. Para este proyecto se han utilizado dos bases de datos distintas, la primera para hacer un estudio de las diferentes estructuras que podíamos llevar a cabo, y una base de datos con un mes de cada satélite.

Para llevar a cabo la instalación de esta herramienta, se puede realizar de dos formas distintas en el caso del sistema operativo Ubuntu, que ha sido el utilizado. Se puede utilizar la consola de dicho sistema operativo o se puede descargar una versión portable. Toda esta información viene reflejada en la web de Apache Cassandra (Apache Cassandra, 2015).

2.1.2. Ubuntu

Se trata del sistema operativo utilizado, es de software libre y está basado en Linux, la versión utilizada ha sido 14.04 LTS.

Todas las herramientas utilizadas durante el proyecto se han instalado en este sistema operativo. Se puede descargar dicho sistema a partir de la siguiente página web de forma gratuita (Ubuntu.org, 2014).

2.1.3. Java

Con el lenguaje de programación Java se han realizado programas como la consulta de los datos, la creación de las diferentes estructuras para la base de datos, y todo el proceso de la página web realizada para la consulta de éstos.

Para las consultas a las bases de datos ha sido necesario añadir los driver de Cassandra mediante un archivo JAR.

La versión utilizada ha sido 1.8 que la podemos encontrar en su página web (ORACLE, 2014).

2.1.4. Python 2.7

Con este lenguaje de programación se ha realizado la fase de transformación de los datos, es decir, se ha obtenido ficheros CSV a partir de ficheros netCDF4 proporcionados por el FTP de la NASA. También se ha utilizado para llevar a cabo las gráficas realizadas para el estudio de las estructuras de la base de datos y para ello ha sido necesario los driver de Cassandra para Python (Academy Datastax, 2015b).

Para realizar estos programas se han utilizado una serie de librerías. Para la instalación de este lenguaje de programación, no ha sido necesario realizar ningún tipo de instalación, ya que, el sistema operativo lo tiene instalado. En caso de utilizar otro sistema operativo es necesario acceder a la página web del lenguaje para su instalación (Python.org, 2015).

2.1.5. Eclipse

Eclipse es un entorno de desarrollo integrado (IDE) que contiene un conjunto de herramientas de programación de código abierto multiplataforma.

Este entorno ha sido fundamental para el desarrollo del proyecto, ya que se han realizado todos los programas en él, tanto la transformación de los datos, como la carga y la consulta de estos.

Como se han utilizado dos lenguajes de programación, es necesario instalar un IDE para Python que recibe el nombre de Pydev que lo podemos instalar a partir de su página web (Pydev.org, 2015). Y para java hemos utilizado Java Development Toolkit (JDT).

La versión utilizada de eclipse ha sido Luna y se puede encontrar en la página oficial de Eclipse de forma gratuita (Eclipse.org, 2014).

2.1.6. DataStax DevCenter

DevCenter se trata de una herramienta que nos permite crear y ejecutar consultas CQL contra la base de datos Apache Cassandra. Para ello es necesario realizar una conexión previa a la base de datos. También nos ofrece la posibilidad de navegar por las diferentes tablas de la base de datos.

La versión utilizada en este proyecto ha sido 1.2.2, ésta versión se puede obtener desde su página web (Datastax, 2014).

2.1.7. Librerías necesarias para la transformación de datos netCDF4

Para el tratamiento de los ficheros con extensión ‘nc’ proporcionado por la NASA para ofrecer los datos de los dos satélite, es necesario la instalación de la librería netCDF, para ello es necesario descárgala en la siguiente página (UNIDATA.UCAR, 2015).

Como estos ficheros contienen datos científicos, es necesaria la instalación de un paquete de Python, llamado Numpy, que nos permite la lectura de estos datos. La descarga de este paquete la encontramos en su página web (Numpy, 2015).

2.1.8. Librerías necesarias para crear las gráficas en Python

Para poder realizar las gráficas en Python, utilizadas para el estudio de las estructuras de la base de datos, es necesario instalar la librería Matplotlib y encontraremos toda la información necesaria en su página web (Matplotlib, 2015).

2.1.9. FileZilla

FileZilla se trata de un software de código abierto y de distribución gratuita que nos permite transferir archivos.

Esta herramienta nos permite descargarnos los datos procedentes del FTP de la NASA. Estos datos vienen divididos en años y estos a su vez en días, por lo que la utilización de esta herramienta ha sido muy útil. La versión utilizada de esta herramienta ha sido 3.7.3, dicha herramienta ha sido instalada en Ubuntu. Esta herramienta se puede descargar desde su página web (Filezilla.org, 2014).

2.1.10. Apache Tomcat

Apache Tomcat es una aplicación de software de código libre de Java Servlet, JavaServer Pages, entre otras. Se trata de una aplicación libre y multiplataforma.

En este proyecto se utiliza como servidor de la página web realizada para la visualización de las consultas realizadas en Apache Cassandra. La versión utilizada en el proyecto es Tomcat 7.0 y se puede descargar desde su página web (Tomcat.Apache.org, 2015).

2.1.11. Google Maps y Google Earth

Google Maps y google Earth son API's web para la visualización de mapas en páginas web. Estas dos API's han sido utilizadas para mostrar en un mapa la visualización de las consultas, para que sea más fácil de ver por parte del usuario.

Para la inserción de estas dos API's se ha utilizado un lenguaje de programación llamado JavaScript (W3SCHOOLS, 2015). La página de referencia utilizada para Google Maps ha sido su página oficial (Developers.Google, 2015b), de igual manera para Google Earth (Developers.Google, 2015a).

2.2. Hardware

Durante todo el desarrollo del proyecto, se ha utilizado un ordenador portátil personal, dicho portátil tiene un procesador Intel Core i5-430M, con 4GB de memoria RAM y con un disco duro de 640GB.

Hay que añadir que el equipo tiene diferentes particiones y el sistema operativo Ubuntu, que ha sido el sistema operativo en el cual se ha desarrollado el proyecto, tiene una capacidad total de 140GB, al tratarse de datos de gran capacidad ha sido utilizado un disco duro HP de 2TB.

Capítulo 3: Desarrollo del proyecto

3.1. Viabilidad del proyecto

Para llevar a cabo este proyecto, se puede dividir en las diferentes partes que lo forman, por lo que el estudio de viabilidad del mismo ha sido muy diverso. Este proyecto consiste en un proceso de extracción, transformación y carga de los datos en una base de datos NoSQL, en concreto en Apache Cassandra, y su posterior consulta.

Durante el proceso de extracción, se ha utilizado un programa llamado FileZilla, que realiza la descarga de los datos geoespaciales utilizados en este proyecto. Estos datos provienen de la NASA de dos fuentes diferentes, en primer lugar por el satélite QuikSCAT y en segundo lugar, por el instrumento RapidSCAT. Estos datos son obtenidos mediante la transferencia de archivos FTP, por lo que el uso de FileZilla ha sido muy importante. La instalación de este programa no ha sido un gran impedimento, ya que se encuentra para diferentes sistemas operativos y su instalación es muy sencilla para todos ellos.

Para el proceso de transformación, se ha realizado en primer lugar, un proyecto con el lenguaje de programación Python, para la transformación de los ficheros descargados con extensión NetCDF4 a ficheros con extensión CSV. Este proceso puede ser complicado, debido a que es necesaria la instalación de las librerías adecuadas para tratar archivos con extensión "*.nc*" y comprender la estructura que usan estos archivos.

Hay que añadir, que en el FTP en el cual se encuentran los datos, existen una serie de archivos con diferentes lenguajes de programación para el tratamiento de los mismos. Esto hace mucho más sencillo llevar a cabo este proceso, ya que se pueden obtener fácilmente.

A continuación, se ha realizado otra aplicación para la transformación de los datos. En dicha aplicación tenemos ahora como archivo origen los ficheros CSV

obtenidos del proceso anterior. El objetivo de este programa es estructurar los datos de diferentes formas para la inserción de ellos en la base de datos Apache Cassandra.

Dentro de este programa, se puede hacer la carga de estos datos en nuestra base de datos. Para realizar esta carga, es necesario utilizar la API de Cassandra que se proporciona en su página web. Su acceso es muy sencillo y en la misma se encuentran también, diferentes tutoriales para su correspondiente uso. Esta API existe para diferentes lenguajes de programación. Esta aplicación se ha desarrollado con el lenguaje de programación Java.

La instalación de Apache Cassandra es muy sencilla, ya que existe mucha información en la red, además, de en su página oficial. Dicha instalación puede realizarse en diferentes sistemas operativos a partir de la descarga de los ficheros correspondientes de dicha página.

La consulta de los datos, se lleva a cabo a partir de otro programa con lenguaje Python, el cual necesita la conexión a la base de datos y realizar las consultas sobre ella, por lo que también, es necesario el uso de la API, en este caso para Python.

Para mejorar el tiempo de ejecución de las consultas sobre Apache Cassandra, se ha utilizado un árbol R con el lenguaje de programación Java, gracias a la librería "*Java Spatial Index*". Este proyecto realiza el proceso de creación del árbol y la posterior consulta de los datos a partir de él.

Hay que añadir, que para ejecutar este proyecto es necesaria la utilización de la versión 1.8 de Java. Esto será un problema a la hora de realizar las consultas para la aplicación web, ya que el servidor que lanza la aplicación, solo ejecuta con la versión 1.7 provocando que no se pueda insertar el árbol R en la aplicación web.

Por último, para entender mejor las consultas y situarlas en el espacio y en el tiempo, se ha desarrollado una aplicación web desarrollada con Servlet de Java y las API's de Google Maps y Google Earth, aunque esta última ya no se puede utilizar debido a que ha sido desaprobado a partir del día 12 de diciembre de 2014 y se dejara de trabajar el 12 de diciembre de 2015. Llevar a cabo todo este proceso, ha sido fácil

ya que se encuentra mucha información en las páginas oficiales de Google Maps y Google Earth.

Hay que añadir, que durante la realización de este proyecto, se han encontrado diferentes limitaciones, por lo que se han tenido que usar una pequeña parte de los datos proporcionados por la NASA. También hay que añadir que al utilizar datos con una resolución de 12 kilómetros y medio, la cantidad de datos es enorme, y el proceso de transformación y de carga de los datos es muy lento. Y por último, al usar únicamente un equipo no se ha podido demostrar correctamente la escalabilidad horizontal que nos proporciona la base de datos Apache Cassandra.

3.1.1. Tareas realizadas durante el proyecto

Durante este proyecto se han llevado a cabo diferentes tareas como podemos ver en la siguiente tabla en función del tiempo.

Tareas Realizadas	Tiempo en días
Búsqueda de información	30 días
Descarga de los datos	20 días
Instalación de Apache Cassandra	10 días
Programas de Transformación de los datos	120 días
Aprendizaje de lenguaje CQL	31 días
Programa para generar diferentes estructuras para Apache Cassandra	90 días
Desarrollo aplicación web	90 días
Árbol R	30 días
Documentación	45 días

A continuación, se muestran dichas tareas sobre un diagrama de Gantt con un mayor desglosamiento.

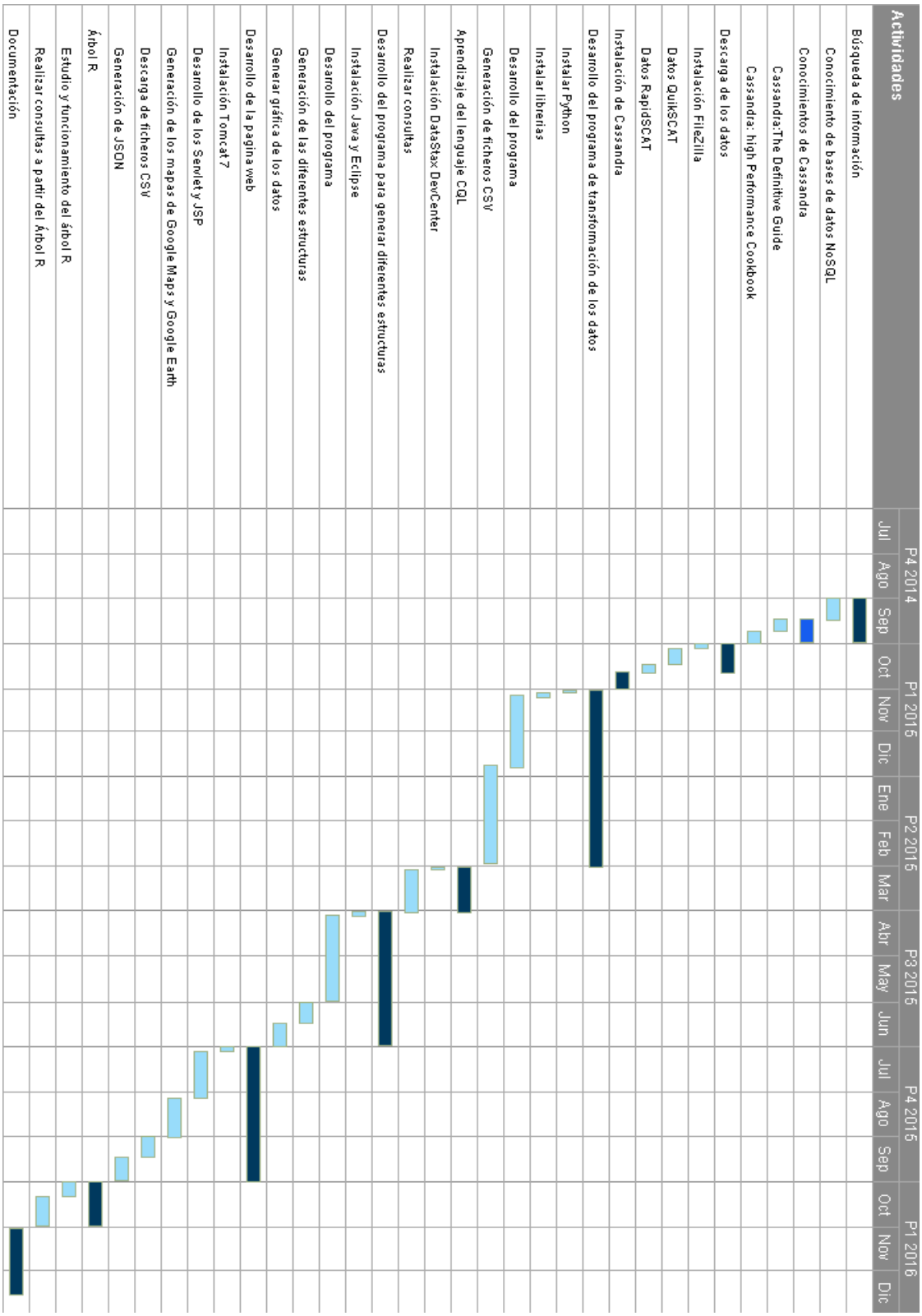


Figura 18: Diagrama de Gantt

3.2. Análisis general del sistema desarrollado

Para llevar a cabo este proyecto, se han realizado un conjunto de aplicaciones para los procesos de obtención, transformación, carga y consulta de los datos, además de la utilización del gestor de base de datos Apache Cassandra. También se ha añadido la utilización de un índice externo mediante un árbol R.

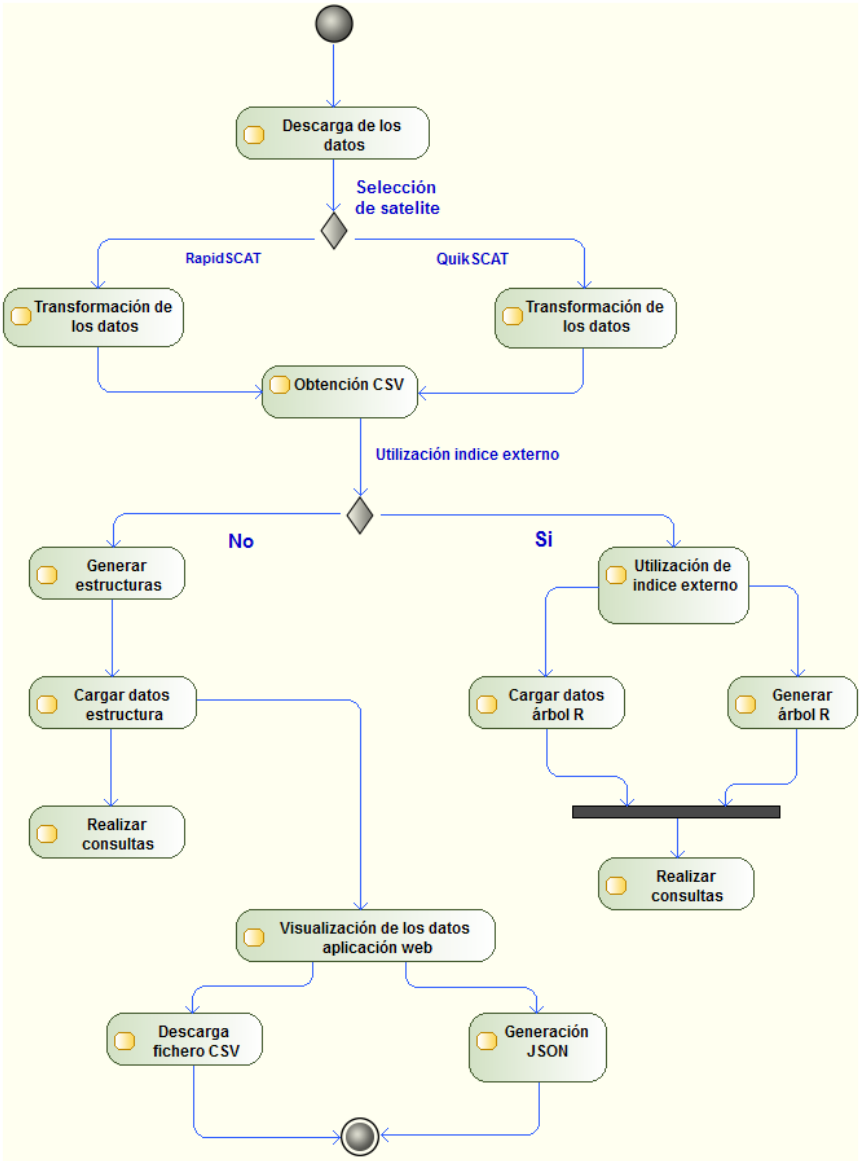


Figura 19: Diagrama de actividad global del proyecto

Para entender mejor todo el proceso que se lleva a cabo para la realización de este proyecto, a continuación, se puede ver con el diagrama de actividad mostrado en la figura 19.

En los siguientes apartados se irá realizado el análisis de las diferentes aplicaciones que se han creado de forma individual.

3.2.1. Programa de transformación

Una vez realizada la descarga de todos los datos, entra en juego el proceso de transformación, este proceso consiste en generar ficheros CSV a partir de los datos descargados.

Esta aplicación está desarrollada con el lenguaje de aplicación Python con el nombre de "*TFG_GenerarCSV*". El uso de esta aplicación es mediante una interfaz que se encuentra en el script principal que recibe el nombre de "*crearCSV*". Pudiendo generar diferentes CSV dependiendo del satélite seleccionado, existiendo dos métodos diferentes para generarlo.

En primer lugar, si utilizamos el satélite QuikSCAT, se ejecutará el modulo "*Quikscat*", este módulo nos permitirá ir recorriendo el directorio en el cual se encuentran todos los días que queremos añadir al fichero CSV, ya que los datos descargados están separados en días y cada día tiene una media de 14 ficheros con extensión "*.nc*". De igual forma, nos permitirá descomprimir los ficheros con extensión "*.gz*". Dentro de este método se encuentra el método "*writeCSVQuikscat*", que nos permitirá ir recuperando todos los campos de cada uno de los ficheros descargados.

En segundo lugar, en el caso de utilizar el instrumento RapidSCAT, el proceso es el mismo pero esta vez recuperará más campos que el satélite QuikSCAT, para ello utilizaremos el método "*Rapidscat*" y dentro de éste "*writeCSVRapidscat*". Hay que añadir que todos los ficheros CSV generados tienen su correspondiente cabecera, dependiendo de sí es QuikSCAT o RapidSCAT.

Por último, hay que añadir que para realizar estos procesos, se utiliza un scrip llamado "*readnc*", este script fue descargado desde la página de la NASA, y realiza la función de lectura del fichero, siendo fundamental para llevar a cabo este proceso.

Para entender mejor el proceso que se realiza en esta aplicación, a continuación mostramos el diagrama de actividad con la figura 20.

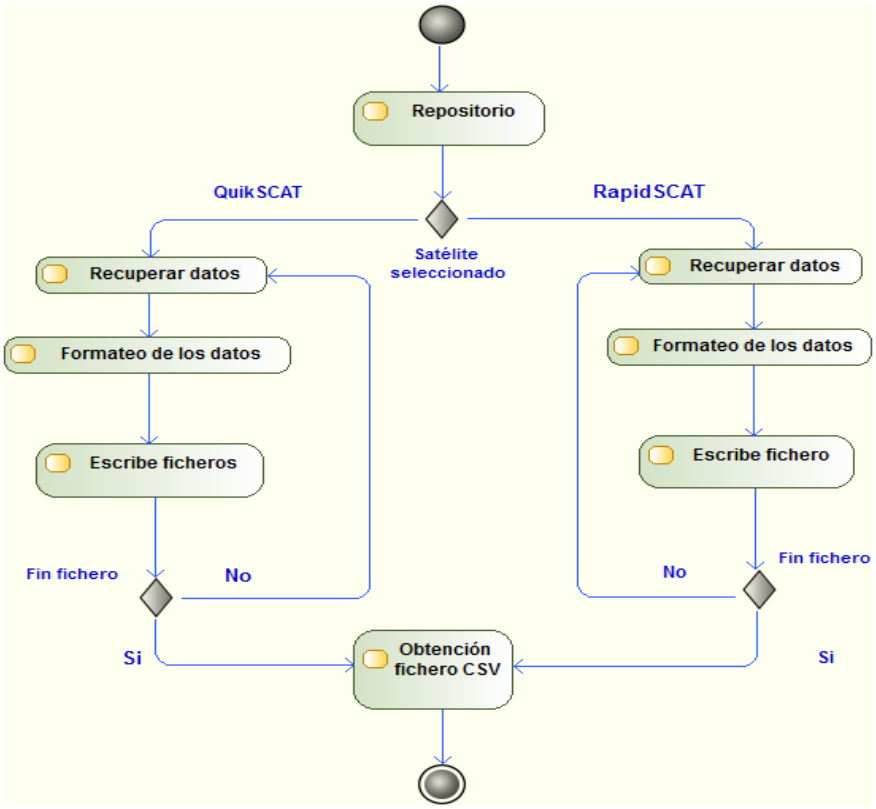


Figura 20: Diagrama de actividad programa de transformación

3.2.2. Programa de generación de estructura y carga

Una vez generado el CSV en el proceso anterior, nos disponemos a generar las diferentes estructuras para la base de datos. El objetivo de estas estructuras es generar cubos de diferentes tamaño para comprobar cuál es la mejor a la hora de realizar búsquedas en la base de datos. De tal forma que dividimos la tierra en diferentes partes en función de su latitud y longitud, y a su vez en días, de tal forma que cada una de ella es un cubo.

Para llevar a cabo este proceso, se ha desarrollado una aplicación con el lenguaje de programación Java que recibe el nombre de "*TFG_GenerarEstructura*". Este proyecto contiene diferentes paquetes, en primer lugar, podemos hablar del "*ficherosCSVCubos*", en el cual se encuentra la clase java "*crearCubos*", en esta clase se realiza todo el proceso de transformación de los datos, ya que únicamente nos quedamos con 3 decimales del campo longitud y latitud, y dos decimales en el resto de los datos. Estos métodos reciben el nombre de "*formatoLon*", "*formatoLat*" y "*formatoDatos*", respectivamente. También esta clase contiene el método principal que recibe el nombre de "*leerFicheroCubos*", este método nos va a formar las diferentes estructuras y su posterior escritura de los datos en ficheros CSV.

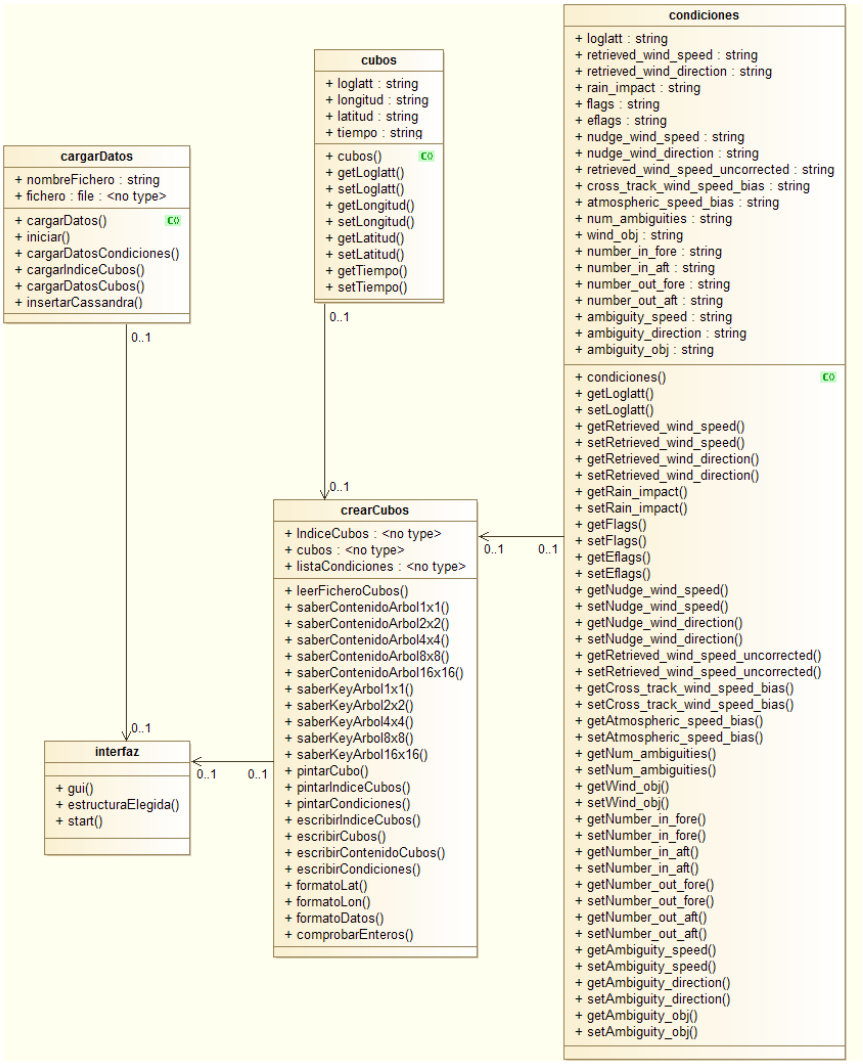


Figura 21: Diagrama de clases de la aplicación estructura y carga

Existe también un paquete con el nombre de "*estructuras*", en éste, se encuentran dos clases llamadas "condiciones" y "cubos". Estas son clases contenedoras de los diferentes datos, y son utilizadas en el proceso anterior.

También está el paquete con el nombre "*cargarDatos*", que contiene una clase que recibe el mismo nombre, que se encarga de generar un archivo con extensión "*cql*" para la carga de los datos en Apache Cassandra.

Y por último, comentar el paquete interfaz que de igual forma contiene una clase con el mismo nombre que se encarga de generar la interfaz de la aplicación.

3.2.3. Programa de consultas

Una vez que tenemos todos los datos cargados, nos disponemos a realizar la consulta a dichos datos, para ello, vamos a utilizar un programa Python, este programa recibe el nombre de "*TFG_GenerarCSV*" y accedemos al script "*Grafica_tfg*". Dentro de este script podemos encontrar diferentes métodos, entre los cuales destacan la conexión a la base de datos Apache Cassandra que recibe el nombre de "*iniciarConexionCassandra*", así como la desconexión a la misma con el nombre de "*cerrarConexionCassandra*". También podemos encontrar los diferentes métodos de consulta dependiendo de la estructura que se desee. Así como la generación de las diferentes graficas espaciales, temporales y espacio-temporales para su posterior estudio.

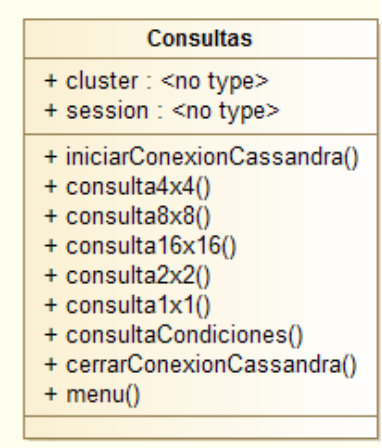


Figura 22: Diagrama de clase de la aplicación consulta

3.2.4. Programa del árbol R

El objetivo de esta aplicación, es generar un índice externo con un árbol R para mejorar el tiempo de las consultas realizadas sobre la base de datos Apache Cassandra. Para la utilización de este árbol, se ha utilizado la librería "*Java Spacial Index*".

Para utilizar dicho árbol, hay que llevar a cabo diferentes pasos. En primer lugar, es necesaria la obtención de los diferentes puntos de los ficheros CSV generados en la primera aplicación.

Por cada punto obtenido, generamos un valor entero relacionado con la curva de Hilbert para organizar los puntos obtenidos por proximidad, y por último generamos el árbol.

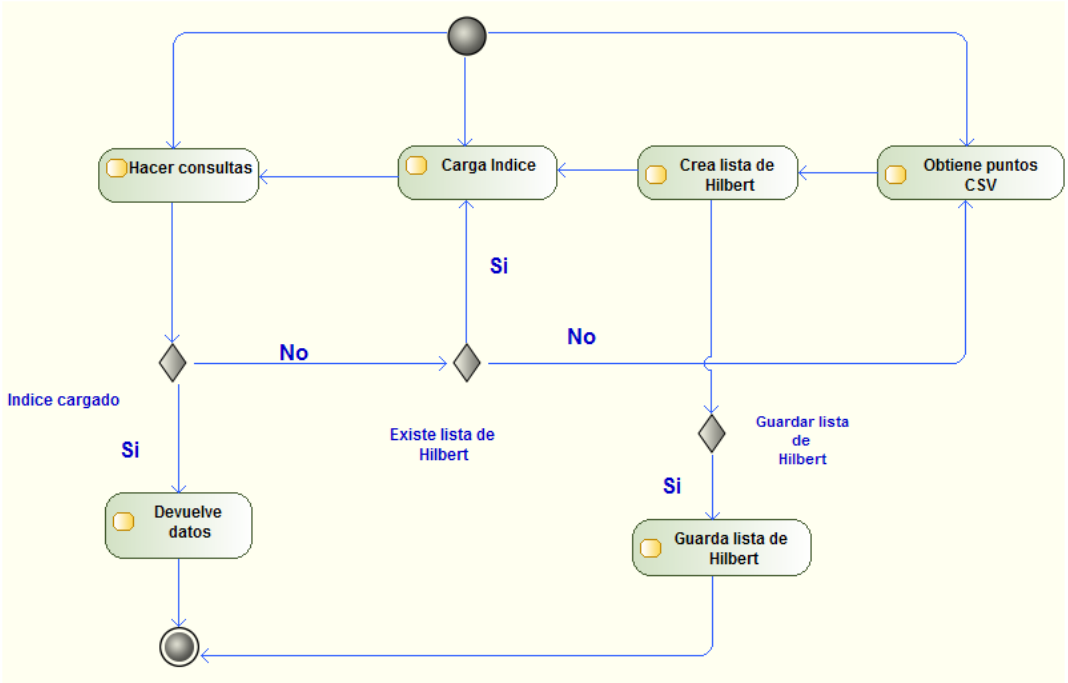


Figura 23: Diagrama de actividad de la aplicación árbol R

Una vez generado el árbol, se podrá realizar la consulta de los datos. Esta consulta se puede dividir a su vez en dos partes, en primer lugar consultamos sobre el árbol para que nos indique qué región tenemos que consultar en la base de datos, y a continuación recuperar los datos de la región seleccionada.

Todo este proyecto se ha realizado con el lenguaje de programación Java y recibe el nombre de "*TFG_Arbol*". Dicho proyecto tiene diferentes paquetes, entre los que hay destacar los tres más importantes, como son "*HilbertAndTree*" que contiene tres clases que se encargan de la creación del árbol con su correspondiente curva de Hilbert. Otro paquete que recibe el nombre de "*Cassandra*" que contiene una clase con el nombre de "*consultasCassandra*", que se encarga de realizar la conexión y la posterior consulta de los datos. Y por último, el paquete "*GenerarEstructuraArbol*" que nos genera la estructura utilizada en Apache Cassandra para la carga de los datos.

3.2.5. Programa de visualización de los datos con la aplicación web

Y la última aplicación realizada en este proyecto, consiste en la visualización de los datos y que se lleva a cabo mediante una aplicación web con el lenguaje de programación Java y JavaScript.

Para llevar a cabo este proyecto, se han utilizado tres Servlet. El Servlet con el nombre "*ServletInicio*" se encarga de llevar a cabo todo el proceso de consulta y salida de los datos. El Servlet "*generarJsonServlet*" se encarga de generar un JSON de la consulta realizada y de igual forma, el Servlet "*enviarFicheroCSVServlet*" descarga un fichero CSV con la consulta realizada.

El proceso de visualización de los datos en un mapa se lleva a cabo con Google Maps y Google Earth mediante JavaScript.

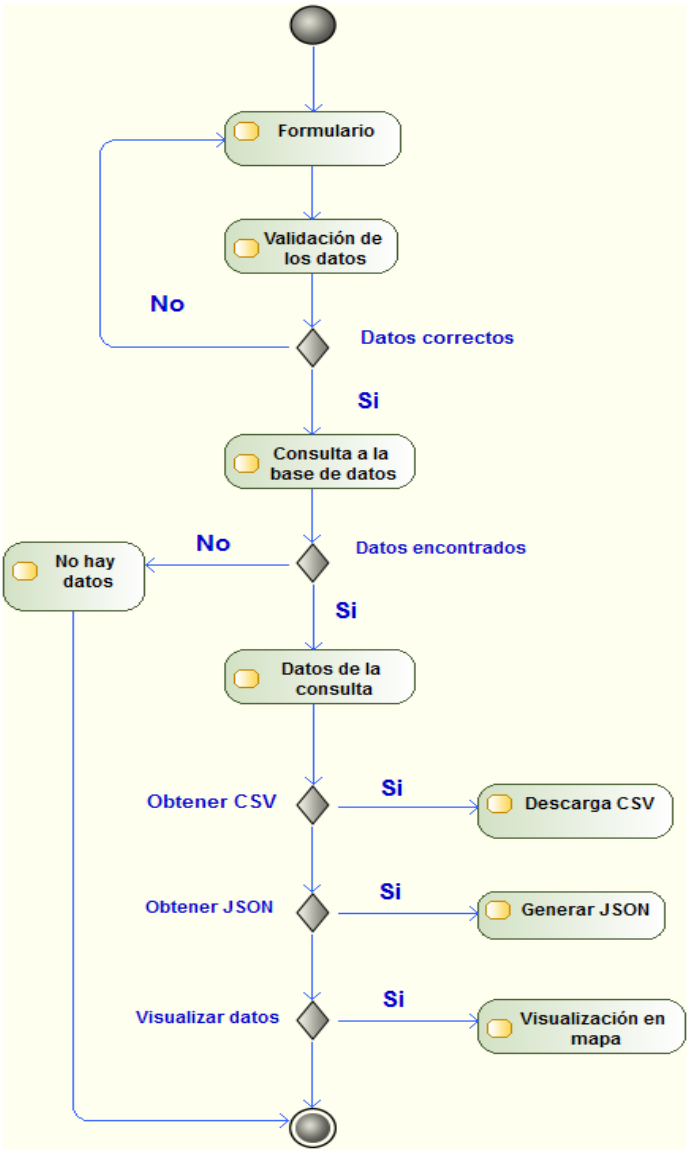


Figura 24: Diagrama de actividad de la aplicación visualización de los datos

3.3. Identificación de las fuentes y extracción de los datos

El primer paso para llevar a cabo este proyecto, es saber dónde se encuentran y cómo nos podemos descargar los datos. Como ya hemos comentado durante este documento, los datos son proporcionados por la NASA desde la página web PODAAC (https://podaac.jpl.nasa.gov/). Hay que decir que estos datos se encuentran a disposición de todo el mundo que quiera utilizarlos.

Estos datos se pueden descargar de diferentes forma, en este caso han sido descargados mediante FTP utilizando para ello el software libre FileZilla. Se ha utilizado este software porque nos permite la parada y su posterior reanudación mientras se realizan las descargas, así como su uso sencillo y la descarga correcta de los ficheros. La instalación de este software es muy sencilla, y está explicada en el apartado de "*Preparación del entorno*".

Como ya hemos comentado en varias ocasiones, para realizar este proyecto utilizamos dos satélites diferentes, por lo que tenemos que acceder a dos directorios distintos. En primer lugar si accedemos a la página indicada anteriormente, tenemos que acceder al apartado "*Data Access*" y a continuación pulsar sobre "*FTP*".

Una vez realizado este paso, accedemos a "*OceanWinds*", ahí podemos observar los dos satélite que vamos a utilizar. Una vez que accedemos a uno de los dos, pulsamos sobre su carpeta "*L2B12*" y seleccionamos la versión "v3" en el caso de QuikSCAT y "1.1" en el caso de RapidSCAT. Una vez realizado esto, nos aparecerán los años y a su vez los días con sus correspondientes datos, en el caso de QuikSCAT desde el año 1999 hasta el 2009, en nuestro caso nos hemos descargado los datos correspondientes al año 2005. Y en el caso de RapidSCAT nos aparecen los años 2014 y 2015, y en este caso nos hemos descargado los datos correspondientes entre los meses octubre y diciembre del 2014.

Hay que añadir que todos los datos descargados corresponden a datos con una densidad de 12.5 km^2, por ello se ha seleccionado la opción "*L2B12*".

3.4. Transformación

Una vez descargados los datos, tendremos una carpeta principal correspondiente al año que hemos seleccionado y un conjunto de subcarpetas correspondiente a los días del año y éstos a su vez un grupo de ficheros comprimidos con los datos.

El siguiente paso consiste en transformar todos esos ficheros en un único fichero CSV para su posterior carga en la base de datos. Para realizar la transformación,

tenemos que diferenciar un satélite de otro, ya que contienen muchos datos iguales, pero también hay datos diferentes.

Los dos satélites contienen los datos en ficheros netCDF-4, por lo que ha sido necesario instalar la librería correspondiente para Python, ya que este proyecto se realiza con este lenguaje.

Esta aplicación tiene una interfaz, por lo que se tiene que informar del satélite al que corresponden los datos que vamos a tratar e indicarnos si los datos se encuentran comprimidos, para poder realizar una operación u otra, así como la validación de los campos para que no estén vacíos. Este módulo recibe el nombre de "*empezar*".

```
Método empezar (satélite, file_path, fileCSV)
        Si file_path == "" hacer
                Salida= "Error datos vacios"
        Sino si fileCSV == "" hacer
                Salida = "Error datos vacios"
        Sino
                Si satélite == 1
                        Quikscat()
                Sino si satélite == 2
                        Rapidscat()
                Fin si
        Fin si
Fin método
```

Algoritmo 1: Método de selección de satélite

3.4.1. Datos QuikSCAT

Una vez seleccionada la opción de QuikSCAT, llevamos a cabo un método que recorre toda la ruta indicada, en la cual se encuentran todos los días y a su vez todos datos descargados. Este método contiene la opción de descomprimir los ficheros si vienen comprimidos, ya que cuando se descargar están así, de igual forma, cada fichero CSV tiene su cabecera correspondiente a los datos a escribir, en este caso, QuikSCAT nos informa sobre un total de 14 campos.

Este método es una modificación del script que se encuentra en la página PODAAC, que mostraba por pantalla todos los datos de un único fichero, por lo que

la modificación ha consistido en recorrer todos los ficheros y guardarlos en un fichero CSV, en vez de mostrarlo por pantalla.

Para llevar a cabo todo esto, también se ha utilizado un script proporcionado de nuevo por la NASA que se encarga de guardar en diferentes variables los datos de cada uno de los ficheros, este script recibe el nombre de "*readnc*" y no ha sido modificado, únicamente se le ha añadido un método más, que explicaremos en el siguiente paso.

```
Método Quikscat(file_path, fileCSV, compresión)
        f = open(fileCSV, w)
        para carpeta en file_path hacer
                para fichero en carpeta hacer
                        si compresión == 2
                                descomprimir fichero
                        sino
                        cabecera, datos = readnc.readVars(fichero)
                        si primerIteracion hacer
                                f.write(cabecera)
                        sino
                        writeCSVQuikscat(fichero, datos,f)
                        cerrar fichero
                fin para
        fin para
        cerrarCSV
fin método
```

Algoritmo 2: Conversión de ficheros netCDF4 a CSV

Todos los archivos de tipo netCDF-4 contienen los datos en forma de matriz, en este caso se trata de matrices de 3248x152, a excepción del campo tiempo que contiene una matriz de 3248x1. Es por ello que una vez que tenemos los datos leídos de un fichero tenemos que ir recorriendo campo a campo de la matriz para su posterior inserción en el fichero CSV. Ahora es cuando actúa un nuevo método, llamado en el método anterior con el nombre de "*writeCSVQuikscat*" que realiza lo descrito en el Algoritmo 3.

```
Método writeCSVQuikscat(fichero, datos, f)
        Fecha = readnc.readDate(fichero)
        Para i en len(datos[0])
                Time = Datos[11][i]+Fecha
                For j en len(datos[0][0])
                        Lat = datos[0][i][j]
                        Lon = datos[1][i][j]
                        retr_wind_speed = datos[2][i][j]
                        retr_wind_direction = datos[3][i][j]
                        rain_impact = datos[4][i][j]
                        flags = datos[5][i][j]
                        eflags = datos[6][i][j]
                        nudge_wind_speed = datos[7][i][j]
                        nudge_wind_direction = datos[8][i][j]
                        retr_wind_speed_uncorrect= datos[9][i][j]
                        num_ambiguities = datos[10][i][j]
                        cross_track_wind_speed_bias = datos[12][i][j]
                        atmospheric_speed_bias = datos[13][i][j]

                        f.write(time, lat, lon, retr_wind_speed, retr_wind_direction,
                        rain_impact, flags, eflags, nudge_wind_speed,
                        nudge_wind_direction, retr_wind_speed_uncorrect,
                        num_ambiguities, cross_track_wind_speed_bias,
                        atmospheric_speed_bias)
                Fin para
        Fin para
Fin método
```

Algoritmo 3: Escritura de los datos en CSV

Como podemos observar, se llama a un método que se encuentra en el script "*readnc*", este método nos permite obtener el día y el año de los datos, esto unido con el tiempo nos será de gran importancia más adelante.

3.4.2. Datos RapidSCAT

El proceso llevado a cabo para los datos del instrumento RapidSCAT es muy similar al llevado a cabo con los datos del satélite QuikSCAT, la única diferencia es que el número de datos del instrumento RapidSCAT es mayor. Estos datos no son de gran importancia para nosotros, ya que únicamente los utilizaremos para demostrar la flexibilidad del modelo de datos de nuestro gestor de base de datos.

Los datos que más nos importan son el tiempo, la longitud y latitud, y retrieved_wind_direction, retrieved_wind_speed y rain_impact, que serán los datos que se visualizaran en la aplicación web que explicaremos más adelante.

3.5. Apache Cassandra

En este apartado vamos a explicar todo el proceso que se ha llevado a cabo con el gestor de base de datos Apache Cassandra para la realización de dicho proyecto.

En primer lugar vamos a explicar cómo se crea una base de datos en Apache Cassandra. En segundo lugar, vamos a explicar el diseño que vamos a realizar para la inserción de los datos en dicha base de datos y las diferentes estructuras utilizadas, y por último vamos a explicar cómo se han realizado las consultas.

3.5.1. Creación de la base de datos

Para la creación de lo base de datos, en primer lugar, tenemos que tener instalado nuestro gestor de base de datos como se comenta en los apartados siguientes.

Una vez hecho esto, tenemos que lanzar el servidor de Apache Cassandra como viene explicado en el anexo 2.5.

Cuando ya esté iniciado, podemos crear la base de datos desde dos sitios diferentes, en primer lugar, podemos crearlo desde "*cqlsh*", o también desde DataStax DevCenter, ambas opciones vienen explicadas en el anexo 2.

Para la creación de la base de datos es necesario insertar el siguiente comando:

```
CREATE KEYSPACE if not exists geoespacial WITH REPLICATION =
{ 'class' : 'SimpleStrategy', 'replication_factor' : 1 };
```

Figura 25: Creación de la base de datos en Apache Cassandra

En este caso, nuestra base de datos va a recibir el nombre de "*geoespacial*", a continuación, indicamos la replicación que queremos que exista de nuestra base de datos, en primer lugar indicamos "*SimpleStrategy*", esto quiere decir que vamos a

utilizar un único centro de datos, generando una estructura en anillo. En este apartado también podríamos haber puesto "*NetworkTopologyStrategy*", el cual, permite más de un centro de datos y genera una estructura en red.

Y por último, indicamos el número de nodos de replicación que queremos, en este caso como solo se ha utilizado un equipo, indicamos como "replication_factor" uno.

Para el uso de esta base de datos creada tenemos que poner lo siguiente:

```
use geoespacial;
```

Figura 26: Acceder a la base de datos creada

3.5.2. Diseño de la base de datos

Para llevar a cabo este proyecto, se ha diseñado una base de datos especial. Este diseño contiene dos ColumnFamily obligatorias. La primera ColumnFamily recibe el nombre "*indiceCubos*", ésta nos indicará cual es la ColumnFamily a la que debemos acceder para obtener los datos por los que vamos a realizar la consulta. Para ello, dicha ColumnFamily contendrá dos campos, "*loglatt*" y "*id*".

La segunda ColumnFamily obligatoria, va a recibir el nombre de "*condiciones*", ésta va a contener todos los datos que nos informa el satélite acompañado por un clave que recibirá el nombre de "*loglatt*", de tal forma que contendrá 26 campos más la clave, en este caso solo vamos a poner los campos del satélite QuikSCAT.

Estas ColumnFamily se crean de la forma descrita en la figura 27.

```
CREATE TABLE indicecubos(
  loglatt text PRIMARY KEY,
  id text
);

CREATE TABLE condiciones (
  loglatt text PRIMARY KEY,
  retrieved_wind_speed text,
  retrieved_wind_direction text,
  rain_impact text,
  flags text,
  eflags text,
  nudge_wind_speed text,
  nudge_wind_direction text,
  retrieved_wind_speed_uncorrected text,
  num_ambiguities text,
  cross_track_wind_speed_bias text,
  atmospheric_speed_bias text
);
```

Figura 27: Creación de las tablas en Apache Cassandra

Finalmente, vamos a tener un conjunto de ColumnFamily dependiendo de cómo dividamos la tierra y dependiendo del día, y cada una de ellas tendrá como atributo la longitud, la latitud y el tiempo y una clave. Para visualizarlo mostramos el esquema de la figura 28.

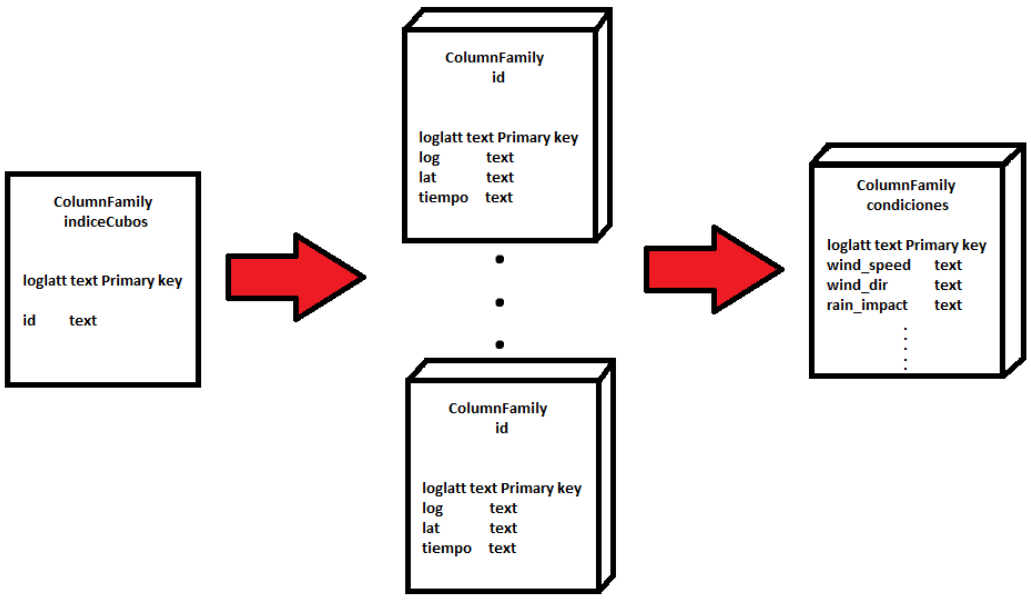


Figura 28: Diseño de la base de datos

El funcionamiento es el siguiente. Si por ejemplo, tenemos dividida la tierra en 4 partes iguales, la primera parte será con latitud 0 hasta 90 y la longitud desde 0 a 180, la segunda parte con latitud entre 0 y 90 y con longitud entre 180 y 360, la tercera parte con latitud entre 90 y 180 con longitud entre 0 a 180, y por último, la cuarta parte con latitud entre 90 y 180 y con longitud entre 180 y 360.

Por lo tanto, en la ColumnFamily con el nombre de "*indiceCubos*" vamos a tener 4 filas. Estas filas tendrían en el campo "*loglatt*", "*0#0#2005244*", "*180#0#2005244*", "*0#90#2005244*" y "*180#90#2005244*", respectivamente. Esto es así, debido a que unimos la longitud, la latitud y el tiempo, en este orden, mediante el símbolo "*#*" y los valores asignados corresponden a el valor más bajo comprendido entre las longitudes y las latitudes. El último dato añadido, "2005244", correspondería al día 244 del año 2005, el día 244 corresponde al día 1 de septiembre. Por lo que esta ColumnFamily actúa como un índice interno que nos ayuda a la hora de realizar una búsqueda.

El siguiente campo que recibe el nombre "*id*" corresponde al nombre de la ColumnFamily a la que hace de referencia. Todo este tipo de ColumnFamily empieza con el nombre de "*tablalog__lat__t__*" sustituyendo los guiones por cada una de las partes en las cuales hemos dividido el mapa, estos datos siempre coinciden con los nombrados en el atributo "*loglatt*".

Por lo tanto en este ejemplo, tendríamos 4 ColumnFamily. Cada una de éstas contendrá las latitudes, las longitudes y el día que se comprendan entre la parte del planeta correspondiente. Nuevamente, tendremos un campo llamado "*loglatt*", que será la unión entre la longitud, la latitud y el día de los otros tres campos de la tabla, como ocurre en el caso anterior. Gracias a este atributo, podremos acceder a la ColumnFamily "*condiciones*" para obtener los datos ofrecidos por los satélites de un punto concreto.

3.5.3. Programa de generación de estructura y carga

Una vez entendido como es el diseño de nuestra base de datos, vamos a generar diferentes divisiones del planeta para comprobar cuál es la división más correcta para que el tiempo de consulta sea el más bajo posible.

Para ello, hemos realizado la aplicación descrita en el Algoritmo 4 con el lenguaje de programación Java que se encarga de leer el CSV generado en el proceso de transformación y generar las diferentes estructuras y la carga de éstas en Apache Cassandra.

```
Método leerFicheroCubos(pathFicheroCSV, pathFicheroSalidaCondiciones,
pathFicheroSalidaIndiceCubos, pathFicheroSalidaCubos, opcionSeleccionada,
satélite)
   F = abrimosFichero(pathFicheroCSV)
   Mientras no-fin-fichero
      Time = f.get("time")
      Latitud = f.get("latitud")
      Longitud = f.get("longitud")
      Evaluamos opcionSeleccionada
            caso 0:
               contenidoArbol = saberContenidoArbol1x1(latitud, longitude, time)
               keyArbol = saberKeyArbol1x1(contenidoArbol)
            caso 1:
               contenidoArbol = saberContenidoArbol2x2(latitud, longitude, time)
               keyArbol = saberKeyArbol2x2(contenidoArbol)
            caso 2:
               contenidoArbol = saberContenidoArbol4x4(latitud, longitude, time)
               keyArbol = saberKeyArbol4x4(contenidoArbol)
            caso 3:
               contenidoArbol = saberContenidoArbol8x8(latitud, longitude, time)
               keyArbol = saberKeyArbol8x8(contenidoArbol)
            caso 4:
               contenidoArbol = saberContenidoArbol16x16(latitud, longitude, time)
               keyArbol = saberKeyArbol16x16(contenidoArbol);
      Fin evaluación
      loglatt = longitude + "#" + latitud + "#" + time
      obtenemosCondiciones(satélite)
      si Indicecubos.containsKey(keyArbol) == true
            lista = cubos.get(contenidoArbol)
            lista.add(d)
            cubos.put(contenidoArbol, lista)
```

```
        sino
                IndiceCubos.put(keyArbol, contenidoArbol)
                lista l = new lista
                l.add(d)
                cubos.put(contenidoArbol, l)
        Fin if
    Fin mientras
    escribimosIndiceCubos(pathFicheroSalidaIndiceCubos)
    escribimosCubos(pathFicheroSalidaCubos)
    escribimosCondiciones(pathFicheroSalidaCondiciones)
Fin método
```

Algoritmo 4: Generar diferentes estructuras

La realización de esta aplicación requiere de un método principal que se encarga de realizar todo el proceso relacionado con la estructura y que explicaremos a continuación.

Como podemos observar en el Algoritmo 4, se trata de un método que lee el fichero CSV que le entra por parámetro, y dependiendo de la opción seleccionada en la interfaz de la aplicación, genera una estructura u otra. Estas estructuras pueden ser 5, es decir, podemos dividir el planeta por días únicamente, por días y a su vez dividiendo el mapa de la Tierra en 4 partes iguales, en 8, 64 o en 256 partes iguales. Esto nos proporciona un gran abanico de posibilidades a la hora de elegir la mejor opción.

Hay que añadir que para la clasificación de las diferentes partes en las que hayamos divido el mapa de la Tierra utilizamos un árbol, esto corresponde a la zona inferior del pseudocódigo anterior. Se utiliza dicha estructura porque el acceso a los datos es más rápido que cualquier otra estructura.

Hay que destacar dos métodos por cada estructura, que gracias a ellos podemos generar esto. Vamos a poner el algoritmo de los dos métodos de una estructura, ya que serían iguales para las demás pero haciendo una mayor división del mapa.

```
Método saberContenidoArbol2x2(lat, log, time)
        latitud = obtenerParteEntera(lat)
        longitud = obtenerParteEntera(log)
        Si latitud < 90
                latexit = “lat0”
        Sino
                latexit=”lat90”
        Fin si
        Si longitud < 180
                lonexit = “log0”
        Sino
                lonexit=”log180”
        Fin si
        Devuelve “tabla”+lonexit+latexit+”t”+time
Fin método
```

Algoritmo 5: SaberContenidoArbol2x2

Como podemos observar en el Algoritmo 5, nos ayuda a diferenciar los datos de una zona u otra. En el Algoritmo 6 se muestra el pseudocódigo del método que nos ayuda a generar el atributo “*loglatt*” de la ColumnFamily “*IndiceCubos*”.

```
Método saberKeyArbol2x2(contenidoArbol)
        tiempo = obtenerAñoDia(contenidoArbol)
        Si contenidoArbol contiene (tablalog0lat0)
                Devolver "0#0#" + tiempo

        Sino
                Si contenidoArbol contiene (tablalog0lat90)
                        Devolver "0#90#" + tiempo
                Sino
                        Si contenidoArbol contiene (tablalog180lat0)
                                Devolver "180#0#" + tiempo
                        Sino
                                Devolver "180#90#" + tiempo
                        Fin si
                Fin si
        Fin si
Fin método
```

Algoritmo 6: saberKeyArbol2x2

Por ultimo añadir, que los datos se insertan siempre con valores positivos, por lo que hay que hacer una transformación en el caso de latitud ya que viene informada con un valor entre -90 y 90.

También hay que añadir que respecto a los datos de longitud y latitud presentan tres decimales, en cambio el resto de los datos solamente presentando dos decimales, es decir, que si alguno de los valores tiene menos decimales se le añade un cero al final. Esto se realiza porque en la base de datos se insertar los valores sin comas, por lo que a la hora de recuperarlo es más sencillo colocar la coma donde sea necesaria.

Una vez realizado todos estos filtros sobre los datos nos encargamos de escribirlos en un fichero CSV, cada parte del mapa sería una ColumnFamily por lo que se generara un fichero CSV por cada uno, además del fichero CSV correspondiente a la ColumnFamily "*condiciones*" y la ColumnFamily "*indiceCubos*".

Esta aplicación, también tiene la opción de realizar la carga de los datos, para ello, hay que indicarlo en la interfaz. Este proceso crear un fichero con extensión "*cql*", que son los ficheros que puede leer nuestro gestor de base de datos.

En este fichero tenemos que escribir la ruta de todos los ficheros CSV generados en el proceso anterior, y la creación de las ColumnFamily correspondientes, ya que solamente están creadas las dos ColumnFamily obligatorias. Una vez creado este fichero, se ejecuta un proceso desde Java para realizar la carga. También se podría hacer la carga desde la consola de Ubuntu utilizando la sentencia "*cqlsh –f nombreFichero.cql*".

```
Método iniciar(nombreFichero)
        f = fichero.crearNewFile(nombreFichero)
        f.write("use geoespacialopciones;\n")
        cargarDatosCondiciones(f)
        cargarIndiceCubos(f)
        cargarDatosCubos(f)
        insertarCassandra(resultado)
Fin método
```

Algoritmo 7: Generar fichero CQL

En la figura 29 se puede observar el resultado final del fichero "cql" generado en el proceso anterior.

```
use geoespacial;
COPY indicecubos (loglatt, id) FROM
'/home/toshiba/Escritorio/CSV/indiceCubos.csv';

COPY condiciones (loglatt, retrieved_wind_speed, retrieved_wind_direction,
rain_impact, flags, eflags, nudge_wind_speed, nudge_wind_direction,
retrieved_wind_speed_uncorrected, num_ambiguities,
cross_track_wind_speed_bias, atmospheric_speed_bias) FROM ’
/home/toshiba/Escritorio/CSV/condiciones.csv';

CREATE TABLE tablalog0lat0t2005244 (loglatt text PRIMARY KEY, log text, lat
text, tiempo text);
COPY tablalog0lat0t2005244(loglatt, log, lat, tiempo) FROM
'/home/toshiba/Escritorio/CSV/cubos/tablalog0lat0t2005244.csv';

CREATE TABLE tablalog180lat0t2005244 (loglatt text PRIMARY KEY, log text, lat
text, tiempo text);
COPY tablalog180lat0t2005244(loglatt, log, lat, tiempo) FROM
'/home/toshiba/Escritorio/CSV/cubos/tablalog180lat0t2005244.csv';

CREATE TABLE tablalog0lat90t2005244 (loglatt text PRIMARY KEY, log text, lat
text, tiempo text);
COPY tablalog0lat90t2005244(loglatt, log, lat, tiempo) FROM
'/home/toshiba/Escritorio/CSV/cubos/tablalog0lat90t2005244.csv';

CREATE TABLE tablalog180lat90t2005244 (loglatt text PRIMARY KEY, log text, lat
text, tiempo text);
COPY tablalog180lat90t2005244(loglatt, log, lat, tiempo) FROM
'/home/toshiba/Escritorio/CSV/cubos/tablalog180lat90t2005244.csv';
```

Figura 29: Fichero CQL generado

3.5.4. Programa de consulta

Cargados los datos, nos disponemos a realizar su consulta. Como ya hemos explicado anteriormente, el proceso de consulta consiste en obtener todos los campos de la ColumnFamily “*indiceCubos*” y traerlos a memoria, ahí será cuando iremos recorriéndolos uno a uno y comprobando si los datos que queremos consultar se encuentra en las diferentes ColumnFamily.

Una vez que sabemos qué ColumnFamily necesitamos consultar, obtenemos todos los datos de dicha ColumnFamily, e iremos comprobando mediante la longitud y la latitud si corresponden a la zona consultada, en caso de afirmación, accederemos

a la ColumnFamily "*condiciones*" con su correspondiente clave y obtendremos los datos correspondientes.

Hay que añadir que las consultas se tienen que realizar a partir de 6 valores, es decir, una latitud inicio y fin, una longitud inicio y fin, y una fecha inicio y fin. Y que las latitudes y longitudes formen un rectángulo, siendo siempre la longitud y latitud inicio menor que la longitud y la latitud fin.

Una vez indicado esto, vamos a ver cómo quedaría el método.

```
Método consulta2x2(session, lonIni, latIni, lonFin, latFin, fechaIni, fechaFin)
        outfile = salida.txt
        result = session.execute("select * from indicecubos2x2")
        Para i in result
                datos = i.loglatt.split("#")
                si (latIni >= int(datos[1]) and latIni < int(datos[1]) + 90)or(latFin >=
                int(datos[1]) and latFin < int(datos[1]) + 90))and(lonIni >=
                int(datos[0]) and lonIni < int(datos[0]) + 180)or(lonFin >=
                int(datos[0]) and lonFin < int(datos[0]) + 180))and(int(datos[2]) >=
                fechaIni and int(datos[2]) <= fechaFin):
                        consulta = "select * from " + str(i.id) + "dimension2x2"
                        result1 = session.execute(consulta)
                        Para j in result1
                                Si(j.log >= lonIni and j.log <= lonFin) and (j.lat >=
                                latIni and j.lat <= latFin):
                                        consulta2 = "select * from condiciones where
                                        loglatt = str(j.loglatt)          result2 =
                                        session.execute(consulta2)[0]
                                        outfile.write(resultado)
                                Fin si
                        Fin para
                Fin si
        Fin para
        close(outfile)
Fin método
```

Algoritmo 8: Consulta a la base de datos con estructuras

El resto de las consultas para las diferentes estructuras sería igual, únicamente cambiaría el primer "*if*" ya que las distancias entre las diferentes partes en las que podemos dividir el mapa de la Tierra cambian.

3.6. Creación del índice externo

Una vez obtenidos los resultados de las diferentes consultas llevadas a cabo, se vio que el tiempo de consulta era muy elevado, por lo que se intentó buscar una solución. Esta solución consiste en la utilización de un índice externo para que nos ayude a realizar las búsquedas.

Este índice externo consiste en la utilización de un árbol R, cuyo funcionamiento es ir insertando las coordenadas de los datos de los satélites, de forma que cuando queramos hacer una consulta nos indique cuál es la ColumnFamily en la que se encuentran los datos en Apache Cassandra. Este índice nos ahorrará la consulta a la ColumnFamily "*indiceCubos*".

Este proyecto está realizado con el lenguaje de programación Java y a continuación iremos explicando cada uno de los pasos llevados a cabo para el uso de este árbol R.

3.6.1. Obtención de regiones

Una vez entendido el porqué del uso del árbol R, vamos a ir viendo los pasos que tenemos que llevar a cabo para su posterior uso.

En primer lugar necesitamos generar una lista con todas las coordenadas de un fichero CSV, por lo que indicamos su ruta desde un menú por consola. Para ello, ejecutamos un método que se encuentra en la clase "*HilbertHandler*" que realiza esto.

Como podemos observar en el Algoritmo 9, leemos un fichero y guardamos los campos de longitud y latitud de cada línea de dicho fichero. Estas coordenadas se pasan por parámetros a un método que se encarga de generar el número de Hilbert a partir de ésta. Este número consiste en generar un valor entero, y nuevamente a partir de éste obtener de nuevo la coordenada.

```
Método CSVToHilbertList(archivo)
        File = open(archive)
        Mientras (leer File)
                latitud = coordenadaStringToInt(lecLati, 90)
                longitud = coordenadaStringToInt(lecLon, 0)
                numeroHilbert = HilbertCurve.Hilbert_to_int(latitude, longitud)
                listaHilbert.add(numeroHilbert)
        Fin mientras
        close(File)
        Devolver listaHilbert
Fin método
```

Algoritmo 9: Obtener coordenadas de un fichero CSV

Estas coordenadas contienen dos decimales, y para generar el número de Hilbert es necesario que sean números enteros, por ello, hay que ejecutar el método "*coordenadaStringToInt*" que se encarga de pasar las coordenadas a valores enteros.

Una vez realizado esto, podemos guardar la lista de Hilbert en un fichero, para ello, tenemos que pulsar la opción dos del menú y se ejecutará el método descrito en el Algoritmo 10.

```
Método serializarListaHilbert(listaHilbert, String nombreFichero)
        fos = File(nombreFichero)
        fos.writeObject(listaHilbert)
        close(fos)
Fin método
```

Algoritmo 10: Guardar lista con las coordenadas de un fichero

La ejecución de este método nos permite no tener que ejecutar el Algoritmo 10 cada vez que generemos el árbol R, ya que ese proceso lo guardamos en un fichero.

Una vez que tenemos todo esto, nos disponemos a realizar la inserción en el árbol R. Para ello, es necesario utilizar el método "*HilbertListToIndex*" que se incluye en el Algoritmo 11 que se describe a continuación.

```
Método HilbertListToIndex(listaHilbert, rtree, aparicionesTotales)
        Ordenar(listaHilbert)
        Para i hasta listaHilbert
                coordenadas = int_to_Hilbert(listaHilbert.get(i))
                latitud = coordenadaIntToFloat(coordenadas.get(0))
                longitud = coordenadaIntToFloat(coordenadas.get(1))
                clave = 0;
                Si (latitud < 90.00 && longitud < 180.00)
                        clave = 1
                Sino
                        Si(latitud >= 90.00 && longitud < 180.00)
                                clave = 2
                        Sino
                                Si(latitud >= 90.00 && longitud >= 180.00)
                                        clave = 3
                                Sino
                                        Si(latitud < 90.00 && longitud >= 180.00)
                                                clave = 4
                                        Fin si
                                Fin si
                        Fin si
                Fin si

                Si(rtree.insertarPuntoSiLibre(longitud, latitud, clave))
                        aparicionesActuales++
                        Si(aparicionesActuales == aparicionesTotales)
                                aparicionesActuales = 0
                                regionActual++
                        Sino
                Fin si
        Fin para
Fin método
```

Algoritmo 11: Insertar coordenadas al árbol R

En este Algoritmo 11 podemos observar que en primer lugar ordenamos la lista de Hilbert, permitiéndonos tener todos los puntos de forma ascendente, de manera que las coordenadas más próximas entre sí estarán muy cerca unas de otras. Para entender mejor esto, se puede ir al apartado de introducción en el cual se explica la lista de Hilbert. Posteriormente vamos recuperando los números de Hilbert uno a uno y los vamos reconvirtiendo en unas coordenadas, con su correspondiente latitud y longitud. Una vez hecho esto, comprobamos en que parte del mapa de la Tierra se correspondería la coordenada y le asignamos un valor.

Durante el estudio del tiempo de consulta, comprobamos que la mejor opción era la estructura 2x2, ésta consistía en dividir el mapa de la Tierra en 4 partes iguales, por ello, ahora en el árbol utilizamos dicha estructura.

Una vez que tenemos el valor que corresponde a la zona de la Tierra, realizamos la inserción en el árbol, para ello utilizamos el método "*insertarPuntoSiLibre*", dicho método nos comprobará si tenemos insertado esa coordenada en el árbol, en caso negativo, lo insertará junto con el valor asignado a esa coordenada.

También, tenemos se realiza el control sobre la cantidad de regiones que vamos a tener dentro del árbol, esto corresponde a la paginación del árbol, en nuestro caso vamos a tener como máximo 500 coordenadas por página. Esto es importante a la hora de realizar la búsqueda.

3.6.2. Diseño de la base de datos

El diseño de la base de datos cambia al utilizado anteriormente. Ahora ya no tenemos las dos ColumnFamily obligatorias utilizadas en el caso anterior, sino que únicamente tendríamos las ColumnFamily correspondientes a cada uno de los cubos que generamos, es decir, tendríamos 4 ColumnFamily por día insertado, ya que utilizamos parte de la estructura 2x2.

Ahora, la ColumnFamily "*indiceCubos*" utilizada anteriormente, seria nuestro árbol R, como mejora hay que añadir que no es necesario realizar una consulta a dicha ColumnFamily con su posterior filtro, sino que el árbol R nos indica directamente a la ColumnFamily que tenemos que acceder, para obtener nuestros resultados de la consulta, acelerando todo este proceso.

Respecto a la ColumnFamily "*condiciones*", ya tampoco existe, debido a que cada vez que accedemos a uno de los cubos, obtendríamos toda la información ofrecida por el satélite, valgan o no esos datos para la consulta realizada. Esto tiene una gran ventaja a la hora de realizar las consultas, porque no tendríamos que realizar consultas a la ColumnFamily "*condiciones*" cada vez que una coordenada se encuentre dentro de los campos establecidos en la consulta.

3.6.3. Carga de los datos

Debido a este cambio realizado sobre el diseño de los datos, el proceso de carga también cambia. Para este apartado nuestro objetivo sigue siendo generar archivos CSV para su posterior carga, para ello es necesario pulsar la opción 4 del menú general. En este apartado, tendríamos que insertar la ruta del fichero CSV general, así como indicar la ruta en la cual queremos guardar los datos e indicar si se trata del satélite QuikSCAT o el instrumento RapidSCAT.

El método utilizado para generar el conjunto de CSV, es muy similar al utilizado para el caso anterior, la única diferencia es que escribimos en cada cubo toda la información que antes separábamos en la lista "condiciones" y ya no hay opción a elegir una estructura, sino que siempre va a ser la misma, 2x2.

Una vez que tenemos todos los cubos en memoria, realizamos la escritura en ficheros CSV de cada uno de los cubos, y generamos el fichero "*CQL*" como ocurría en el apartado anterior, para su posterior inserción.

3.6.4. Consultas

Cuando ya tengamos todos los datos insertados con la nueva estructura comentada anteriormente, podemos realizar consultas utilizando el árbol.

Para ello, tenemos que pulsar la opción 5 del menú principal. A continuación se nos pedirá una serie de datos por pantalla que corresponden a la latitud y longitud inicio, la latitud y la longitud fin, y por último, las dos fechas entre las cuales queremos realizar la consulta. Cuando ya tenemos estos campos informados, nos disponemos a ejecutar el siguiente módulo mostrado en el Algoritmo 12.

Como podemos observar en el método del Algoritmo 12, lo primero que hacemos es consultar sobre el árbol R las zonas por las que tenemos que realizar la consulta en la base de datos. Como en el proceso de carga hemos introducido valores entre el 1 y el 4, ahora tenemos que hacer esa transformación, es decir, el 1 corresponde a la zona con la latitud entre 0 y 90, y con longitud entre 0 y 180, el numero 2 corresponde a la longitud entre 0 y 180, y la latitud entre 90 y 180, el numero 3 corresponde a la zona

con latitud entre 90 y 180, y la longitud entre 180 y 360, y por último el numero 4 corresponde a la zona con latitud entre 0 y 90 y la longitud entre 180 y 360.

```
Método consultaCassandra(lonIni, latIni, lonFin, latFin, fechaIni, fechaFin)
        Bw = new File
        It = ObtenerClaveArbol(lonIni, latIni, lonFin, latFin)
        Mientras it tiene siguiente
                valor = obtenerCampo
                Si (valor == 1)
                        tabla = "tablalog0lat0t"
                Sino
                        Si(valor == 2)
                                tabla = "tablalog0lat90t"
                        Sino
                                Si (valor == 3)
                                        tabla = "tablalog180lat90t"
                                Sino
                                        tabla = "tablalog180lat0t"
                                Fin si
                        Fin si
                Fin si
                Para i desde fechaIni hasta fechaFin
                        tablaAUX = tabla + Integer.toString(i)
                        results1 = session.execute("SELECT * FROM+tablaAUX)
                        Para row1 hasta results1
                        Si (row1.("log") >= lonIni)&& (row1.("log") <= lonFin))&&
                        (row1.("lat")) >= latIni))&&(row1.("lat")) <= (latFin))
                                bw.write(salida);
                        Fin si
                Fin para
        Fin mientras
        close(bw);
Fin método
```

Algoritmo 12: Método de consulta con índice externo

Una vez que tenemos el nombre de la ColumnFamily a la que queremos acceder, realizamos la consulta a la misma. Como podemos observar nos traemos toda la tabla a memoria y la gestionamos. Esta gestión consiste en ir comprobando si los datos de la ColumnFamily están dentro de los rangos establecidos en la consulta, esto se hace mediante el "*if*" que vimos en el algoritmo anterior, y en caso de cumplirse escribimos en un fichero el resultado.

3.7. Visualización de los datos con la aplicación web

La última aplicación realizada en este proyecto consiste en generar una aplicación Web para realizar la visualización de los datos. Para ello vamos a utilizar Servlet de Java y el servidor Tomcat.

En este proyecto se pueden diferenciar distintas partes que explicaremos a continuación.

3.7.1. Obtención de los campos a consultar

Como en toda consulta es necesaria la introducción de una serie de campos para realizarlas, en este caso, vamos a rellenar un formulario web con todos los campos necesarios.

Este formulario está realizado con el lenguaje de programación HTML sobre una página JSP. Los campos obligatorios a insertar son la latitud y la longitud inicio y fin, y la fecha de inicio y fin, igual que en los apartados anteriores. También se tiene que indicar el mapa de Google en el cual se desea utilizar para la visualización de los datos, pueden ser dos tipos de mapa, Google Maps y Google Earth. Google Earth deja de dar soporte a esta plataforma a partir del 12 de diciembre del 2015. Por lo que se utilizará siempre Google Maps a partir de esta fecha.

Otra opción del formulario es indicar el formato de los datos de entrada y de salida; es decir, podemos diferenciar entre la latitud positiva, de 0º a 180º, o la latitud entre -90º y 90º. Lo mismo ocurriría con la longitud, podemos elegir entre 0º a 360º, o de -180º a 180º.

Por último, añadir que existe un apartado de validación de estos datos ya que, como se ha comentado varias veces en este documento, las consultas tienen una condición, la cual es que la latitud y la longitud inicio tiene que ser menor que la longitud y la latitud fin.

3.7.2. Consulta de los datos

A la hora de realizar la consulta a la base de datos desde la aplicación web, no se puede utilizar el árbol R, debido a que el servidor ejecuta sus aplicaciones sobre la versión de Java 1.7 y para el correcto funcionamiento a la hora de generar la lista de Hilbert es necesario la utilización de la versión 1.8 de Java. Por ello, se tienen que usar las consultas utilizadas en el apartado 3.5.

Los datos de la consulta serán mostrados en una tabla en HTML para su visualización.

3.7.3. Muestra de los datos en Google Maps y Google Earth

Para realizar la visualización de los datos en Google Maps o Google Earth, se realiza un script en JavaScript, el cual nos va a permitir obtener los datos de la consulta y añadir las coordenadas al mapa.

El funcionamiento de este método consiste en ir recuperando todos los datos de la tabla del HTML en la que se encuentran todos los datos de la consulta. Para realizar esta recuperación hay que tener en cuenta el formato de los datos seleccionado por el usuario en el proceso anterior. Es por ello que tendríamos que realizar la conversión nuevamente para la visualización de la longitud y la latitud, ya que tanto Google Maps como Google Earth utilizan la latitud entre -90º y 90º y la longitud entre 0º y 180º.

Una vez realizada esta conversión, almacenamos en una lista los datos de la consulta correspondientes a la dirección y velocidad del viento junto con el ratio de lluvia, ya que son los campos que vamos a utilizar para la visualización de los datos. El siguiente paso consiste en recorrer dicha lista para ir añadiendo cada línea de la tabla en el mapa, como podemos ver en el método mostrado por el Algoritmo 13, dicho método se ejecutará siempre que se cargue la página.

También hay que añadir, que existe un formulario en la parte de la izquierda de la pantalla, en el cual, podemos seleccionar uno de los 3 campos nombrados

anteriormente (dirección y velocidad del viento, y el ratio de lluvia), junto con un día concreto, esto nos permitirá hacer una visualización de los datos más concreta.

```
Método iniciar()
        Para i hasta finTabla
                date = tablaDatos.date
                wind_speed = tablaDatos.wind_speed
                wind_dir = tablaDatos.wind_dir
                rain_rate = tablaDatos.rain_rate
                Evaluamos (datosSalida)
                        caso 1:
                                longitud = tablaDatos.longitud
                                latitud = tablaDatos.latitud
                        caso 2:
                                longitud = tablaDatos.longitud - 180
                                latitud = tablaDatos.latitud
                        caso 3:
                                longitud =tablaDatos.longitud
                                latitud = tablaDatos.latitud - 90;
                        caso 4:
                                longitud = tablaDatos.longitud - 180;
                                latitud = tablaDatos,latitud - 90;
                Fin evaluación
                datos.push({"longitud" : longitud,"latitud" : latitud,"date" : date,
                "wind_speed" : wind_speed,"wind_dir" : wind_dir,"rain_rate" :
                rain_rate});
        Fin Para
        Para I hasta datos.length
                Si datos[i]["date"] == time
                        var place = datos[i]["latitud"], datos[i]["longitud"]
                        marker = CrearMarker
                Fin si
        Fin para
        pintarCuadradoMaps();
Fin método
```

Algoritmo 13: Insertar marker en Google Maps

El procedimiento llevado a cabo es parecido, únicamente que a la hora de crear los marker del mapa existen unos filtros dependiendo del día y el campo seleccionado.

3.7.4. Descarga del fichero CSV de los datos consultados

Otra opción que nos proporciona la aplicación web es la descarga de un fichero CSV con el resultado de la consulta realizada. Para este proceso se ha utilizado un nuevo Servlet. En dicho Servlet recuperamos los datos obtenidos de la consulta, los cuales se encuentra en "*Session*", esto consiste en una clase Java que nos permite pasar información entre un Servlet con otro.

Una vez recuperado los datos de la consulta realizada, generamos un flujo para realizar la escritura de los datos con su cabecera correspondiente, como podemos ver en el método proporcionado por el Algoritmo 14.

```
Método doGet(request, response)
        a = datosSession
        response.setContentType("text/csv");
        indicamosDescargaFichero()
        o = GenerarFlujo
        header = generamosCabecera
        o.write(header)
        Para i hasta a.size()
                String line = a.get(i).getLongitud() + ", "+ a.get(i).getLatitud() + ", "
                + a.get(i).getDate()+ ", " + a.get(i).getRetrieved_wind_speed() + ",
                "+ a.get(i).getRetrieved_wind_direction() + ", "+
                a.get(i).getRain_impact() + ", "+ a.get(i).getFlags() + ", " +
                a.get(i).getEflags()+ ", "+a.get(i).getNudge_wind_speed() + ", "+
                a.get(i).getNudge_wind_direction()
                +","+a.get(i).getRetrieved_wind_speed_uncorrected() + ", "+
                a.get(i).getNum_ambiguities() + ", "+
                a.get(i).getCross_track_wind_speed_bias() + ", "+
                a.get(i).getAtmospheric_speed_bias() + "\n";
                o.write(line.toString().getBytes());
        Fin para
        close(o)
Fin método
```

Algoritmo 14: Método para descargar fichero CSV

3.7.5. Generación JSON de los datos consultados

De la misma forma que en el caso anterior, la aplicación ofrece la posibilidad de generar un JSON de los datos recuperados de la consulta. Nuevamente se trata de un

nuevo Servlet que recupera los datos de sesión, como podemos ver en el método que desarrolla el Algoritmo 15.

```
Método doGet(request, response)
        a = datosSession
        json = new JSONObject()
        lista = new JSONArray()
        JSONObject datos;
        Para i hasta a.size()
                datos = JSONObject()
                datos.put(Longitud)
                datos.put(Latitud)
                datos.put(Date)
                datos.put(retrieved_wind_speed)
                datos.put(retrieved_wind_direction)
                datos.put(rain_impact)
                datos.put(flags)
                datos.put(eflags)
                datos.put(nudge_wind_speed)
                datos.put(nudge_wind_direction)
                datos.put(retrieved_wind_speed_uncorrected)
                datos.put(num_ambiguities)
                datos.put(cross_track_wind_speed_bias)
                datos.put(atmospheric_speed_bias)
                lista.put(datos);
        Fin para
        json.put("datosGeoespaciales", lista)
        indicamosTipoContenido("application/json")
        escribirJSONGenerado()
Fin método
```

Algoritmo 15: Método para generar JSON

Capítulo 4: Resultados y discusión

4.1. Gráficas de comportamiento

Como ya hemos comentado en otros apartados, el objetivo de este proyecto es realizar un estudio sobre las diferentes estructuras utilizadas en la base de datos, y su posterior consultas sobre éstas, con el fin de que las operaciones de consulta no sean muy costosas, computacionalmente hablando, y nos proporcionen la información lo antes posible.

Para dicho estudio, se han realizado una serie de gráficas para que nos ayuden a decidirnos por la mejor estructura a utilizar. Las consultas realizadas han sido de tres tipos diferentes que explicaremos a continuación.

En primer lugar, se han realizado una serie de consultas espaciales consistentes en recuperar todos los datos de una zona concreta desde 1 día hasta un total de 7. En este caso se obtiene la zona con una longitud y latitud mínima de 240/30 y una longitud y una latitud máxima de 260/50. La gráfica de la figura 30 siguiente corresponde a dicha consulta.

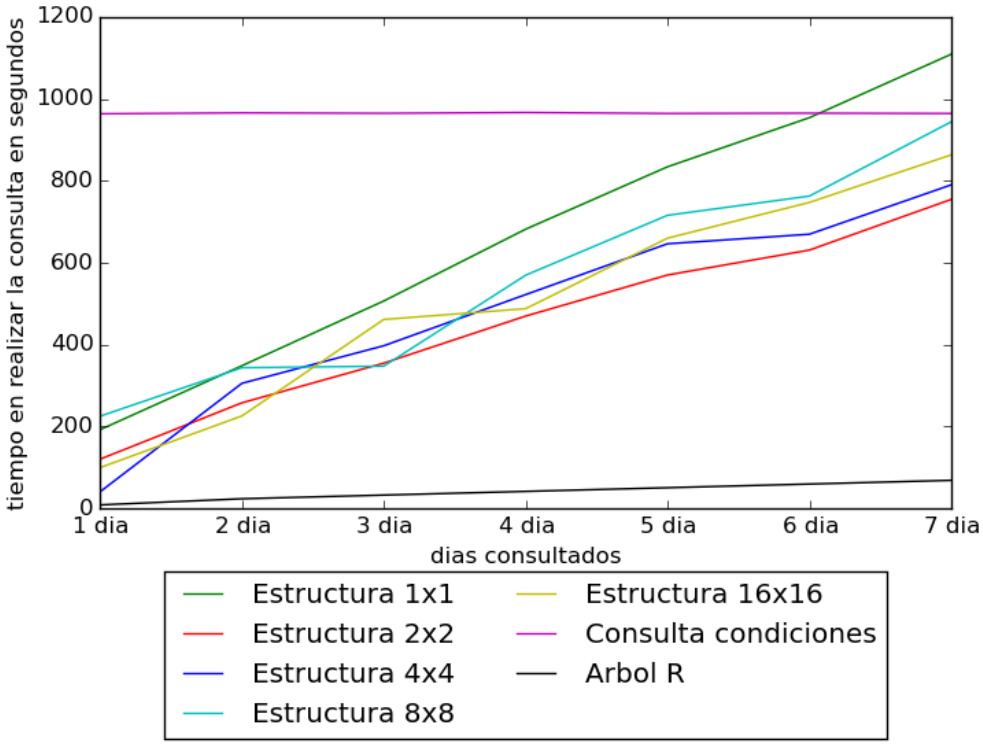


Figura 30: Gráfica espacial

En la figura 30 podemos observar que la estructura 1x1 es la que más tarda en recuperar todos los datos debido a que el filtro a realizar sobre los datos es mayor. En cambio, si realizamos la consulta sin utilizar ninguna estructura y almacenando todos los datos en una única ColumnFamily, el tiempo de consulta sería siempre el mismo, puesto que se recuperarían todos los datos de dicha ColumnFamily. Pero los datos ofrecidos sobre esta consulta no son del todo verdaderos, ya que únicamente están insertados los de dos semanas del mes de septiembre. En el caso de insertar más datos, el tiempo de acceso sería mucho mayor. El resto de las estructuras no nos dan demasiada información para decidir cuál sería la mejor, ya que estarían más o menos todas parejas entre sí.

En cambio, si nos fijamos sobre la línea del árbol R (visualizada en la figura 30 con el color negro) vemos que la diferencia es enorme respecto al resto debido a la eficiencia en acceso que el índice externo proporcionado por el árbol R .

En segundo lugar se ha realizado una consulta temporal como se puede ver en la figura 31.

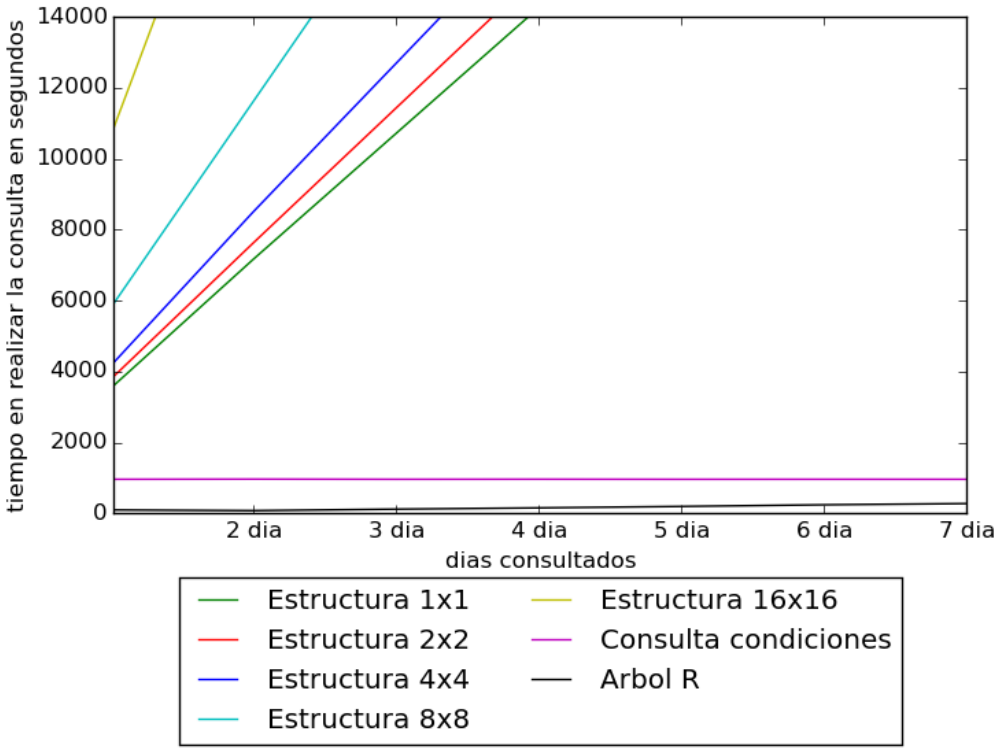


Figura 31: Gráfica temporal

Esta consulta consiste en recuperar todos los datos de 1 día hasta recuperar un total de 7 días. Observamos en la gráfica de la figura 31 que es totalmente inviable utilizar cualquier tipo de estructura. Nuevamente, hay que explicar el caso de no utilizar ninguna estructura como ocurre en el caso anterior. Y también añadir la gran diferencia que existe al utilizar un árbol R.

Por último, se ha realizado una consulta espacio-temporal, que consiste en ir realizando consultas de diferentes zonas de la Tierra de forma incremental en un único día hasta consultar todos los datos de ese día seleccionado.

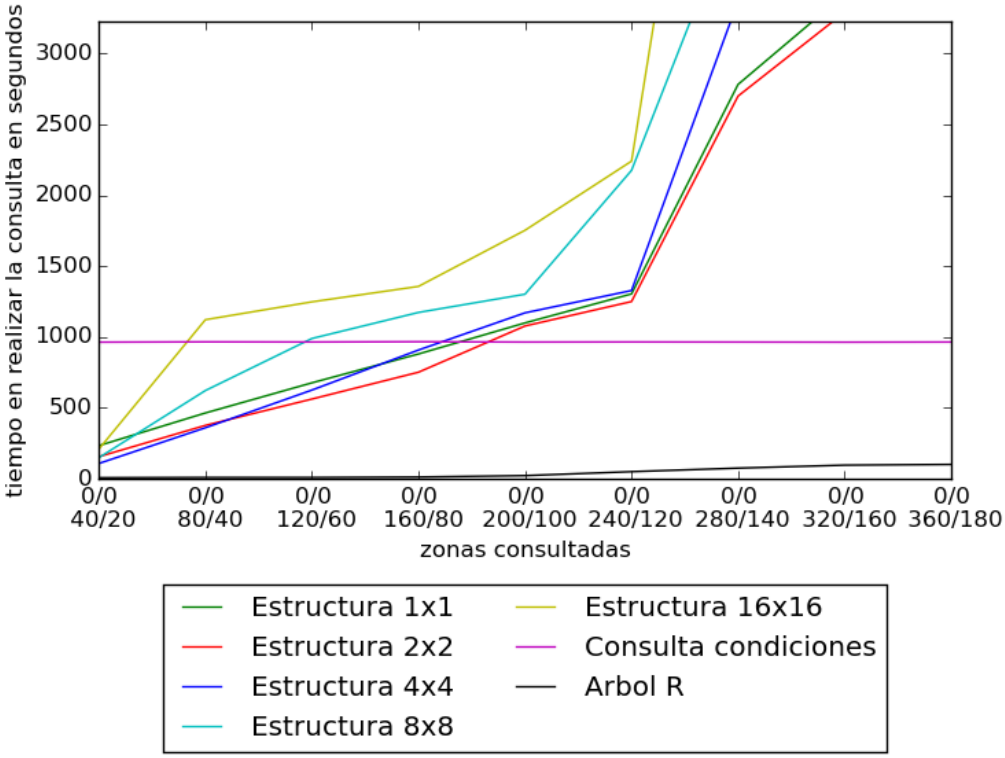


Figura 32: Gráfica espacio-temporal

Como podemos observar nuevamente en la gráfica de la figura 32, las consultas con el árbol R serían las mejores de todas las opciones respecto al uso de alguna de las estructuras.

Si nos ponemos a comparar las tres gráficas generadas, observamos que la mejor estructura, sin contar con el árbol R, ha sido la estructura 2x2. Es por ello que, para el desarrollo del árbol R y sus posteriores resultados, ha sido necesaria la utilización de dicha estructura 2x2.

Como resultado final, podemos decir que se ha mejorado bastante el tiempo de ejecución de las consultas sobre Apache Cassandra gracias al índice externo con el árbol R, ya que las consultas realizadas anteriormente eran inviables debido al gran coste computacional: se realizaban demasiados accesos a la bases de datos y el algoritmo utilizado tenía una complejidad $O(n^3)$, de ahí la gran cantidad de tiempo en espera hasta obtener los datos.

En cambio, con la utilización del árbol R, aunque la complejidad del algoritmo de consulta sigue siendo de $O(n^3)$, el acceso a la base de datos tiene una complejidad de O(n), mejorando formidablemente el tiempo de espera.

4.2. Visualización

Por último, y como resultado añadido al presente Trabajo Fin de Grado también ha de destacarse las visualizaciones de los datos meteorológicos almacenados que se han realizado apoyándose en lo servicios Web proporcionados por Google Maps y Google Earth. Ambos servicios de visualización nos permite interpretar e interactuar de una forma muy simple con los datos obtenidos de las consultas, pudiendo visualizar los datos de diferentes días elegidos por el usuario y de cualquier zona seleccionada del planeta, permitiéndonos ver la formación de diferentes fenómenos marítimos y con ellos analizar visualmente el comportamiento meteorológico registrado.

Un ejemplo de las posibilidades ofrecidas puede observarse en la figura 33. La segunda imagen de la figura 33 muestra la imagen del huracán María obtenida desde el espacio, huracán que se formó durante la primera quincena de septiembre de 2011, y la primera imagen presenta la visualización tanto de la región seleccionada (rectángulo rojo) como los datos obtenidos de la base de datos Apache Cassandra. Los puntos coloreados de la primera imagen de la figura 33 presentan la fuerza del viento siguiendo una escala de color muy simplificada: los puntos rojos corresponden a observaciones de vientos con velocidad superior a 100 km/h; los de color amarillo a velocidades inferiores (este huracán obtuvo una velocidad máxima del viento con 185km/h, y tuvo categoría 3.).

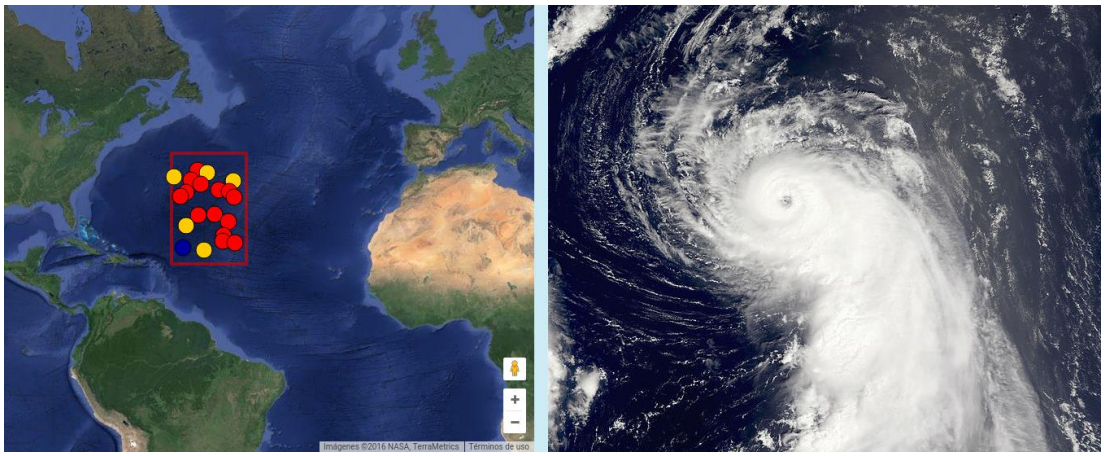


Figura 33: Visualización del huracán María (Wikipedia, 2015f)

Cabe decir que llevar más allá la calidad de las visualizaciones, con una mayor resolución de color, o una figura mucho más ajustada en su representación, es un trabajo que excedería por mucho el tiempo destinado al presente trabajo, quedando entonces fuera de la finalidad y ámbito inicialmente propuesto. No obstante, si que es una propuesta interesante de trabajo futuro para otros Trabajos Fin de Grado que puedan continuar con lo realizado en este.

Capítulo 5: Conclusiones

Tras el desarrollo de este proyecto, he aprendido a trabajar con datos reales y a su vez científicos, ya que nunca había trabajado con ficheros con formato netCDF4 y ha sido un gran reto, tanto entender su estructura como poder tratar los datos.

También he aprendido un nuevo lenguaje de programación como es Python. Este lenguaje ha tenido un gran peso en el proyecto, ya que se ha utilizado en el proceso de transformación de los datos sin el cual el presente trabajo no se podría haber llevado a cabo.

El estudio realizado con las bases de datos NoSQL, en concreto con Apache Cassandra, ha sido muy importante, pues es el núcleo del trabajo. No solamente se ha tenido que comprender su funcionamiento y uso, sino que han debido tomarse una serie de decisiones apoyadas por estudio del comportamiento del motor de la base de datos según el diseño elegido para conseguir una mayor eficiencia en la recuperación. Gracias a estos estudios se ha concluido, que sería necesaria la inserción de un índice espacial externo a la base de datos, produciéndose una gran mejora como ha quedado demostrado en el capítulo anterior.

Hay que añadir que el uso del árbol R ha sido de gran utilidad en la realización de este proyecto porque ha sido una muy buena solución al problema que se nos planteaba en relación con el tiempo de consulta a los datos. La utilización de la curva de Hilbert, asimismo, ha sido de gran importancia a la hora de organizar los datos por proximidad.

Por último, el desarrollo de la aplicación Web ayuda a entender mejor la localización de los datos, ofrecidos por los servicios de Google Map o Google Earth. Destacar la generación de formatos JSON sobre la consulta de los datos, así como la descarga de ficheros CSV de dicha consulta que son utilizados por infinidad de aplicaciones para su posterior interpretación.

Anexo 1. Manual del usuario

2.1. Descarga de los datos

Una vez que tenemos instalado en nuestro equipo el programa Filezilla nos disponemos a abrirlo y nos aparecerá una ventana parecida a esta:

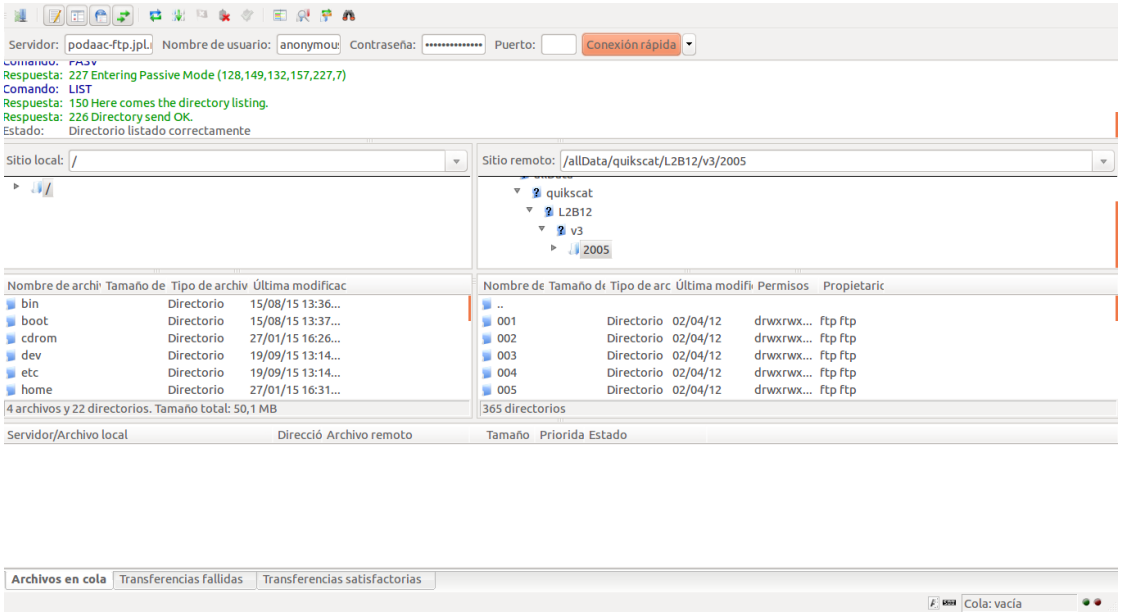


Figura 34: Descargar los datos con Filezilla

Como podemos observar existen diferentes zonas dentro de este programa. Para realizar la descargar tenemos que poner "*anonymous*" en la parte superior izquierda en la casilla con "*Nombre de usuario*", y en la de contraseña "*blah@blah.blah*".

En mi caso, hemos descargado desde diferentes apartados del FTP, ya que hemos utilizados los datos de RapidSCAT y QuikSCAT, por lo que hemos utilizado dos direcciones, en primer lugar utilizamos la dirección "*ftp://podaac-ftp.jpl.nasa.gov/OceanWinds/quikscat/L2B12/v3/2005/*" para descargarnos los datos del satélite QuikSCAT y para RapidSCAT añadiremos "*ftp://podaac-ftp.jpl.nasa.gov/OceanWinds/rapidscat/L2B12/v1.1/2014/*". Estas dos direcciones irán en la parte superior izquierda en el campo "S*ervidor*" y a continuación pulsamos el botón "*conexión rápida*".

Justamente debajo de esto nos indica, si la conexión se ha realizado correctamente, en la parte de la derecha nos aparecerán los diferentes directorios que tiene el FTP y en la parte de la izquierda, nuestro equipo, para indicar dónde queremos guardar los archivos descargados.

Para realizar la descargar únicamente tenemos que seleccionar los archivos que queremos descargar y pulsar botón derecho en el apartado de "*Descargar*".

2.2. Importar proyectos a Eclipse

Para importar proyectos en Eclipse, podemos hacerlo de dos formas diferentes, en primer lugar podemos pulsar con el botón derecho sobre la ventana de "*Project Explore*" a continuación accedemos a "*Import*", y a continuación nos aparecerá una ventana como la de la figura 35. Otra opción es acceder al menú superior y pulsar sobre "*File*" y nuevamente sobre "*Import*" y nos aparecerá la misma pantalla.

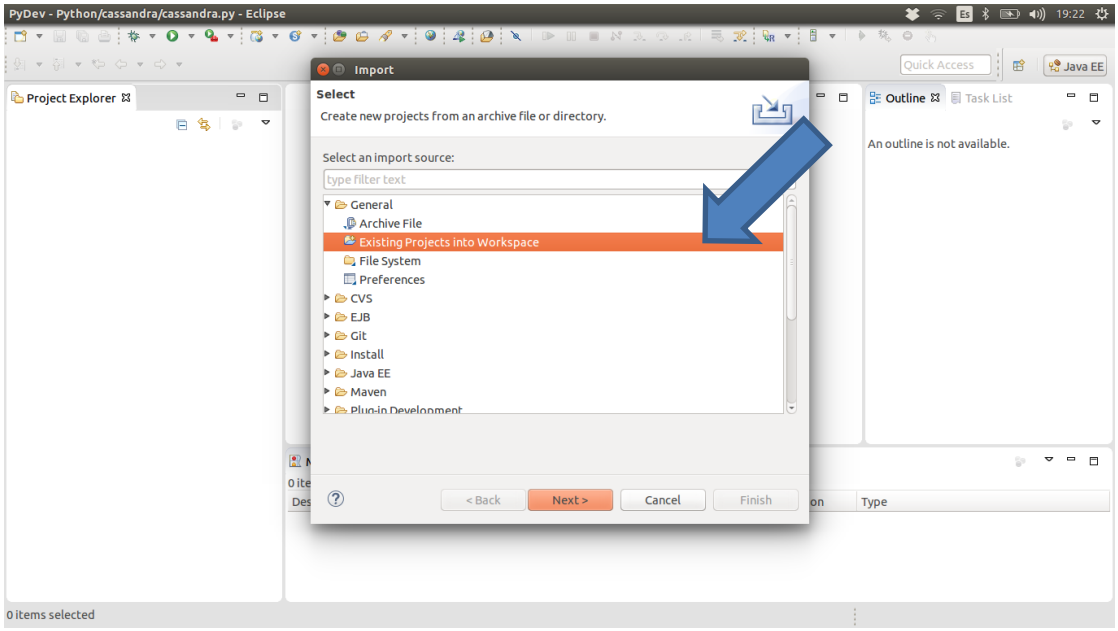


Figura 35: Ventana para importar un proyecto

Como podemos ver en la figura 35 anterior, seleccionamos dentro del apartado "*General*", la opción de "*Existing projects into Workspace*" y pulsamos el botón "*Next*" y nos aparecerá la siguiente pantalla de la figura 36:

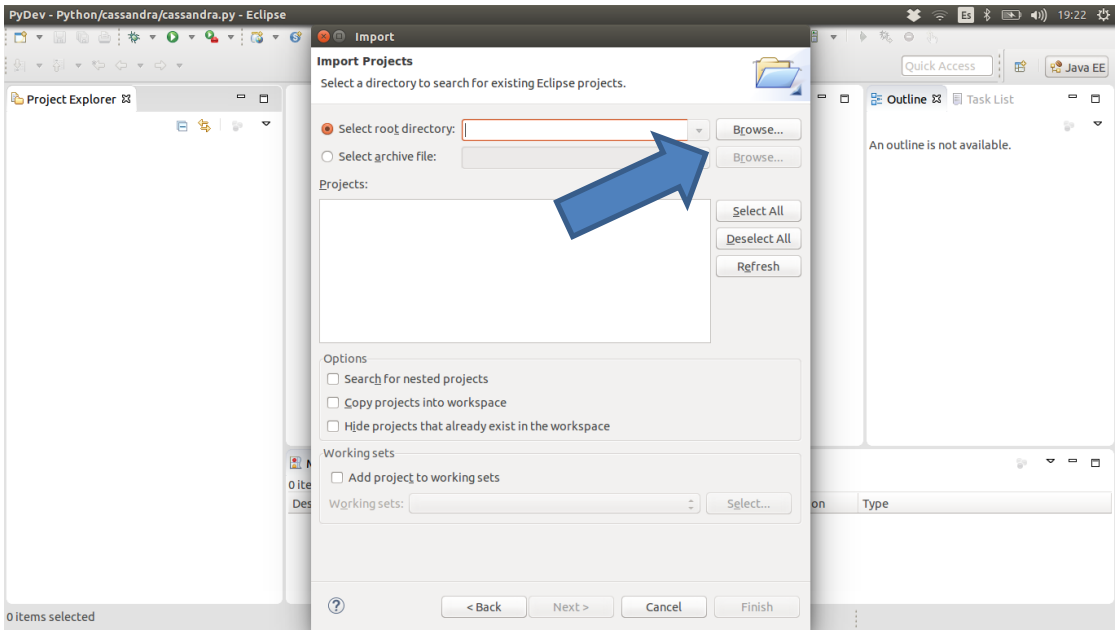


Figura 36: Buscamos el proyecto a insertar

El siguiente paso es buscar nuestro proyecto sobre los directorios de trabajo, para ello tenemos que pulsar el botón "*Browse*" como se indica en la figura 36. A continuación, se abrirá una nueva ventana que nos ayudará a buscar nuestro proyecto en el equipo.

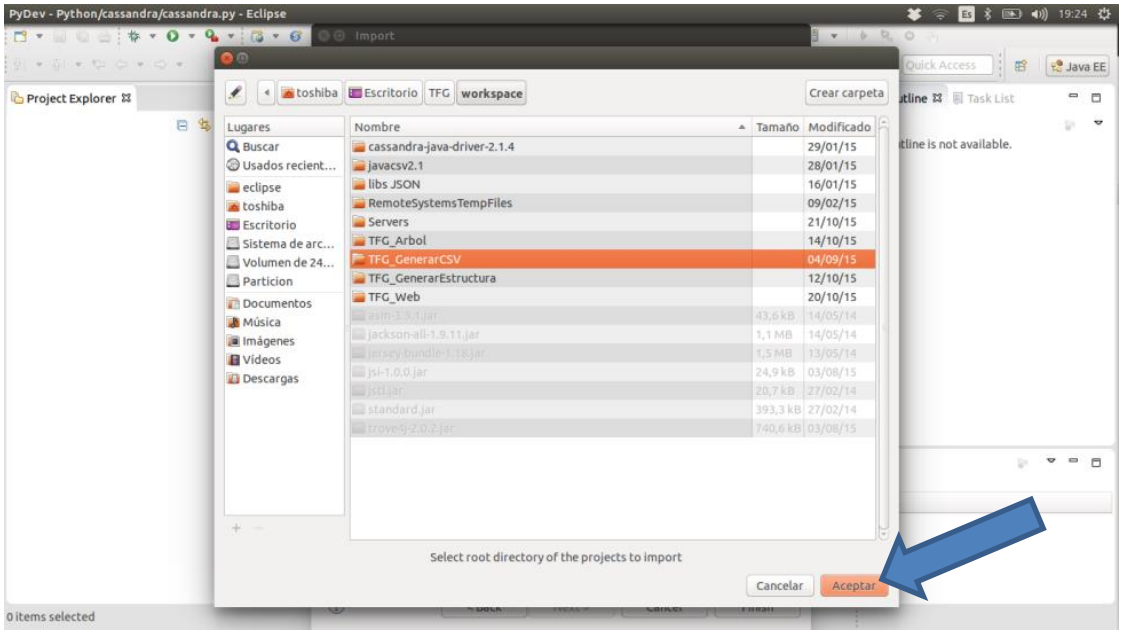


Figura 37: Seleccionamos el proyecto a importar

Una vez que tenemos ya localizado nuestro proyecto en el equipo, lo seleccionamos y pulsamos el botón "Aceptar" (véase figura 37), de tal forma que volvemos a la ventana anterior y tendremos que pulsar la opción "*Copy projects into workspace*" y por último pulsamos el botón "*Finish*" (figura 38), de esta forma quedaría importado un proyecto en el entorno de desarrollo Eclipse.

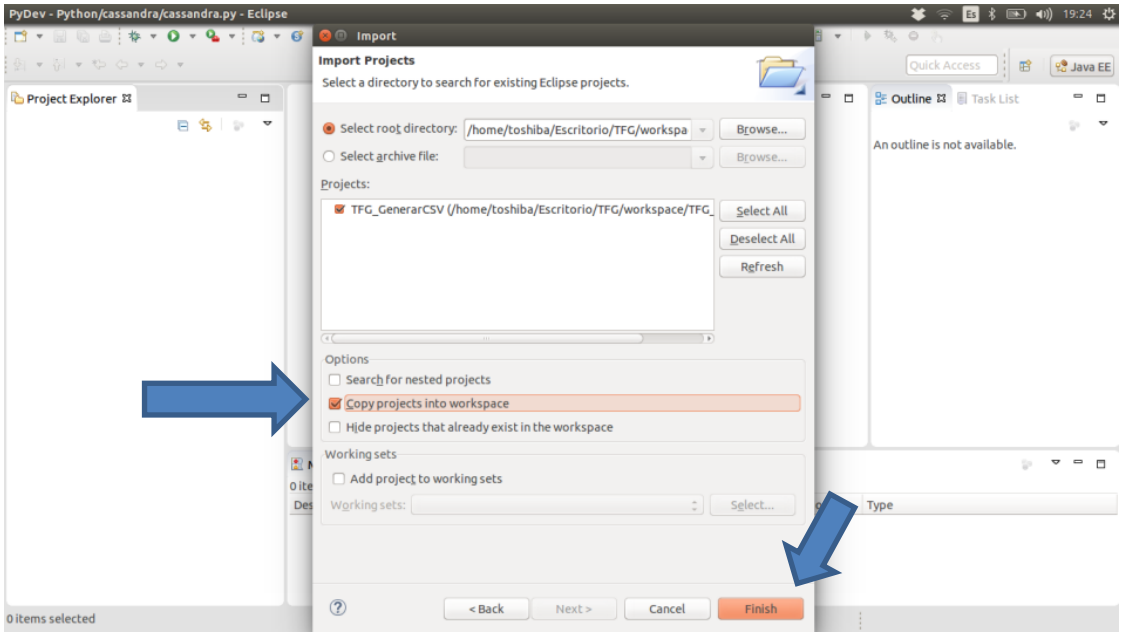


Figura 38: Importamos el proyecto

2.3. Convertidor de netCDF4 a CSV

Una vez que tenemos nuestros archivos descargados en una carpeta, esta contendrá subcarpetas que corresponden a cada día con sus correspondientes datos. A continuación tenemos que acceder a la carpeta con el proyecto que se encarga de la transformación, este proyecto recibe el nombre de "*TFG_GenerarCSV*".

La mejor opción es utilizar la consola de Ubuntu, y movernos hasta el directorio en el que se encuentra el proyecto y ejecutar "*python -O -m* ", y a continuación todos los archivos con extensión python, y después es necesario ejecutar la sentencia "*python*" seguido del archivo "*__init__.py*" que se encuentra en la carpeta "*crearCSV*". Otra opción es añadir el proyecto a Eclipse, como hemos explicado en el apartado anterior y ejecutar dicho archivo.

Una vez importado el proyecto o ejecutado desde consola, nos aparecerá en nuestra pantalla la ventana mostrada en la figura 39.

Figura 39: Convertidor de ficheros netCDF4 a CSV

Como podemos observar en la figura 39, tendremos diferentes opciones. En primer lugar tenemos que seleccionar la carpeta en la cual se encuentran todos los datos descargados. Para ello tenemos que pulsar sobre el botón que está a la derecha en el apartado de "*Directorio origen*" y nos aparecerá una ventana para seleccionar dicha carpeta, como vemos en la figura 40:

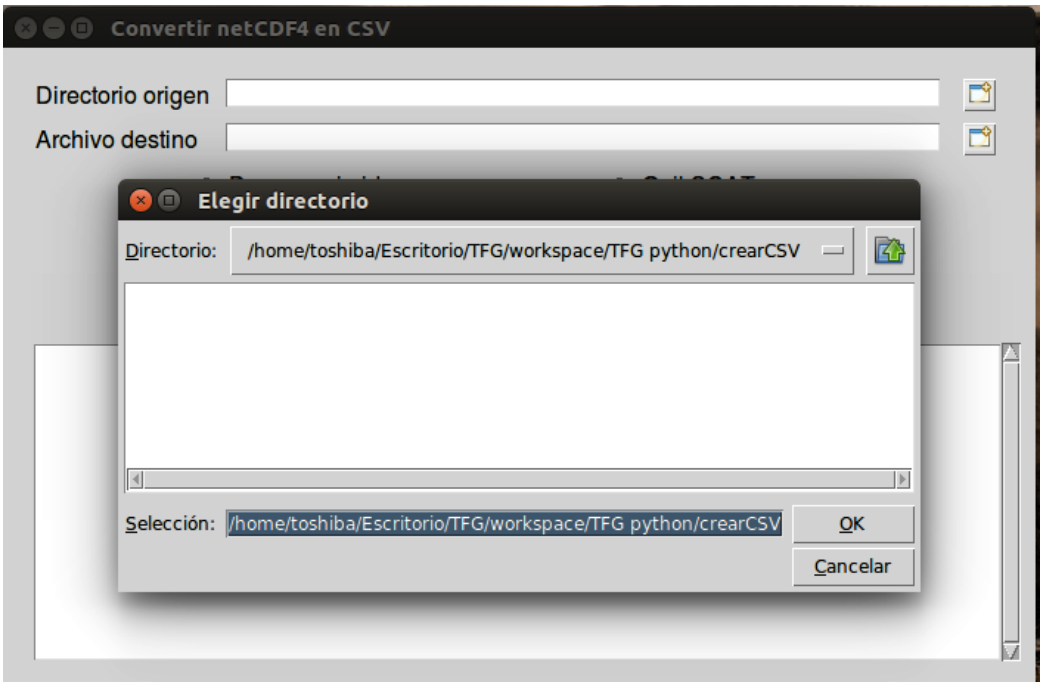


Figura 40: Seleccionar directorios en el proceso de conversión

Observamos en la figura 40 que en la ventana tenemos que ir buscando la carpeta. A continuación hay que realizar el mismo proceso pero en este caso en el apartado de "*Archivo destino*", en él seleccionamos un fichero CSV en el que guardaremos todos los datos, de no existir se creará.

Justamente debajo, tenemos dos opciones, en primer lugar podemos seleccionar que los datos descargados se encuentran descomprimidos o comprimidos con una extensión "*.gz*", y en segundo lugar tenemos que seleccionar si nuestros datos corresponden al satélite QuikSCAT, o por el contrario al RapidSCAT.

Para llevar a cabo el proceso de transformación, es necesario pulsar el botón "*Empezar*" y nos irá mostrando en el apartado inferior todo el proceso que se lleva a cabo durante la transformación, como vemos en la figura 41.

Figura 41: Ejecución del proceso de conversión de los datos

2.4. Crear estructuras para Apache Cassandra

Una vez que tenemos el fichero CSV obtenido del paso anterior, vamos a generar la estructura que deseemos insertar en nuestra base de datos Apache Cassandra, para ello tenemos que acceder al proyecto que recibe el nombre de "*TFG_GenerarEstructura*" e importarlo en Eclipse como explicamos en el anexo 2.2.

Una vez que tenemos nuestro proyecto importado, nos disponemos a ejecutarlo, para ello tenemos que ejecutar el archivo "*interfaz.java*" y nos aparecerá una ventana como la que se muestra en la figura 42.

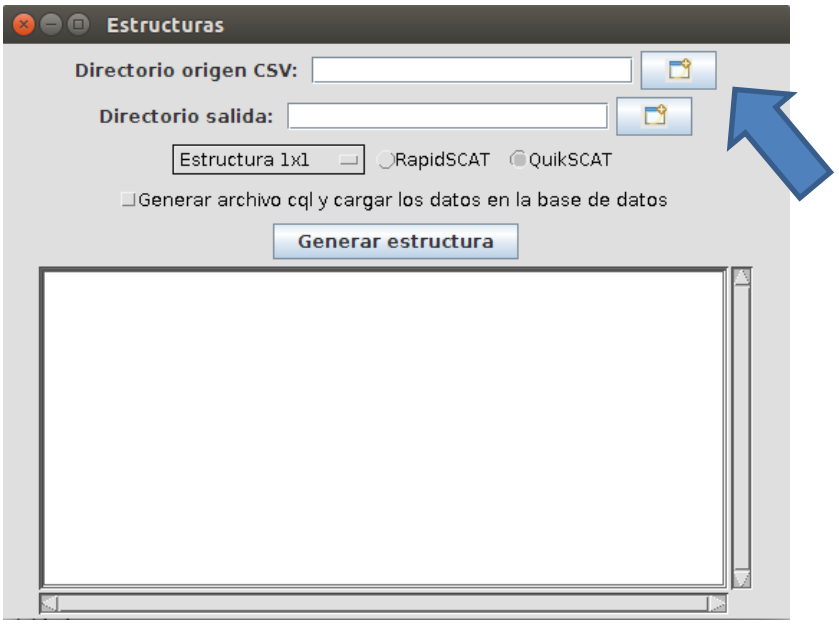


Figura 42: Ventana para generar estructuras

Como podemos observar, tenemos que rellenar una serie de campos para llevar a cabo el proceso. En primer lugar, tenemos que indicar la ruta del fichero CSV. Para ello, pulsamos sobre el botón que aparece en la parte superior derecha, como indica la flecha.

Cuando pulsamos sobre dicho botón, nos aparecerá la ventana que se muestra en la figura 43.

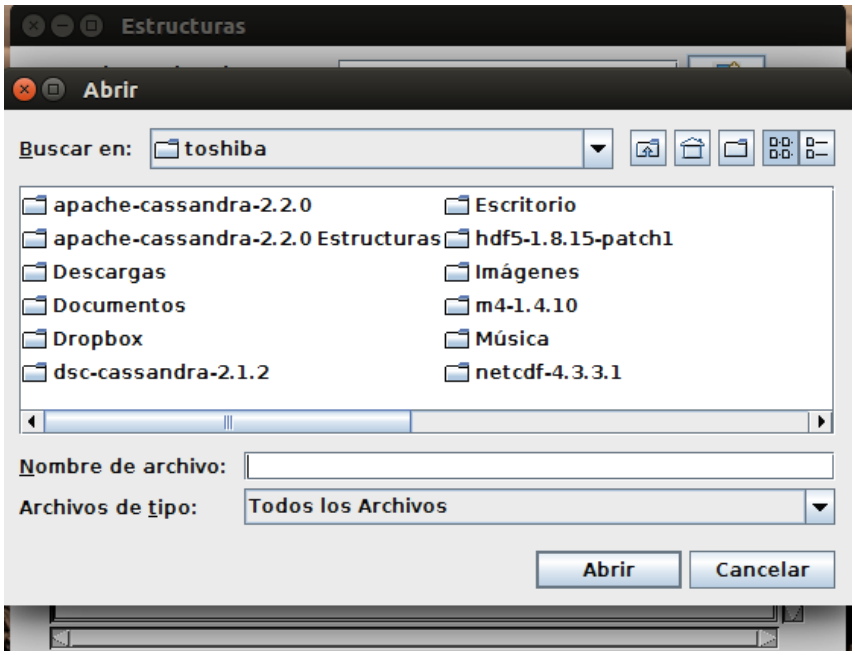


Figura 43: Selección del fichero CSV

Desde esta nueva ventana, tenemos que buscar el fichero CSV del cual queremos generar la estructura.

A continuación, tenemos que realizar el mismo proceso, pero ahora indicaremos la carpeta en la que queremos guardar los diferentes ficheros CSV que se van a generar para obtener la estructura. Esta carpeta tiene que tener una subcarpeta que reciba el nombre de "*cubos*", ya que ahí se van a guardar las diferentes porciones que vamos a generar dependiendo de la estructura.

Las otras opciones que aparecen son para indicar qué tipo de satélite estamos tratando, ya que como hemos comentado en otras ocasiones, los datos son diferentes. Y también nos aparece una lista con todas las estructuras que podemos generar. Podemos encontrar 5 estructuras diferentes. En primer lugar "*Estructura 1x1*", esto quiere decir que dividimos los datos por días, en segundo lugar "*Estructura 2x2*", con esta estructura dividimos el mundo el 4 partes, es decir, la latitud la dividimos entre 0 a 90, y 90 a 180, y la longitud entre 0 a 180 y entre 180 a 360, y además se divide en días también. En el caso de la "*Estructura 4x4*" dividimos el mundo entre 4 la longitud y de igual forma la latitud, y entre días. "*Estructura 8x8*" seria dividir la

latitud entre 8 y la longitud también y entre días. Y de igual forma que "*Estructura 16x16*" pero en 16 partes.

Y, por último, podemos marcar la opción de generar un archivo con extensión CQL. Esto consiste en generar dicho archivo con toda la información necesaria para insertar los datos con su correspondiente estructura en la base de datos Apache Cassandra, pero para ello tiene que estar el servidor de Cassandra en funcionamiento. Esta opción es opcional, todas las anteriores son obligatorias.

Una vez que tenemos todos los campos con la información correcta, tenemos que pulsar el botón "Generar estructura" y nos aparecerá algo parecido a lo mostrado en la figura 44.

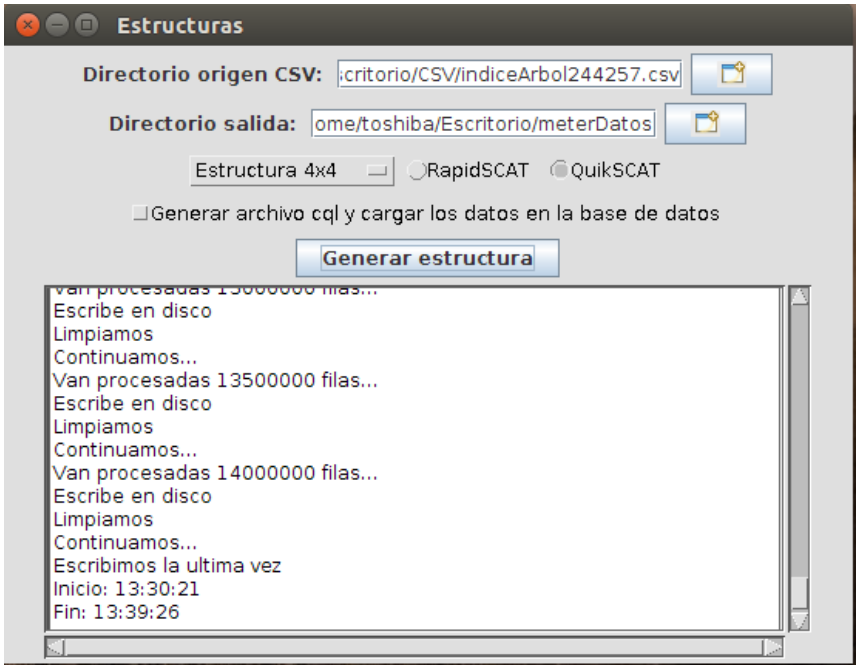


Figura 44: Ejecución del programa TFG_GenerarEstrutura

2.5. Conexión de la base de datos Apache Cassandra

Para realizar la conexión con la base de datos Apache Cassandra es necesario lanzar el servidor de dicha base de datos. En primer lugar hay que realizar todas las operaciones como administrador, para ello es necesario acceder a la consola de

Ubuntu y escribir "*sudo su*". A continuación nos pedirá la contraseña para ser administrador como podemos ver en la figura 45.

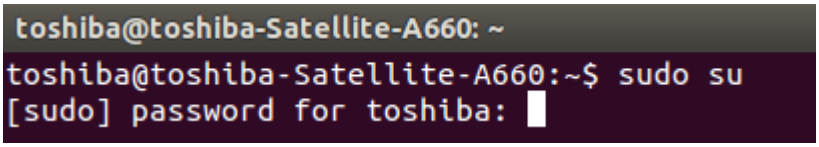


Figura 45: Registro como administrador

Una vez que ya seamos administradores tenemos que comprobar que el puerto que utiliza Apache Cassandra por defecto, no se está utilizando, para ello usamos el comando "*sudo netstat –putan*". Con este comando nos aparecerán todos los puertos que se están utilizando, como podemos ver en la siguiente imagen.

Apache Cassandra utiliza el puerto "*7199*" y como vemos en la figura 46, éste ya se está utilizando, por lo que es necesario matar el proceso y se identifica con PID.

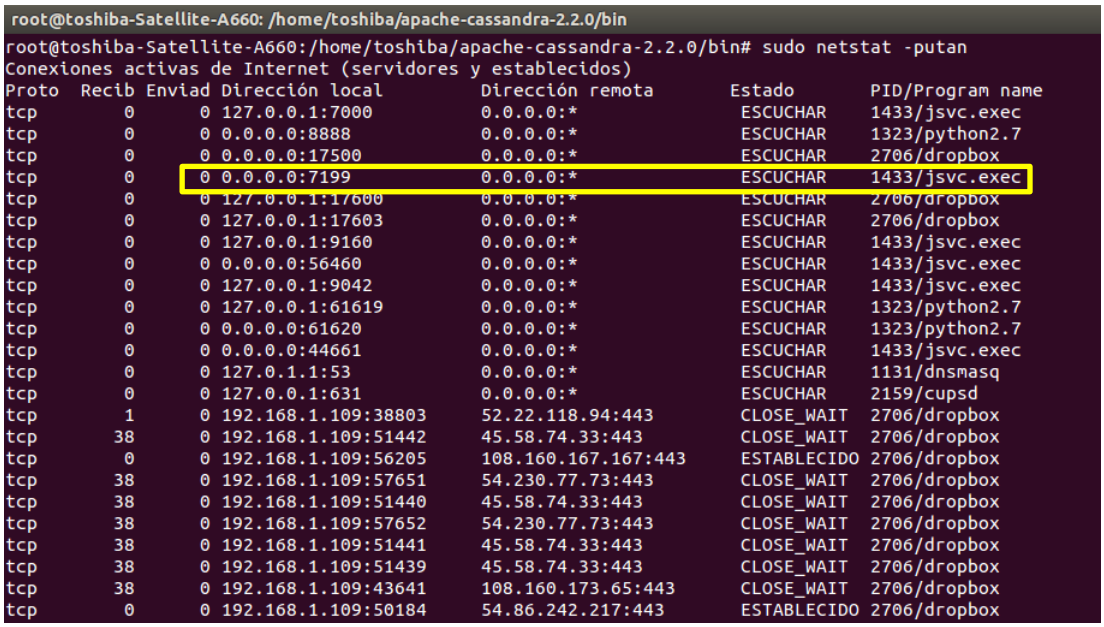


Figura 46: Búsqueda de procesos que utilizan puertos

Para matar el proceso es necesario utilizar el comando "*kill*" seguido del PID del proceso que deseamos matar (figura 47).

```
root@toshiba-Satellite-A660:/home/toshiba/apache-cassandra-2.2.0/bin# kill 1433
root@toshiba-Satellite-A660:/home/toshiba/apache-cassandra-2.2.0/bin#
```

Figura 47: Matamos el proceso que utiliza el puerto 7199

Por último, lanzamos el archivo Cassandra. En este caso, primero tenemos que acceder a la carpeta en la que se encuentra, ya que se trata de una versión portable, en el caso de que esté instalada en el sistema, no es necesario acceder a la carpeta correspondiente. Una vez realizado esto, también podemos acceder al "*cqlsh*" que se encuentra en el mismo directorio. Con esta herramienta podremos realizar consultas, modificaciones, inserciones y borrados de datos y ColumnFamilys que se encuentren en Cassandra.

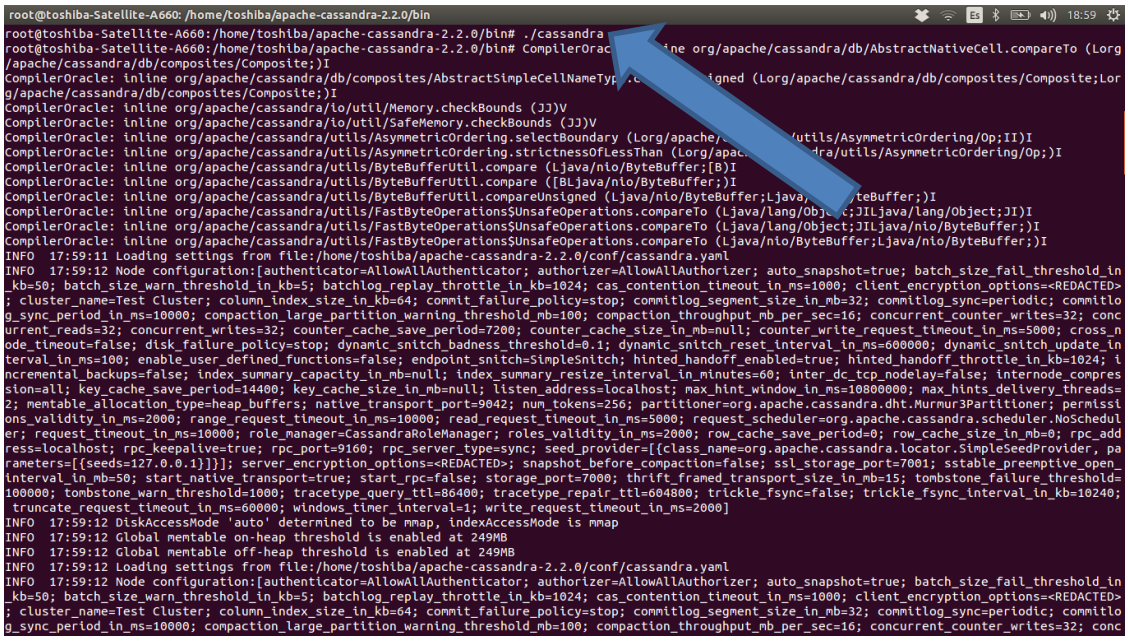


Figura 48: Lanzamos Cassandra

2.6. Utilización de DataStax DevCenter

La herramienta DataStax DevCenter nos permite realizar consultas, crear ColumnFamily o incluso crear keyspace. Para realizar todo esto, tenemos que lanzar la aplicación y a continuación nos aparecerá una ventana, en la que lo primero que tenemos que hacer es realizar la conexión a la base de datos, para ello tenemos que

situarnos en la parte de la izquierda y pulsar sobre el botón de crear conexión y nos aparece la pantalla de la figura 49.

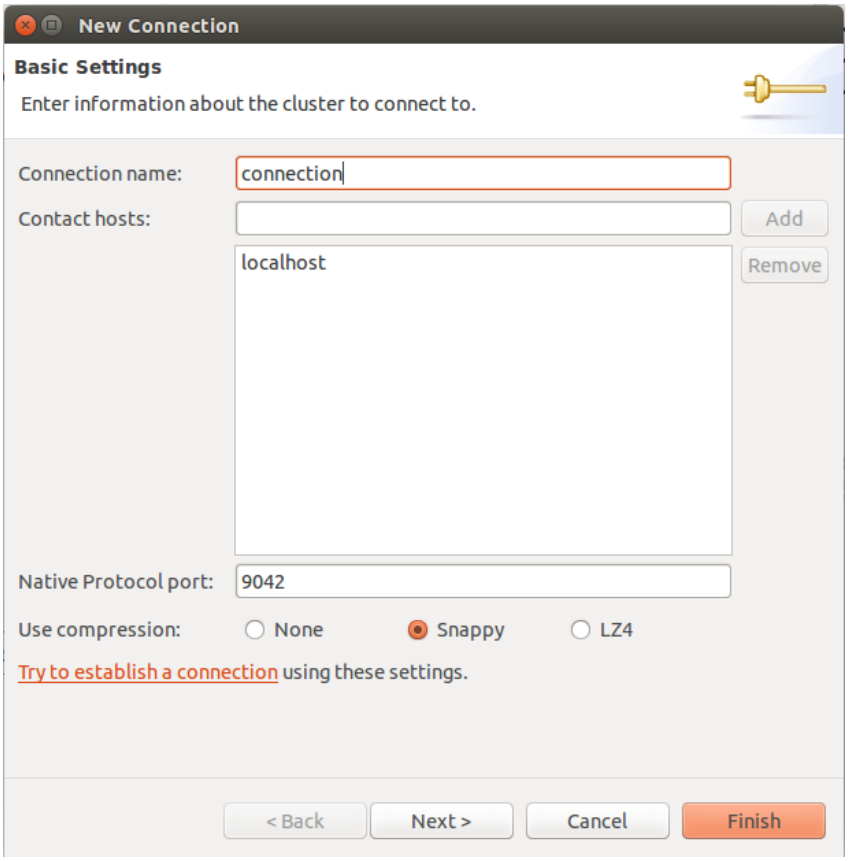


Figura 49: Crear conexión en DataStax DevCenter

Como en nuestro caso la base de datos se encuentra en el mismo equipo, tenemos que poner "*localhost*" en el apartado de "*Contact hosts*" y a continuación pulsamos el botón "*Add*". Por ultimo pulsamos el botón "*Finish*" y ya tendríamos la conexión realizada.

Una vez realizado esto, nos aparecerá, la ventana de la figura 50.

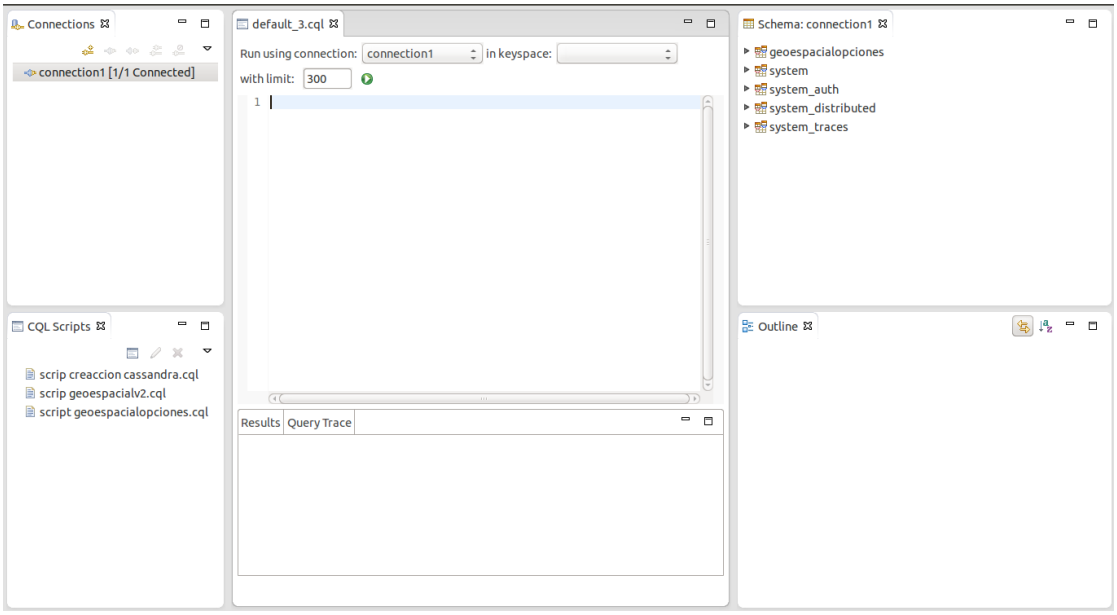


Figura 50: Pantalla DataStax DevCenter

Como podemos observar en la figura 50, tenemos diferentes apartados dentro de la aplicación. En la parte de la derecha, podemos ver todas las keyspace que tiene la conexión y si desplegamos cada una de ellas observaremos las diferentes ColumnFamily que contienen.

En la parte superior izquierda tenemos la conexión que hemos realizado anteriormente, y justamente debajo aparecerán los archivos con extensión ".cql" que tenemos guardados en nuestro equipo.

Y por último, en el centro es donde vamos a realizar las diferentes consultas, para ello utilizaremos el lenguaje CQL. Primero tenemos que indicar la keyspace que vamos a utilizar, esto se indica en el apartado de "*in keyspace*", que es una lista expandible en la que se encuentra todas las keyspace de la conexión. Una vez que tenemos escrita la sentencia CQL, pulsaremos sobre el botón verde para su ejecución. Los datos de salida se mostraran en la ventana inferior.

2.7. Realizar Consultas sobre Apache Cassandra y generar gráficas

Una vez que tenemos la estructura o las diferentes estructuras ya insertadas en nuestra base de datos, podemos realizar consultas sobre ellas. Para ello tenemos que ejecutar el proyecto "*TFG_GenerarCSV*", acceder a la carpeta "*Grafica_tfg*", y ejecutar el fichero Python que hay.

Una vez hecho esto, nos aparece un menú como el de la figura 51.

```
Selecciona una opcion
        1 - Generar grafica temporal
        2 - Generar grafica espacial
        3 - Generar grafica espacio-temporal
        4 - Realizar una consulta
        9 - salir
inserta un numero valor >>
```

Figura 51: Menú para realizar consultas

Como podemos observar en la figura 51, nos aparecen diferentes opciones, en primer lugar se pueden generar las gráficas utilizadas para el estudio de las diferentes estructuras, ya sean las gráficas temporales, espaciales o espacio-temporales.

En cuarto lugar podemos observar la opción para realizar consultas. Si insertamos la opción 4 nos aparecerá lo mostrado en la figura 52.

```
Has pulsado la opcion 4...
pulsa una tecla para continuar
Iniciar conexion
conectado
Selecciona una opcion
        1 - Consulta con la estructura 1x1
        2 - Consulta con la estructura 2x2
        3 - Consulta con la estructura 4x4
        4 - Consulta con la estructura 8x8
        5 - Consulta con la estructura 16x16
        6 - Consulta con la estructura Condiciones
        9 - salir
inserta un numero valor >> 1
inserta latitud inicio >> 0000
inserta longitud inicio >> 0000
inserta latitud fin >> 40000
inserta longitud fin >> 20000
inserta fecha inicio >> 2005244
inserta fecha fin >> 2005244
La tabla consultada ha sido -> tablalog0lat0t2005244
Ha tardado 384.2062201500 segundos
```

Figura 52: Realizar consultas dependiendo de la estructura utilizada

Primeramente se conectará a la base de datos, por lo que es necesario tener el servidor de Apache Cassandra lanzado. A continuación es necesario indicar cuál es la estructura que tenemos en Apache Cassandra para realizar la consulta. Como podemos ver están todas las que hemos nombrado anteriormente durante el proceso de generar la estructura, y además encontramos una más, se trata de la estructura condiciones, ésta consiste en insertar los datos directamente del fichero CSV que se genera durante la transformación de los datos de netCDF4 a CSV.

Una vez seleccionada la estructura, nos aparecerán los diferentes datos de entrada que tenemos que insertar. Como podemos observar en la figura 51, los datos tienen 3 decimales, pero no se añade el punto de separador entre la parte entera y la parte decimal. La consulta se tiene que realizar en forma de rectángulo; es decir, tenemos que indicar en la longitud y la latitud inicio la parte inferior izquierda del rectángulo, y como longitud fin y latitud fin la parte superior derecha, de tal forma que la latitud inicio siempre va a ser más pequeña que la latitud fin, y lo mismo ocurre con la longitud.

Respecto a la fecha a introducir tenemos que indicar cuál es la fecha origen y la fecha destino, en primer lugar tenemos que indicar el año, y justamente a

continuación, el día del año; es decir, sin indicar el mes, en este caso, se ha introducido la fecha "*2005244*".

Posteriormente se nos muestra las ColumnFamily que ha consultado a la hora de realizar la búsqueda y el tiempo que ha tardado en realizarla.

2.8. Uso del árbol R con Apache Cassandra

Para la utilización del árbol R, es necesario importar el proyecto que recibe el nombre de "*TFG_Arbol*" y ejecutar el archivo "*Main.java*", una vez hecho esto nos aparecerá el menú mostrado en la figura 53.

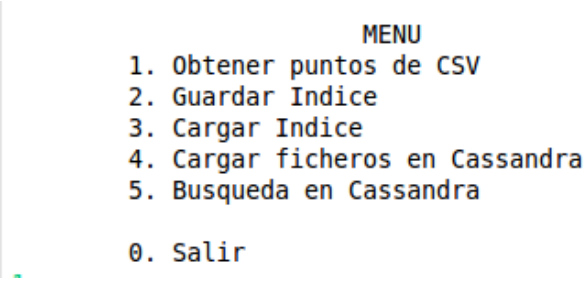


Figura 53: Menú para la ejecución del árbol R

Como podemos observar, tenemos diferentes opciones, en primer lugar tendríamos que realizar la opción 4, para tener todos los datos cargados con la estructura necesaria para el árbol. Cuando pulsamos esta opción nos aparecerá la salida incluida en la figura 54.

```
                    MENU
        1. Obtener puntos de CSV
        2. Guardar Indice
        3. Cargar Indice
        4. Cargar ficheros en Cassandra
        5. Busqueda en Cassandra

        0. Salir
4
Introduzca la ruta del fichero CSV
/home/toshiba/Escritorio/CSV/indiceArbol244257.csv
Introduzca la ruta de salida
/home/toshiba/Escritorio/meterDatos
Indica el satelite (R/Q)
Q
Introduzca la ruta del fichero CQL que se desea generar
/home/toshiba/Escritorio/meterDatos/salida.cql
```

Figura 54: Cargar ficheros para el árbol R

Esta opción es similar al proyecto "*TFG_GenerarEstructura*" pero en este caso solo realizamos la estructura necesaria para el árbol. En primer lugar indicamos el fichero CSV con todos los datos, éste corresponde al fichero obtenido del proceso de transformación de netCDF4 a CSV. A continuación indicamos la ruta en la que queremos guardar los datos de salida. En el siguiente paso indicamos si los datos corresponden al satélite RapidSCAT o QuikSCAT, y, por último, indicamos el fichero de salida con extensión CQL necesario para hacer la inserción de los datos en Apache Cassandra.

Una vez que tenemos todos los datos insertados en la base de datos, vamos a generar nuestro índice. Para ello pulsamos la opción 1.

En este paso, tenemos que indicar la ruta del fichero CSV y obtendrá únicamente los campos longitud y latitud del fichero para generar la lista de Hilbert. Como podemos ver en la figura 55.

```
                    MENU
        1. Obtener puntos de CSV
        2. Guardar Indice
        3. Cargar Indice
        4. Cargar ficheros en Cassandra
        5. Busqueda en Cassandra

        0. Salir
1
Introduzca la ruta de su fichero CSV
/home/toshiba/Escritorio/CSV/indiceArbol244257.csv

Obteniendo puntos de fichero CSV...
```

Figura 55: Obtención de los puntos para la generación del índice

Una vez que obtenemos todos los puntos que hay en el archivo CSV, transformados en la lista de Hilbert, nos disponemos a guardarla, para ello tenemos que pulsar la opción 2 como podemos observar en la figura 56. Este fichero se guardará en el proyecto con el nombre de "*índice*".

```
                    MENU
        1. Obtener puntos de CSV
        2. Guardar Indice
        3. Cargar Indice
        4. Cargar ficheros en Cassandra
        5. Busqueda en Cassandra

        0. Salir
2

Guardando lista Hilbert...
Guardado indice con 14317184 puntos
```

Figura 56: Guardar la lista de Hilbert

Una vez hecho esto, no es necesario realizar los dos pasos anteriores cada vez que ejecutemos esta aplicación, ya que los tenemos guardados en el fichero. Por lo que si queremos realizar consultas únicamente, y ya tenemos guardado el fichero, solamente tenemos que cargar el índice con la función 3 como vemos en la figura 57.

```
                    MENU
        1. Obtener puntos de CSV
        2. Guardar Indice
        3. Cargar Indice
        4. Cargar ficheros en Cassandra
        5. Busqueda en Cassandra

        0. Salir
3

Cargando lista Hilbert...

Obteniendo Indice de regiones a partir de lista Hilbert
Insertados 493685 puntos en 987 regiones
13823499 puntos no han sido insertados
Obtenido indice con 14317184 puntos
```

Figura 57: Cargar índice

Una vez que tenemos esto, podremos realizar consultas sobre el árbol y Apache Cassandra, para ello tenemos que pulsar la opción 5.

```
                    MENU
        1. Obtener puntos de CSV
        2. Guardar Indice
        3. Cargar Indice
        4. Cargar ficheros en Cassandra
        5. Busqueda en Cassandra

        0. Salir
5

Cargando lista Hilbert...

Obteniendo Indice de regiones a partir de lista Hilbert
Insertados 493685 puntos en 987 regiones
13823499 puntos no han sido insertados
Obtenido indice con 14317184 puntos

Introduzca la latitud inicio con 3 decimales (0..180)
0.000
Introduzca la longitud inicio con 3 decimales (0..360)
0.000
Introduzca la latitud fin con 3 decimales (0..180)
40.000
Introduzca la longitud fin con 3 decimales (0..360)
80.000
Introduzca el tiempo de inicio (AñoDia)
2005244
Introduzca el tiempo fin (AñoDia)
2005244

La tabla consultada ha sido -> tablalog0lat0t
tuplas insertadas: 44577 de 114390 posibles
Consulta ejecutada en 9 segundos
```

Figura 58: Realizar consultas sobre el árbol R

En el caso de no tener cargado la lista de Hilbert, cuando pulsemos el opción 5, lo comprueba y la carga si es necesario. Una vez que tenemos el índice cargado, nos disponemos a realizar una consulta. En primer lugar es necesario introducir los datos.

Como ocurre exactamente igual, cuando realizamos las consultas sobre las diferentes estructuras, la longitud y la latitud inicio tiene que ser menor que la longitud y la latitud fin, y la fecha se indica con el año y el día del año, no del mes, de igual forma que el caso anterior. La única diferencia es que a los datos, sí hay que añadirles un punto entre la parte decimal y la entera, pero de igual forma la parte decimal son 3 dígitos.

2.9. Uso de la página web

Para utilizar la página web, es necesario poner en el navegador "*localhost:8080/TFG_Web*" y nos aparecerá una página parecida a la siguiente imagen. Para que se vean correctamente los diferentes apartados es recomendable utilizar el navegador Chrome.

Figura 59: Formulario web

Para rellenar los datos de la zona a consultar, podemos realizarlo de dos formas diferentes. En primer lugar si pulsamos sobre las letras "*Coordenadas en mapa*" nos aparecerá un mapa de Google Maps, en el cual, tenemos que pulsar sobre él para marcar las zonas a buscar. Cuando se realiza el segundo click, automáticamente nos pintará la zona por la que se va a consultar, como podemos ver en la figura 60.

Hay que destacar que las zonas a consultar tienen que tener las dos etiquetas en diagonal para realizar bien la búsqueda.

Figura 60: Selección de coordenadas con Google Maps

Otra opción, es insertar los datos, que corresponden a la latitud y longitud, directamente en los cuatro campos correspondientes. Para que nos aparezcan estos campos es necesario pulsar sobre "*Coordenadas en formulario*".

Hay que destacar que la latitud inicio tiene que ser menor que la latitud fin, de igual forma ocurriría con la longitud.

Figura 61: Selección de coordenadas mediante formulario

A continuación, tenemos que seleccionar como se van a introducir los datos de entrada y de salida, tenemos diferentes opciones, dependiendo de si consideramos la latitud positiva o negativa y la longitud positiva o negativa. Para seleccionar esta opción, pulsamos sobre "*Datos de entrada*" y nos aparecerá una lista con todas las posibles soluciones, y de igual forma sobre "*Datos de salida*", como podemos ver en la figura 62.

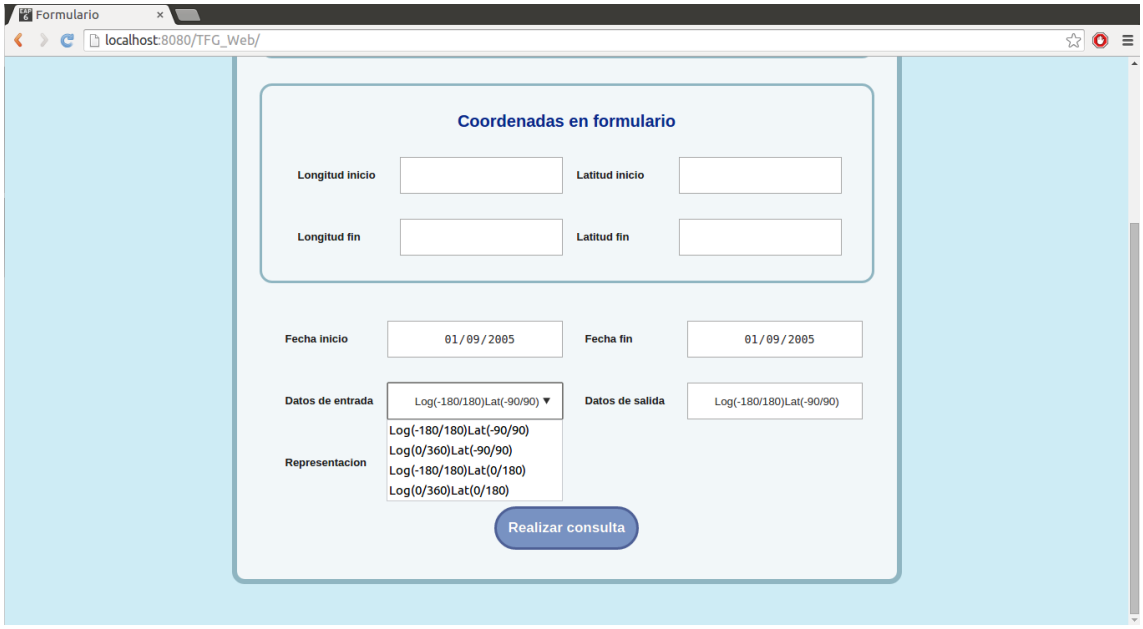


Figura 62: Seleccionar el formato de los datos

El siguiente campo a seleccionar, es la fecha de inicio y de fin, esta fecha nos indicará la cantidad de días que queremos recuperar a la hora de realizar la consulta. En caso de ser únicamente un día, en la fecha de entrada y de salida habría que poner la misma fecha.

Para modificar la fecha tenemos que pulsar sobre el campo "*Fecha inicio*" y se abrirá un calendario para indicar el día, el mes y el año, de igual forma con el campo "*Fecha fin*".

Figura 63: Seleccionar la fecha de consulta

Por ultimo, podemos indicar sobre que mapa queremos visualizar los datos, podemos elegir entre Google Maps y Google Earth.

Una vez que tenemos todo relleno pulsamos el botón "*Realizar consulta*".

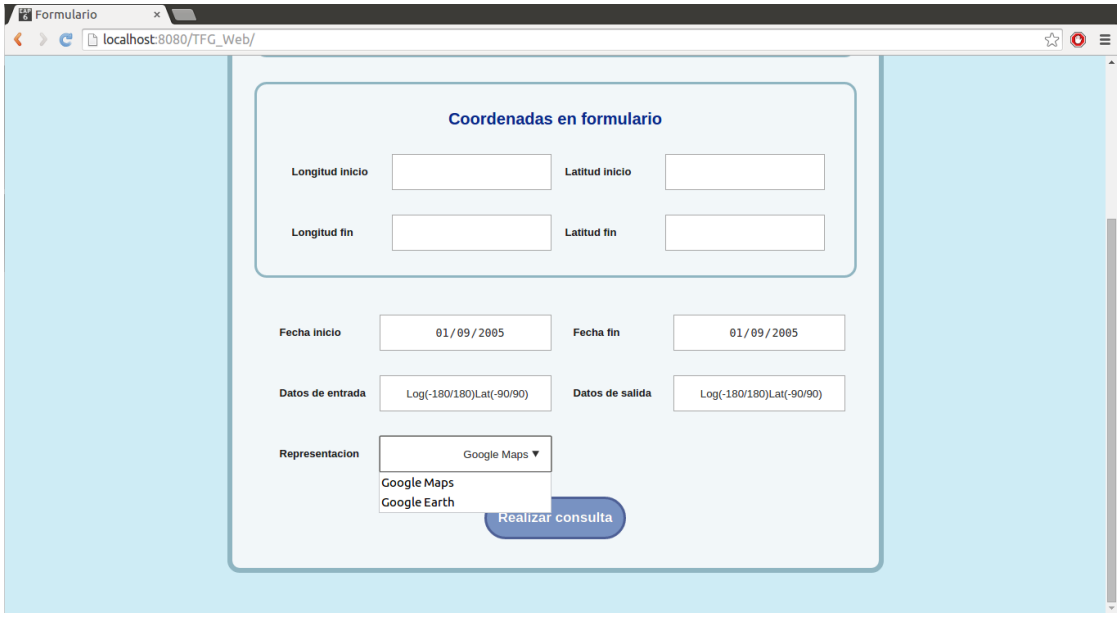


Figura 64: Selección del mapa a utilizar

En el caso de que se nos olvide rellenar algún campo, nos mostrarán mensajes de errores, de igual forma que si la latitud y la longitud inicio son mayores que la latitud y la longitud fin respectivamente.

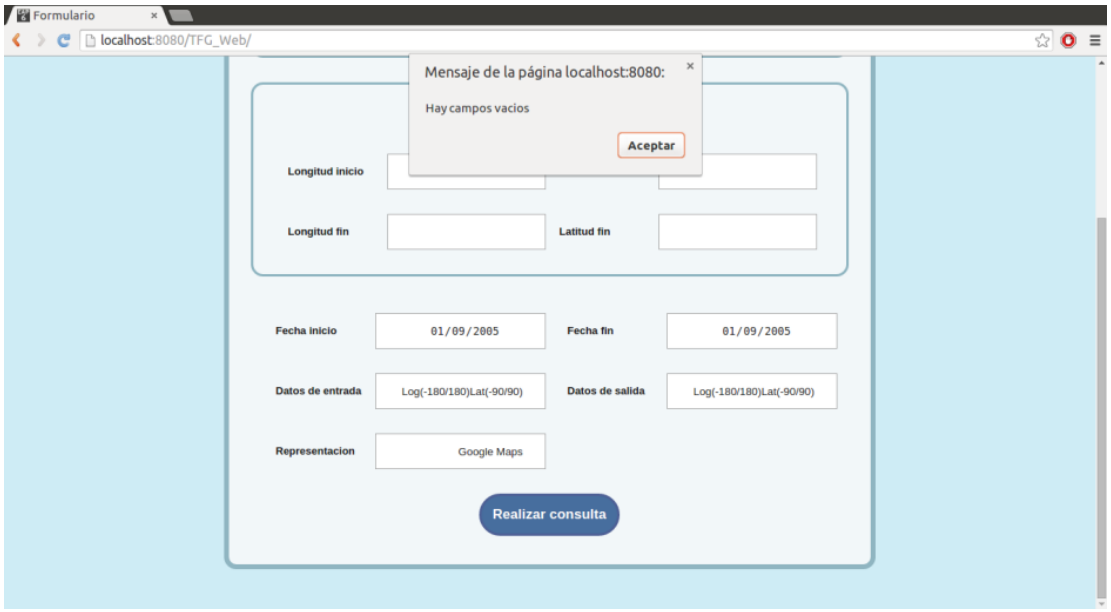


Figura 65: Control de campos vacíos

Si todos los datos han sido válidos, se mostrará otra página que nos indicará que se está realizando la consulta, ya que las consultas pueden llegar a tardar bastante tiempo.

Figura 66: Página de espera

Una vez que ya se obtienen todos los datos, se mostrará una nueva página, lo primero que veremos será parecido a lo mostrado en la figura 67.

Nos aparecerá el mapa que hayamos seleccionado con todos los datos de la consulta. Esta visualización se puede cambiar dependiendo de 3 campos que nos aparecen en la parte izquierda. Como podemos observar tenemos un pequeño formulario con 3 campos de la base de datos que corresponden a la dirección del viento y su velocidad, y el impacto de lluvia. Según seleccionemos uno u otro, se pintaran sobre el mapa. De igual forma podemos cambiar la fecha, para pintar diferentes días sobre el mapa. Una vez que tenemos el campo y la fecha seleccionada, tenemos que pulsar sobre el botón "*Mostrar en el mapa*".

Justamente debajo del formulario nombrado anteriormente, tenemos dos opciones, que son, mostrar un JSON con todos los datos de la consulta o descargarnos un fichero CSV con dichos datos.

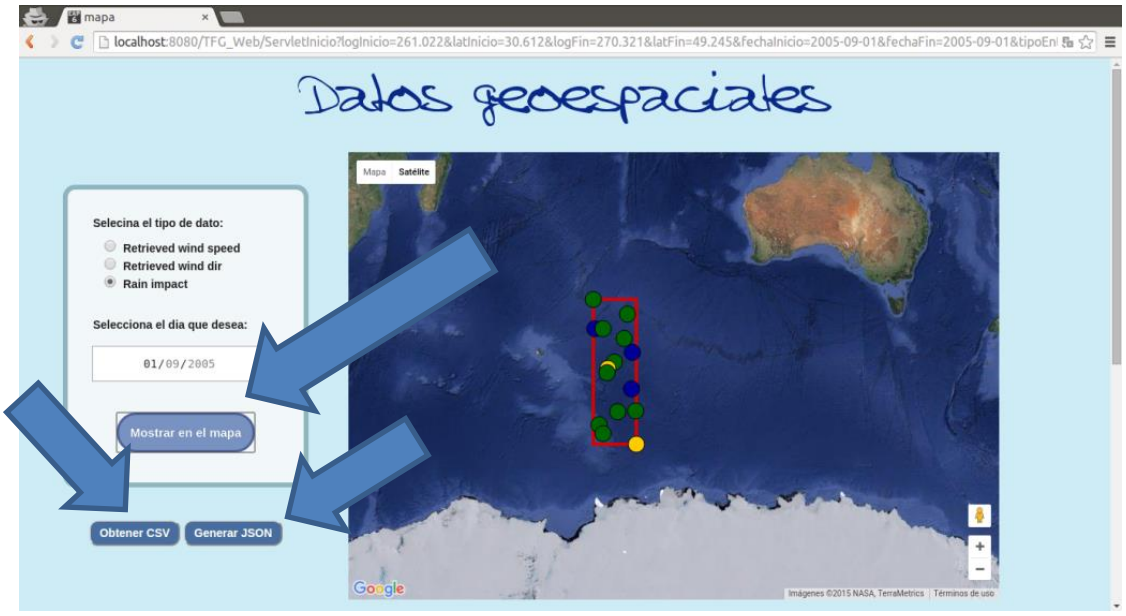
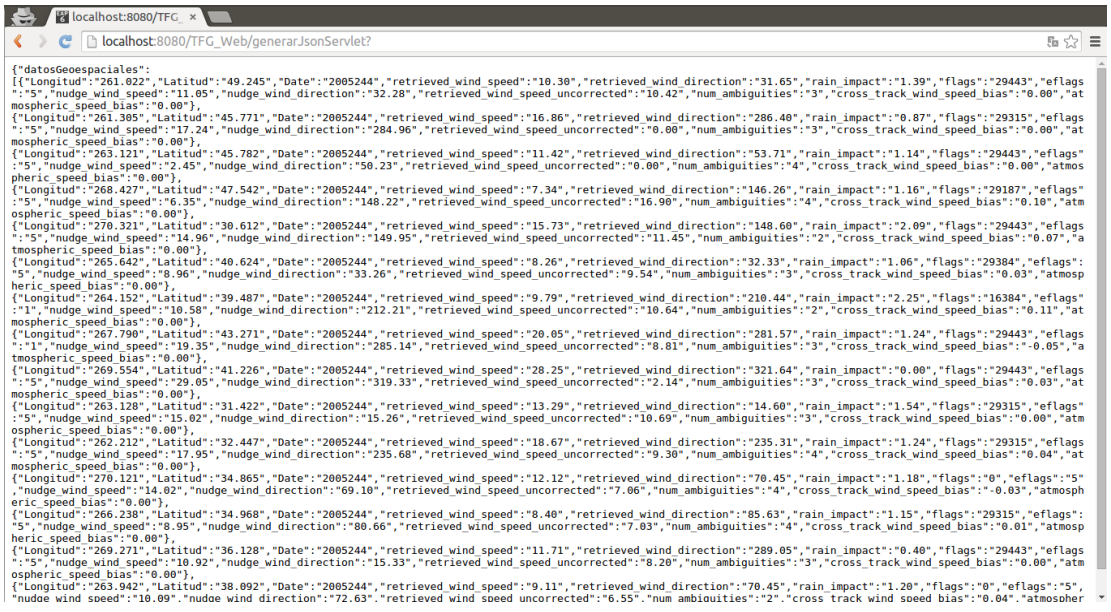


Figura 67: Muestra de la consulta en el mapa

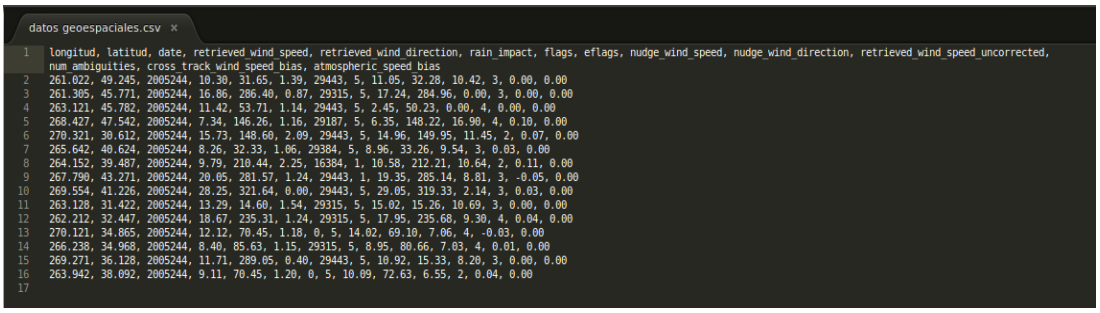
En el caso de haber pulsado sobre el botón "*Generar JSON*" se nos mostrará una nueva página con los datos de la consulta en forma de JSON, como podemos ver en la figura 68.




```
{"datosGeoespaciales":
[{"Longitud":"261.022","Latitud":"49.245","Date":"2005244","retrieved_wind_speed":"10.30","retrieved_wind_direction":"31.65","rain_impact":"1.39","flags":"29443","eflags
":"5","nudge_wind_speed":"11.05","nudge_wind_direction":"32.28","retrieved_wind_speed_uncorrected":"10.42","num_ambiguities":"3","cross_track_wind_speed_bias":"0.00","at
mospheric_speed_bias":"0.00"},
{"Longitud":"261.305","Latitud":"45.771","Date":"2005244","retrieved_wind_speed":"16.86","retrieved_wind_direction":"286.40","rain_impact":"0.87","flags":"29315","eflags
":"5","nudge_wind_speed":"17.24","nudge_wind_direction":"284.96","retrieved_wind_speed_uncorrected":"0.00","num_ambiguities":"3","cross_track_wind_speed_bias":"0.00","at
mospheric_speed_bias":"0.00"},
{"Longitud":"263.121","Latitud":"45.782","Date":"2005244","retrieved_wind_speed":"11.42","retrieved_wind_direction":"53.71","rain_impact":"1.14","flags":"29443","eflags"
:"5","nudge_wind_speed":"2.45","nudge_wind_direction":"50.23","retrieved_wind_speed_uncorrected":"0.00","num_ambiguities":"4","cross_track_wind_speed_bias":"0.00","atmos
pheric_speed_bias":"0.00"},
{"Longitud":"268.427","Latitud":"47.542","Date":"2005244","retrieved_wind_speed":"7.34","retrieved_wind_direction":"146.26","rain_impact":"1.16","flags":"29187","eflags"
:"5","nudge_wind_speed":"6.35","nudge_wind_direction":"148.22","retrieved_wind_speed_uncorrected":"16.90","num_ambiguities":"4","cross_track_wind_speed_bias":"0.10","atm
ospheric_speed_bias":"0.00"},
{"Longitud":"270.321","Latitud":"30.612","Date":"2005244","retrieved_wind_speed":"15.73","retrieved_wind_direction":"148.60","rain_impact":"2.09","flags":"29443","eflags
":"5","nudge_wind_speed":"14.96","nudge_wind_direction":"149.95","retrieved_wind_speed_uncorrected":"11.45","num_ambiguities":"2","cross_track_wind_speed_bias":"0.07","a
tmospheric_speed_bias":"0.00"},
{"Longitud":"265.642","Latitud":"40.624","Date":"2005244","retrieved_wind_speed":"8.26","retrieved_wind_direction":"32.33","rain_impact":"1.06","flags":"29384","eflags":
"5","nudge_wind_speed":"8.96","nudge_wind_direction":"33.26","retrieved_wind_speed_uncorrected":"9.54","num_ambiguities":"3","cross_track_wind_speed_bias":"0.03","atmosp
heric_speed_bias":"0.00"},
{"Longitud":"264.152","Latitud":"39.487","Date":"2005244","retrieved_wind_speed":"9.79","retrieved_wind_direction":"210.44","rain_impact":"2.25","flags":"16384","eflags"
:"1","nudge_wind_speed":"10.58","nudge_wind_direction":"212.21","retrieved_wind_speed_uncorrected":"10.64","num_ambiguities":"2","cross_track_wind_speed_bias":"0.11","at
mospheric_speed_bias":"0.00"},
{"Longitud":"267.790","Latitud":"43.271","Date":"2005244","retrieved_wind_speed":"20.05","retrieved_wind_direction":"281.57","rain_impact":"1.24","flags":"29443","eflags
":"1","nudge_wind_speed":"19.35","nudge_wind_direction":"285.14","retrieved_wind_speed_uncorrected":"8.81","num_ambiguities":"3","cross_track_wind_speed_bias":"-0.05","a
tmospheric_speed_bias":"0.00"},
{"Longitud":"269.554","Latitud":"41.226","Date":"2005244","retrieved_wind_speed":"28.25","retrieved_wind_direction":"321.64","rain_impact":"0.00","flags":"29443","eflags
":"5","nudge_wind_speed":"29.05","nudge_wind_direction":"319.33","retrieved_wind_speed_uncorrected":"2.14","num_ambiguities":"3","cross_track_wind_speed_bias":"0.03","at
mospheric_speed_bias":"0.00"},
{"Longitud":"263.128","Latitud":"31.422","Date":"2005244","retrieved_wind_speed":"13.29","retrieved_wind_direction":"14.60","rain_impact":"1.54","flags":"29315","eflags"
:"5","nudge_wind_speed":"15.02","nudge_wind_direction":"15.26","retrieved_wind_speed_uncorrected":"10.69","num_ambiguities":"3","cross_track_wind_speed_bias":"0.00","atm
ospheric_speed_bias":"0.00"},
{"Longitud":"262.212","Latitud":"32.447","Date":"2005244","retrieved_wind_speed":"18.67","retrieved_wind_direction":"235.31","rain_impact":"1.24","flags":"29315","eflags
":"5","nudge_wind_speed":"17.95","nudge_wind_direction":"235.68","retrieved_wind_speed_uncorrected":"9.30","num_ambiguities":"4","cross_track_wind_speed_bias":"0.04","at
mospheric_speed_bias":"0.00"},
{"Longitud":"270.121","Latitud":"34.865","Date":"2005244","retrieved_wind_speed":"12.12","retrieved_wind_direction":"70.45","rain_impact":"1.18","flags":"0","eflags":"5"
,"nudge_wind_speed":"14.02","nudge_wind_direction":"69.10","retrieved_wind_speed_uncorrected":"7.06","num_ambiguities":"4","cross_track_wind_speed_bias":"-0.03","atmosph
eric_speed_bias":"0.00"},
{"Longitud":"266.238","Latitud":"34.968","Date":"2005244","retrieved_wind_speed":"8.40","retrieved_wind_direction":"85.63","rain_impact":"1.15","flags":"29315","eflags":
"5","nudge_wind_speed":"8.95","nudge_wind_direction":"80.66","retrieved_wind_speed_uncorrected":"7.03","num_ambiguities":"4","cross_track_wind_speed_bias":"0.01","atmosp
heric_speed_bias":"0.00"},
{"Longitud":"269.271","Latitud":"36.128","Date":"2005244","retrieved_wind_speed":"11.71","retrieved_wind_direction":"289.05","rain_impact":"0.40","flags":"29443","eflags
":"5","nudge_wind_speed":"10.92","nudge_wind_direction":"15.33","retrieved_wind_speed_uncorrected":"8.20","num_ambiguities":"3","cross_track_wind_speed_bias":"0.00","atm
ospheric_speed_bias":"0.00"},
{"Longitud":"263.942","Latitud":"38.092","Date":"2005244","retrieved_wind_speed":"9.11","retrieved_wind_direction":"70.45","rain_impact":"1.20","flags":"0","eflags":"5",
"nudge_wind_speed":"10.09","nudge_wind_direction":"72.63","retrieved_wind_speed_uncorrected":"6.55","num_ambiguities":"2","cross_track_wind_speed_bias":"0.04","atmospher
```


Figura 68: JSON de la consulta realizada

En el caso de haber pulsado sobre el botón "*Obtener CSV*" se nos descargará un fichero con extensión CSV con los datos de la consulta y tendrá un aspecto a como se muestra en la figura 69

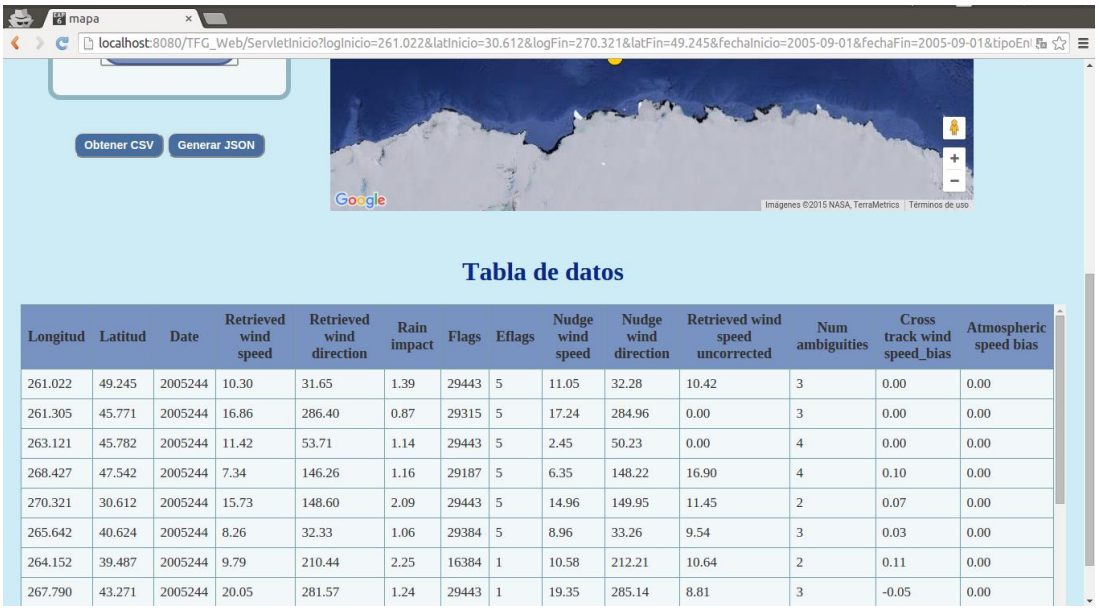


datos geoespaciales.csv

```
longitud, latitud, date, retrieved_wind_speed, retrieved_wind_direction, rain_impact, flags, eflags, nudge_wind_speed, nudge_wind_direction, retrieved_wind_speed_uncorrected,
num_ambiguities, cross_track_wind_speed_bias, atmospheric_speed_bias
261.022, 49.245, 2005244, 10.30, 31.65, 1.39, 29443, 5, 11.05, 32.28, 10.42, 3, 0.00, 0.00
261.305, 45.771, 2005244, 16.86, 286.40, 0.87, 29315, 5, 17.24, 284.96, 0.00, 3, 0.00, 0.00
263.121, 45.782, 2005244, 11.42, 53.71, 1.14, 29443, 5, 2.45, 50.23, 0.00, 4, 0.00, 0.00
268.427, 47.542, 2005244, 7.34, 146.26, 1.16, 29187, 5, 6.35, 148.22, 16.90, 4, 0.10, 0.00
270.321, 30.612, 2005244, 15.73, 148.60, 2.09, 29443, 5, 14.96, 149.95, 11.45, 2, 0.07, 0.00
265.642, 40.624, 2005244, 8.26, 32.33, 1.06, 29384, 5, 8.96, 33.26, 9.54, 3, 0.03, 0.00
264.152, 39.487, 2005244, 9.79, 210.44, 2.25, 16384, 1, 10.58, 212.21, 10.64, 2, 0.11, 0.00
267.790, 43.271, 2005244, 20.05, 281.57, 1.24, 29443, 1, 19.35, 285.14, 8.81, 3, -0.05, 0.00
269.554, 41.226, 2005244, 28.25, 321.64, 0.00, 29443, 5, 29.05, 319.33, 2.14, 3, 0.03, 0.00
263.128, 31.422, 2005244, 13.29, 14.60, 1.54, 29315, 5, 15.02, 15.26, 10.69, 3, 0.00, 0.00
262.212, 32.447, 2005244, 18.67, 235.31, 1.24, 29315, 5, 17.95, 235.68, 9.30, 4, 0.04, 0.00
270.121, 34.865, 2005244, 12.12, 70.45, 1.18, 0, 5, 14.02, 69.10, 7.06, 4, -0.03, 0.00
266.238, 34.968, 2005244, 8.40, 85.63, 1.15, 29315, 5, 8.95, 80.66, 7.03, 4, 0.01, 0.00
269.271, 36.128, 2005244, 11.71, 289.05, 0.40, 29443, 5, 10.92, 15.33, 8.20, 3, 0.00, 0.00
263.942, 38.092, 2005244, 9.11, 70.45, 1.20, 0, 5, 10.09, 72.63, 6.55, 2, 0.04, 0.00
```

Figura 69: Fichero CSV de la consulta realizada

Por último, justamente debajo del mapa y del formulario tendremos una tabla con todos los datos de la consulta, que tiene un scroll para ir viendo todos los datos (ver figura 70). En esta tabla podremos apreciar todos los datos que tenemos recogidos en la base de datos sobre las zonas concretas por las que hemos realizado la consulta.



Longitud	Latitud	Date	Retrieved wind speed	Retrieved wind direction	Rain impact	Flags	Eflags	Nudge wind speed	Nudge wind direction	Retrieved wind speed uncorrected	Num ambiguities	Cross track wind speed_bias	Atmospheric speed bias
261.022	49.245	2005244	10.30	31.65	1.39	29443	5	11.05	32.28	10.42	3	0.00	0.00
261.305	45.771	2005244	16.86	286.40	0.87	29315	5	17.24	284.96	0.00	3	0.00	0.00
263.121	45.782	2005244	11.42	53.71	1.14	29443	5	2.45	50.23	0.00	4	0.00	0.00
268.427	47.542	2005244	7.34	146.26	1.16	29187	5	6.35	148.22	16.90	4	0.10	0.00
270.321	30.612	2005244	15.73	148.60	2.09	29443	5	14.96	149.95	11.45	2	0.07	0.00
265.642	40.624	2005244	8.26	32.33	1.06	29384	5	8.96	33.26	9.54	3	0.03	0.00
264.152	39.487	2005244	9.79	210.44	2.25	16384	1	10.58	212.21	10.64	2	0.11	0.00
267.790	43.271	2005244	20.05	281.57	1.24	29443	1	19.35	285.14	8.81	3	-0.05	0.00

Figura 70: Tabla con todos los datos de la consulta

Anexo 2. Preparación del entorno

2.1. Instalación de FileZilla

Para llevar a cabo la descarga de los datos, es necesaria la instalación del software libre, FileZilla. Este software se ha instalado sobre el sistema operativo Ubuntu.

La instalación se lleva a cabo desde el centro de software de Ubuntu, en la parte superior derecha tenemos que escribir "Filezilla", a continuación, nos aparecerá dicho software, si aplicamos doble click sobre la imagen nos aparecerá la siguiente pantalla mostrada en la figura 71. Y pulsamos sobre el botón de instalar.

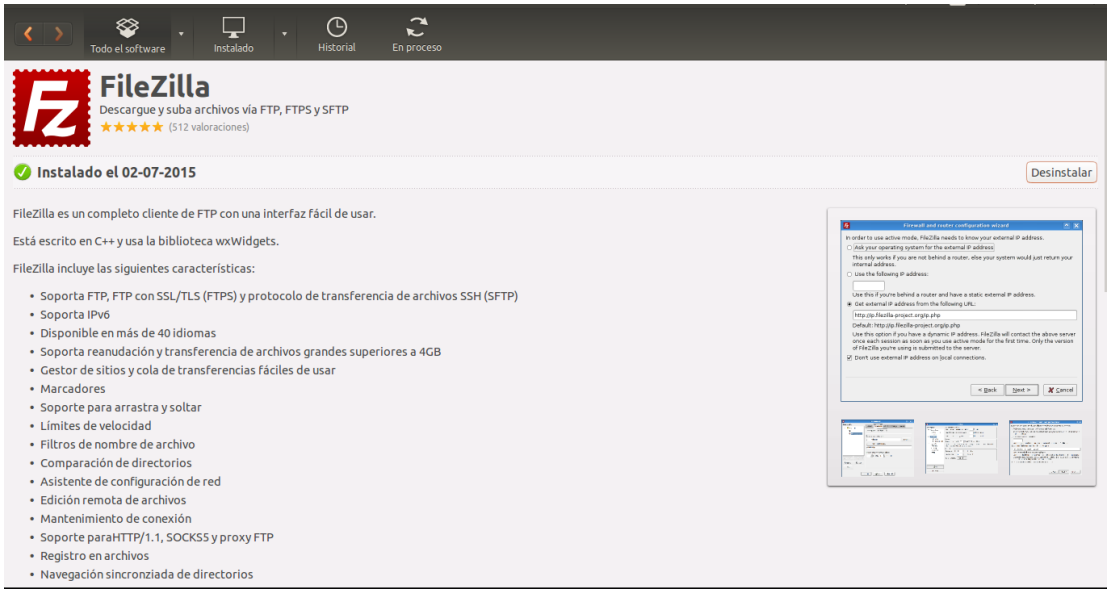


Figura 71: Instalar FileZilla

2.2. Instalación de Apache Cassandra

Para la instalación de Apache Cassandra sobre el sistema operativo Ubuntu, se puede realizar de dos formas diferentes que veremos a continuación.

En primer lugar se puede instalar desde la consola de Ubuntu, y para ello se tienen que llevar a cabo los siguientes pasos:

- En primer lugar registramos una clave pública:

  gpg --keyserver wwwkeys.pgp.net --recv-keys 4BD736A82B5C1B00

- Añadimos la clave pública registrada:

  sudo apt-key add ~/.gnupg/pubring.gpg

- Actualizamos los repositorios de descarga:

  sudo apt-get update

- Por último, instalamos Apache Cassandra:

  sudo apt-get install Cassandra

De esta forma tendríamos instalada nuestra base de datos en el sistema. La otra opción es descargarnos el fichero comprimido que nos ofrece la página web de Apache Cassandra, para ello, tenemos que acceder a la URL http://cassandra.apache.org/download/ , como podemos ver en la figura 72.

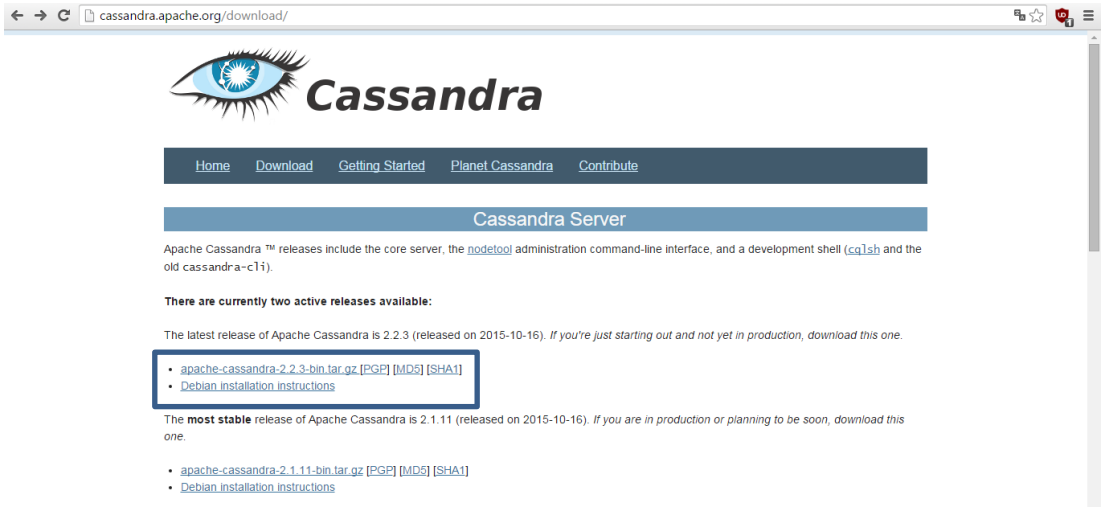


Figura 72: Página web de Apache Cassandra Download

En esta página podemos obtener la última versión de Apache Cassandra o la versión estable de dicha base de datos. En nuestro caso, hemos utilizado la última versión de Apache Cassandra, para su descarga, tenemos que pinchar sobre el enlace

que nos indica el recuadro azul. Una vez hecho esto, nos aparecerá la página Web mostrada en la figura 73.

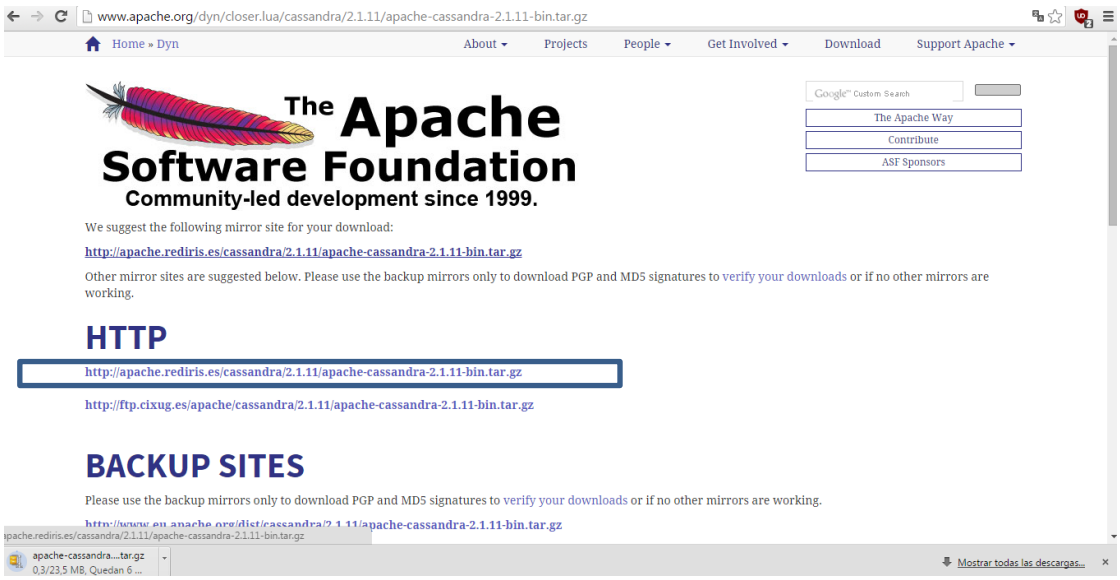


Figura 73: Descargar Apache Cassandra

Una vez que estemos en esta página Web, pulsamos sobre el recuadro que tenemos marcado y comenzará la descarga.

Cuando ya tenemos el archivo descargado, solamente tendríamos que descomprimirlo y ya estaríamos en condiciones para utiliza Apache Cassandra.

2.3. Instalación de Apache Tomcat

Para instalar Apache Tomcat en Eclipse, es necesario tener la versión de Eclipse relacionada con aplicaciones web. A continuación es necesario abrir la vista que recibe el nombre de "*server*". Seguidamente pulsamos el botón derecho dentro de la vista añadida y pulsamos sobre "*add*" y nos aparecerá la pantalla de la figura 74.

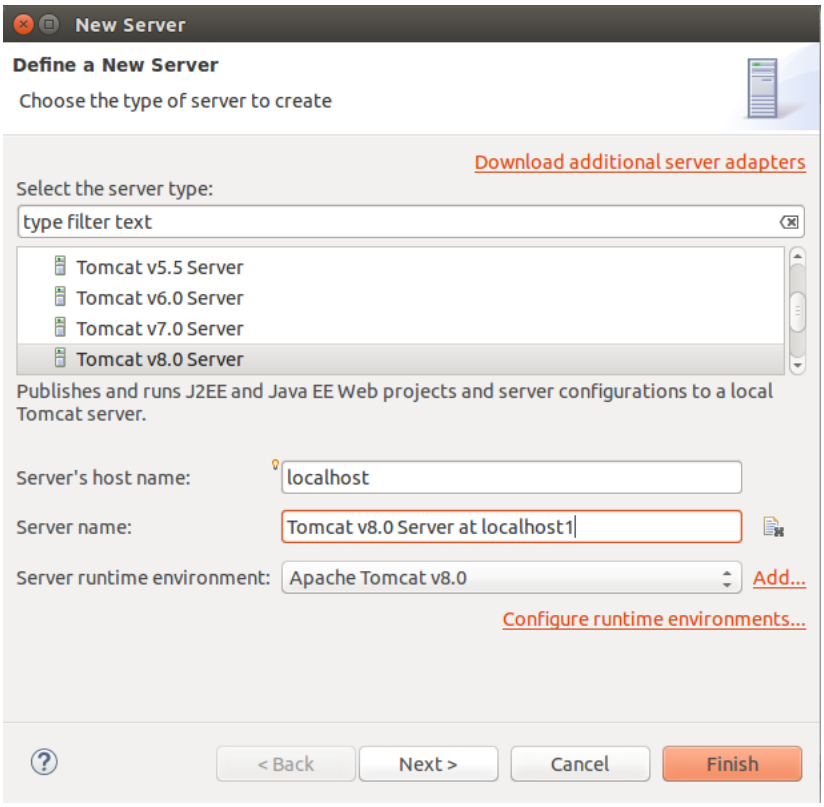


Figura 74: Instalación de Apache Tomcat

A continuación señalamos el servidor a utilizar y la versión de dicho servidor, en este caso, señalamos el servidor "*Apache Tomcat*" con la versión 7. Cuando tenemos estas opciones señaladas, pulsamos el botón "*Next*" y nos aparecerá la ventana de la figura 75.

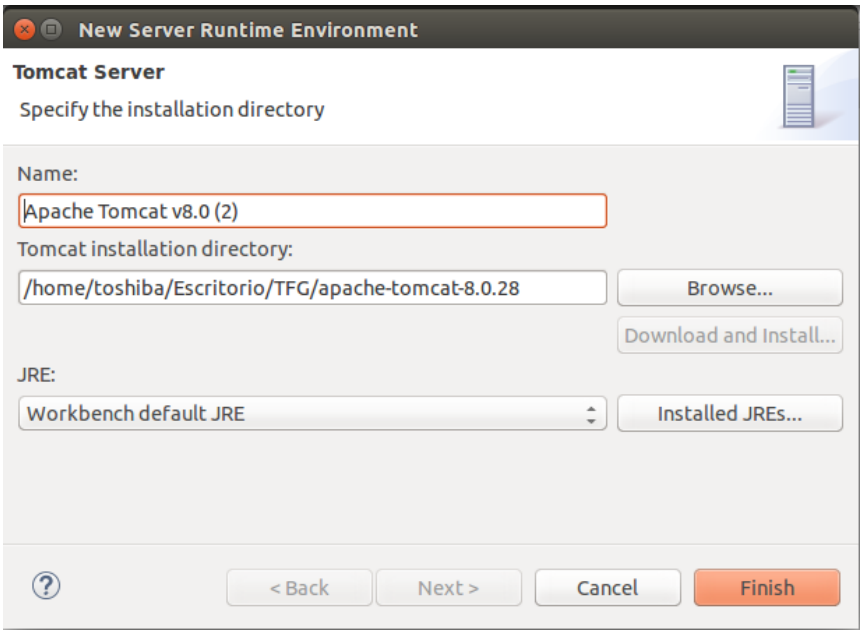


Figura 75: Paso a seguir para la instalación de Apache Tomcat

Como podemos observar en la ventana de la figura 75, podemos realizar la instalación de dos formas diferentes, en primer lugar podemos tener el servidor descargado desde su página web (http://tomcat.apache.org/) y añadir la ruta como vimos en la figura anterior. Y la otra opción, es realizar la descarga del servidor desde dicha ventana, para ello sería necesario pulsar el botón "*Download and install*" pero como podemos ver en este caso esta desactivado ya que tiene una ruta añadida.

Anexo 3. Solución a posibles problemas

3.1. Errores de librerías en Eclipse

Cuando importamos los proyectos en Eclipse nos pueden aparecer errores en los diferentes proyectos, esto puede ser porque no estén correctamente las rutas de las librerías utilizadas. En este caso, nos aparecerá un error parecido al de la mostrada en la figura 76.

consultasCassandra.java

```
1  package Cassandra;
2
3
4  import java.util.HashSet;
14
15 public class consultasCassandra {
16     Rtree indice;
17     Cluster cluster;
18     Session session;
19
20     public consultasCassandra() {
21         // TODO Auto-generated constructor stub
22         cluster = Cluster.builder().addContactPoint("127.0.0.1").build();
23         session = cluster.connect("geoespacialarbol");
24     }
25
26     public void cerrarCassandra() {
27         session.close();
28         cluster.close();
29     }
```

Figura 76: Errores con las librerías

Para solucionar este problema, tenemos que pulsar el botón derecho sobre el proyecto y pulsar sobre "*Properties*", una vez hecho esto nos aparecerá una ventana como la mostrada en la figura 77.

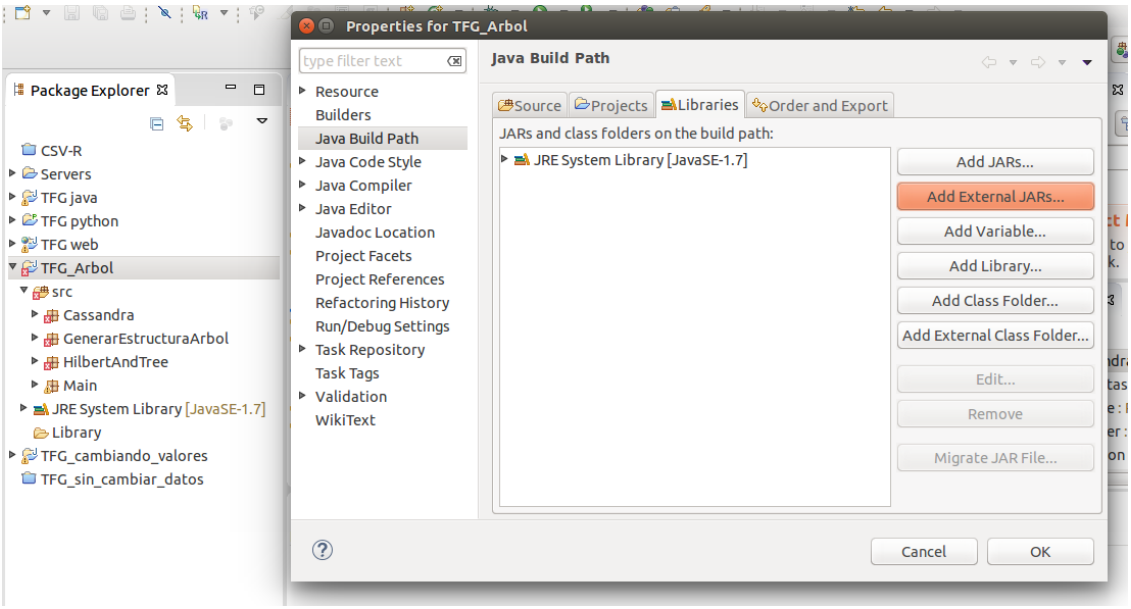


Figura 77: Insertar librerías en Eclipse

A continuación, tenemos que pulsar sobre el botón que aparece a la derecha de la imagen con el nombre de "*Add External JARs*". Con este botón se nos abrirá una nueva ventana, en la que tenemos que seleccionar las librerías correspondientes. Todos los proyectos tienen una carpeta con el nombre de "*Library*" en la cual se encuentran todas las librerías necesarias para la ejecución correcta de dicho proyecto.

Una vez seleccionadas todas las librerías pulsamos el botón aceptar, nuestro problema debe de estar resuelto.

3.2. Error al lanzar el servidor de Apache Cassandra

Cuando lanzamos el servidor de Apache Cassandra pueden ocurrir dos errores, en primer lugar, si no eres administrador o superusuario da un error de lectura y no se puede lanzar.

El segundo error, se trata de la utilización de un puerto en uso como podemos ver en la figura 78.

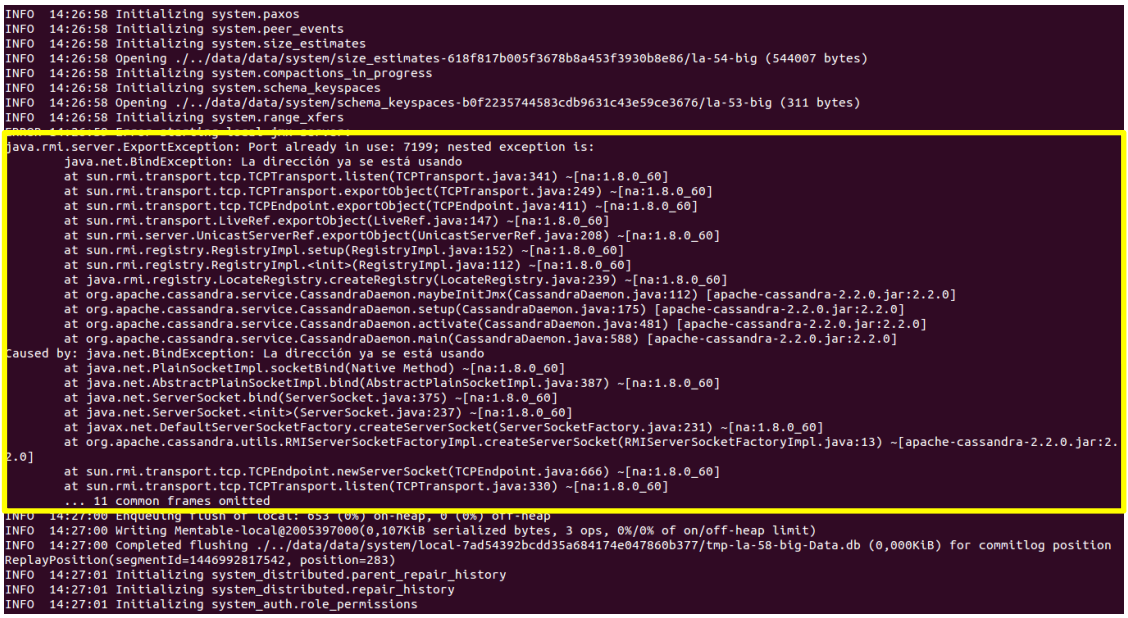


Figura 78: Error al lanzar Apache Cassandra

Observamos que el puerto "*7199*" ya se encuentra en uso y no se puede lanzar el servidor de Apache Cassandra. Este puerto es utilizado por Java, por lo que es necesario matar el proceso que utiliza ese puerto y lanzar nuevamente Apache Cassandra, como se explica en el anexo 1, con concreto en el apartado 2.5.

3.3. Error al insertar datos en Apache Cassandra

Durante el proceso de inserción de los datos sobre la base de datos, pueden existir una serie de problemas. Como podemos observar en la figura siguiente, da un error de "*rpc_timeout*".

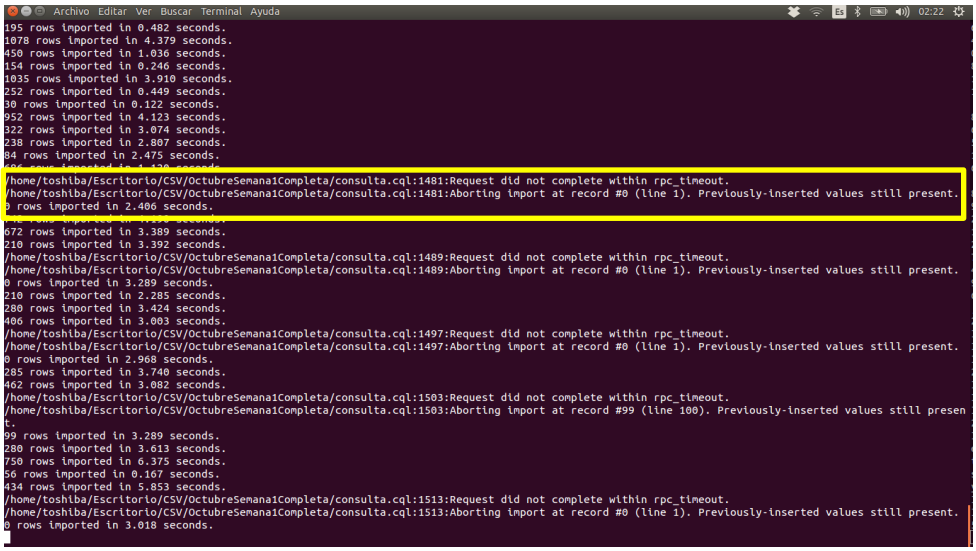


Figura 79: Errores del rpc_timeout

Este error (figura 79), se soluciona accediendo al archivo que se encuentra en la carpeta "*conf*" y que recibe el nombre de "*cassandra.yaml*". Este archivo se abre con un editor de texto. En concreto, tenemos que acceder a la zona relacionada con el "*timeout*" como vemos en la figura 80.

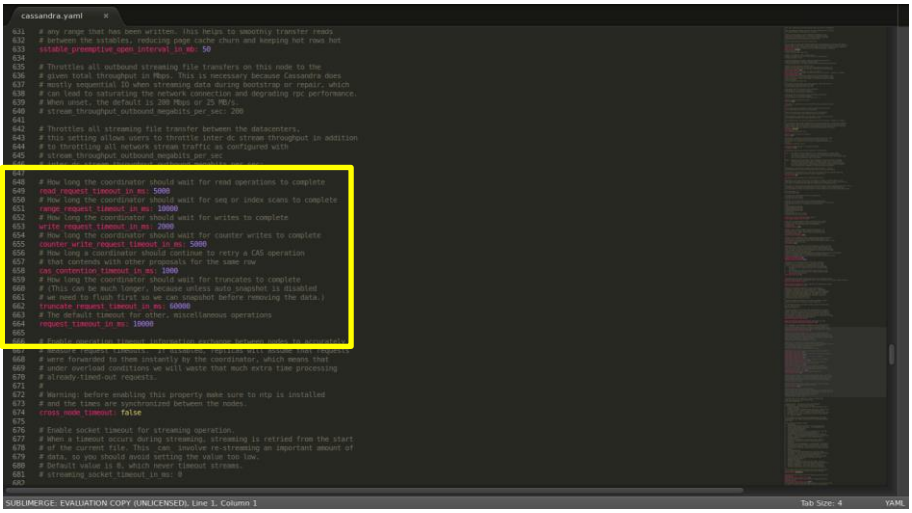


Figura 80: Modificar el timeout de Apache Cassandra

En este apartado podemos modificar el tiempo de espera a la hora de realizar escritura o lectura de datos sobre la base de datos. Este tiempo se encuentra en milisegundos.

Bibliografía y referencias

(AbreojosEnsamblador, 2015) Ilustración relacionada con la curva de Hilbert, http://www.abreojosensamblador.net/Productos/AOE/html/Imags/Cap32/Fig_32_25.png , visitado en diciembre de 2015.

(Academy Datastax, 2015a) Descripción de las características fundamentales de Apache Cassandra, https://academy.datastax.com/demos/brief-introduction-apache-cassandra , visitada en septiembre de 2015.

(Academy Datastax, 2015b) Driver para conectar Cassandra con el lenguaje de programación Python, https://academy.datastax.com/demos/getting-started-apache-cassandra-and-python-part-i , visitada en julio de 2015.

(Acens, 2015) Tipos de bases de datos NoSQL según su modelo, http://www.acens.com/wp-content/images/2014/02/bbdd-nosql-wp-acens.pdf , visitada en septiembre de 2015.

(Apache Cassandra, 2015) Apache Cassandra, base de dato NoSQL, http://cassandra.apache.org/ , visitada en enero 2015.

(Asto, P.C. y Apaza, Y. G., 2015) Como funciona un árbol R, http://es.slideshare.net/motita2894/arbol-r-r-tree , visitado en noviembre de 2015.

(CAMPUSMVP, 2015) Ilustración relacionada con el Teorema CAP, http://www.campusmvp.es/recursos/image.axd?picture=CAP.png , visitada en diciembre de 2015.

(Datastax, 2014) Herramienta para realizar consultas a la base de datos Apache Cassandra, http://www.datastax.com/what-we-offer/products-services/devcenter , visitada en enero de 2014.

(Dataversity, 2015) Descripción de las características BASE, http://www.dataversity.net/acid-vs-base-the-shifting-ph-of-database-transaction-processing/ , visitada en septiembre de 2015.

(Developers.Google, 2015a) Google Earth, Google Earth APIs Developers, https://developers.google.com/earth/ , visitada en febrero de 2015.

(Developers.Google, 2015b) Google Maps, Google Maps APIs Developers, https://developers.google.com/maps/ , visitada en Marzo de 2015.

(Doc.datastax, 2015) Cassandra Query Lenguaje, lenguaje para realizar consultas en Apache Cassandra, http://docs.datastax.com/en/cql/3.1/cql/cql_intro_c.html , visitado en enero de 2015.

(Eclipse.org, 2014) Entorno de desarrollo Eclipse, https://eclipse.org/ , visitada en septiembre de 2014.

(ECURED, 2015) Como generar la curva de Hilbert, http://www.ecured.cu/index.php/Curva_de_Hilbert , visitada en noviembre de 2015.

(Filezilla.org, 2014) Filezilla, FTP Client, https://filezilla-project.org/ , visitada en septiembre de 2014.

(Fowler, M., 2015) Video realizado por Martin Fowler que explica cómo surge las bases de datos NoSQL, https://www.youtube.com/watch?v=qI_g07C_Q5I , visitado en septiembre de 2015.

(GoogleUserContent, 2015) Ilustración relacionada con Apache Cassandra, https://lh5.googleusercontent.com/MVl5XV4LrXX1irtvBZ4bVIZ2EFSeKgYZff-YdOH9FpIkaS0_mlc-zTRmkYIABPiWNr9nnK5G82gcxPZQfS9MlsukVwKs73c0EG8eT_NMgR-DYgKXLJxKpjr2hjgd-Q8JpaQRBMI , visitado en diciembre de 2015.

(HDFGroup, 2015) Ficheros de tipo HDF, https://www.hdfgroup.org/products/hdf4/whatishdf.html , visitada en agosto de 2015.

(Hewitt, E., 2011) E. Hewitt, Titulo: Cassandra:The Definitive Guide, Año:2011, Publicado: Estados Unidos.

(LNDS, 2015) Explicación del teorema CAP, http://www.lnds.net/blog/2012/05/dos-de-tres.html , visitada en septiembre de 2015.

(Matesmates, 2015) Que es la curva de Hilbert y sus orígenes. https://matesmates.wordpress.com/2012/03/29/la-curva-de-hilbert/ , visitado en noviembre de 2015.

(Matplotlib, 2015) Graphic for Python, http://matplotlib.org/ , visitada en agosto de 2015.

(MDAD1415, 2015) Blogs realizado en MDAD en el curso escolar 2014/2015, https://gim.unex.es/mdad1415/a04/2015/03/08/introduction-to-nosql-martin-fowler/ , visitado en septiembre de 2015.

(Media.Licdn, 2015) Ilustración sobre la diferencia entre ACID y BASE, https://media.licdn.com/mpr/mpr/shrinknp_800_800/AAEAAQAAAAAAAAYCAAAAJDgzMDAyZjExLTFlODgtNDBjYy1iN2QyLWNmNDM4ZGEwNWRjOQ.png , visitado en diciembre de 2015.

(Numpy, 2015) Paquete para Python de datos científicos, http://www.numpy.org/ , visitada en marzo de 2015.

(ORACLE, 2014) Oracle Java, lenguaje de programación, https://www.oracle.com/java/index.html , visitada en diciembre de 2014.

(PlanetCassandra.org, 2015) Que es Apache Cassandra y cómo surge, http://www.planetcassandra.org/what-is-apache-cassandra/ , visitado en septiembre de 2015.

(PODAAC-TOOLS.JPL.NASA, 2015) Ilustración relacionada con el movimiento del satélite QuikSCAT, obtenida a partir de la consulta de los datos desde la página http://podaac-tools.jpl.nasa.gov/hitide/Subsetter.html?&datasets=QSCAT_LEVEL_2B_OWV_COMP_12&datasetsI=undefined&startdate=1125525600000&enddate=1125612000000&datasetStartdate=1999-10-27&datasetEnddate=2009-11-22 , visitada en diciembre de 2015.

(PODAAC.JPL.NASA, 2015a) Distributed Active Archive Center, ftp://podaac-ftp.jpl.nasa.gov/ , visitada en marzo de 2015.

(PODAAC.JPL.NASA, 2015b) NASA Scatterometer, https://podaac.jpl.nasa.gov/NSCAT , visitada en septiembre de 2015.

(PODAAC.JPL.NASA, 2015c) NASA QuikSCAT data "Ocean Winds QuikSCAT" https://podaac.jpl.nasa.gov/QuikSCAT , visitada en mayo de 2015.

(PODAAC.JPL.NASA, 2015d) OceanWind QuikSCAT, https://podaac.jpl.nasa.gov/OceanWind , visitada en septiembre de 2015.

(PODAAC.JPL.NASA, 2015e) NASA RapidSCAT data "Ocean Winds RapidSCAT" http://podaac.jpl.nasa.gov/dataset/RSCAT_LEVEL_2B_OWV_COMP_12_V1 , visitada en mayo de 2015.

(Pydev.org, 2015) Python IDE para Eclipse, http://www.pydev.org/ , visitada en febrero de 2015.

(Python.org, 2015) Python, lenguaje de programación, https://www.python.org/ , visitada en febrero de 2015.

(RIBW1314, 2015) Blogs realizado en RIBW en el curso escolar 2013/2014, https://gim.unex.es/ribw1314/a01/2014/04/01/video-introduction-to-nosql/ , visitado en septiembre de 2015.

(Saxena, U., 2015) Ilustración relacionada con el modelo de datos de Apache Cassandra, http://www.slideshare.net/UpaangSaxena/cassandra-45060830 , visitado en diciembre de 2015.

(Tomcat.Apache.org, 2015) Apache Tomcat, contenedor de Servlets, http://tomcat.apache.org/ , visitada en febrero de 2015.

(WINDS.JPL.NASA, 2015a) Ilustración relacionada con el movimiento del instrumento RapidSCAT, https://winds.jpl.nasa.gov/images/ISSFig2.jpg , visitada en diciembre de 2015.

(WINDS.JPL.NASA, 2015b) Ilustración relacionada con el instrumento RapidSCAT, https://winds.jpl.nasa.gov/images/RapidScat-image.jpg , visitada en diciembre de 2015.

(WINDS.JPL.NASA, 2015c) Misión y objetivos QuikSCAT, https://winds.jpl.nasa.gov/missions/quikscat/ , visitada en septiembre de 2015.

(WINDS.JPL.NASA, 2015d) Misión y objetivos RapidSCAT, https://winds.jpl.nasa.gov/missions/RapidScat/ , visitada en septiembre de 2015.

(Wikipedia, 2015a) ¿Qué es un árbol B?, https://es.wikipedia.org/wiki/Árbol-B , visitado en noviembre de 2015.

(Wikipedia, 2015b) ¿Qué es un árbol R? , https://en.wikipedia.org/wiki/R-tree , visitado en noviembre de 2015.

(Wikipedia, 2015c) Red Social Facebook, https://es.wikipedia.org/wiki/Facebook, visitada en septiembre de 2015.

(Wikipedia, 2015d) Wikipedia, Enciclopedia libre, Información sobre los huracanes en 2005, https://es.wikipedia.org/wiki/Temporada_de_huracanes_en_el_Atl%C3%A1ntico_de_2005 , visitada en julio de 2015.

(Wikipedia, 2015e) Cohete Titan II, https://es.wikipedia.org/wiki/Titan_(cohete), visitada en septiembre de 2015.

(Wikipedia, 2015f) Ilustración relacionada con el hurracan María generado en septiembre del 2005, https://en.wikipedia.org/wiki/File:Hurricane_Maria_September_6_2005.jpg , visitada en enero de 2016.

(W3SCHOOLS, 2015) JavaScript, Lenguaje de programación, http://www.w3schools.com/js/ , visitada en Marzo de 2015.

(Ubuntu.org, 2014) Sistema operativo Ubuntu, http://www.ubuntu.com/ , visitada en septiembre de 2014.

(UNIDATA.UCAR, 2015) Información sobre ficheros netCDF4, http://www.unidata.ucar.edu/software/netcdf/docs/ , visitado en marzo de 2015.