



UNIVERSIDAD DE EXTREMADURA

Escuela Politécnica

Grado en Ingeniería Informática en
Ingeniería del Software

Trabajo Fin de Grado

Almacenamiento NoSQL de
datos geoespaciales
con MongoDB y Geohashing

Francisco Javier Palacios Fernández

Septiembre, 2016



UNIVERSIDAD DE EXTREMADURA

Escuela Politécnica

Grado en Ingeniería Informática en
Ingeniería del Software

Trabajo Fin de Grado

Almacenamiento NoSQL de
datos geoespaciales
con MongoDB y Geohashing

Autor: Francisco Javier Palacios Fernández

Tutor: Félix Rodríguez Rodríguez

Tribunal Calificador

Presidente: Antonio Polo Márquez

Secretario: M^a Luisa Durán Martín-Merás

Vocal: Antonio J. Plaza Miguel

Calificación:

Agradecimientos:

Me gustaría agradecer a mi tutor Félix Rodríguez, que me ha ayudado a realizar este proyecto. Agradezco también a todo el profesorado de la universidad por los conocimientos adquiridos gracias a ellos.

Y en lo personal agradezco a mi familia por apoyarme en este camino.

Contenido

Índice de figuras.....	9
Índice de algoritmos.....	12
Resumen	13
1.- Introducción.	15
1.1.- Big Data.....	16
1.2.- Misión QuikSCAT y RapidSCAT.....	17
1.3.- NoSQL.....	20
1.3.1.- Teorema CAP.....	21
1.3.2.- Tipos de bases de datos NoSQL.....	22
1.3.3.- MongoDB.....	23
1.4.- Geohash.....	24
1.5.- Alcance y Objetivos del Proyecto.	27
2.- Materiales utilizados: Entorno, herramientas y librerías.	29
2.1.- Software.	29
2.1.1.- Windows 7.....	29
2.1.2.- Java.....	29
2.1.3.- Python.	30
2.1.4.- MongoDB.....	30
2.1.5.- Eclipse.....	30
2.1.6- Librerías utilizadas para la transformación de datos.....	31
2.2.- Hardware.....	31
3.- Desarrollo del proyecto.....	33
3.1.- Estudio de viabilidad del proyecto.	33
3.1.1.- Tareas realizadas.	34
3.2.- Análisis del sistema desarrollado.	38
3.2.1.- Extracción de los datos del repositorio.	40

3.2.2.- Estructura del proyecto.....	41
3.2.3.- Estructura de la base de datos.	45
3.2.4.- Transformación de los datos extraídos.....	46
3.2.5.- Índices utilizados.	50
3.2.6.- Carga de datos por índices.....	51
3.2.7.- Consulta de datos.....	59
3.2.8.- Índices GeohashMod y GeohashModTime.....	65
4.- Resultados.....	69
4.1.- Preparación de pruebas de rendimiento.....	69
4.2.- Datos QuikSCAT.	72
4.2.1.- Consultas temporales.	72
4.2.2.- Consultas espaciales.....	73
4.2.3.- Consultas espacio-temporales.	74
4.3.- Datos RapidSCAT.	75
4.3.1.- Consultas temporales.....	75
4.3.2.- Consultas espaciales.....	77
4.2.3.- Consultas espacio-temporales.	78
4.4.- Datos QuikSCAT y RapidSCAT.....	80
4.4.1.- Consultas temporales.....	80
4.4.2.- Consultas espaciales.....	81
4.4.3.- Consulta espacio-temporales.	82
5.- Conclusiones y Trabajo Futuro.	87
5.1.- Conclusiones según los resultados obtenidos en el proyecto.	87
5.2.- Conclusiones personales.	88
5.3.- Trabajos futuros sugeridos.	89
6.- Anexo 1: Manual del programador.....	91
6.1 Java.	91

6.2 HDF4.	94
6.3.- Python, Numpy y netCDF4.	96
6.3.1 Python y Numpy.	96
6.3.2.- netCDF4.	97
6.4.- MongoDB.	98
7.- Anexo 2: Manual de usuario.	99
7.1.- MongoDB.	99
7.2.- Aplicación Python.	101
7.3.- Aplicación java.	103
7.3.1.- Transformación.	103
7.3.2.- Cargar.	105
7.3.3.- Consultas.	106
Bibliografía.	109

Índice de figuras.

Figura 1: Resultado de las observaciones del QuikSCAT durante un periodo orbital (NASA/JPL, 2000).....	18
Figura 2: Teorema CAP y bases de datos que cumplen parte de las propiedades (Luis Miguel Gracia, 2013).....	22
Figura 3: Muestra de datos recogidas por RapidSCAT (Phil Whelan, 2011).	25
Figura 4: Muestra de la segunda división Geohash (Phil Whelan, 2011).	26
Figura 5: Diagrama Gantt tiempo estimado.	36
Figura 6: Diagrama Gantt tiempo utilizado.	37
Figura 7: Diagrama de acciones para del proyecto.	39
Figura 8: Muestra de datos QuikSCAT L2Bv2 (PODAAC, 2016d)	40
Figura 9: Muestra de datos RapidSCAT L2B12v1.2 (PODAAC, 2016f)....	41
Figura 10: Diagrama de clases de diseño.	44
Figura 11: Esquema de diseño de la base de datos.....	46
Figura 12: Comparación de los tiempos de respuesta de las consultas temporales de los datos QuikSCAT almacenados en MongoDB utilizando los índices Simple, Compuesto y 2D.	72
Figura 13: Comparación de los tiempos de respuesta de las consultas temporales de los datos QuikSCAT almacenados en MongoDB utilizando los índices Simple, Compuesto, 2D y GeohashMod, y con una reducción de los intervalos de consulta en 10°.	73
Figura 14: Comparación de los tiempos de respuesta de las consultas espaciales de los datos QuikSCAT almacenados en MongoDB para todos los tipos de indexación seleccionados.....	74
Figura 15: Comparación de los tiempos de respuesta de las consultas espacio-temporales de los datos QuikSCAT almacenados en MongoDB para todos los tipos de indexación seleccionados.	75

Figura 16: Comparación de los tiempos de respuesta de las consultas temporales de los datos RapidSCAT almacenados en MongoDB utilizando los índices Simple, Compuesto y 2D.	76
Figura 17: Comparación de los tiempos de respuesta de las consultas temporales de los datos RapidSCAT almacenados en MongoDB utilizando los índices Simple, Compuesto, 2D y GeohashMod, y con una reducción de los intervalos de consulta en 10°.	77
Figura 18: Comparación de los tiempos de respuesta de las consultas espaciales de los datos RapidSCAT almacenados en MongoDB para todos los tipos de indexación seleccionados.	78
Figura 19: Comparación de los tiempos de respuesta de las consultas espacio-temporales de los datos RapidSCAT almacenados en MongoDB para todos los tipos de indexación seleccionados.	79
Figura 20: Comparación de los tiempos de respuesta de las consultas espacio-temporales de los datos RapidSCAT almacenados en MongoDB para todos los tipos de indexación seleccionados., en detalle.	79
Figura 21: Comparación de los tiempos de respuesta de las consultas temporales de los datos QuikSCAT y RapidSCAT almacenados en MongoDB utilizando los índices Simple, Compuesto y 2D.	80
Figura 22: Comparación de los tiempos de respuesta de las consultas espaciales de los datos QuikSCAT y RapidSCAT almacenados en MongoDB utilizando los índices Simple, Compuesto y 2D.	81
Figura 23: Comparación de los tiempos de respuesta de las consultas espaciales de los datos QuikSCAT y RapidSCAT almacenados en MongoDB utilizando los índices GeohashMod y GeohashModTime.	82
Figura 24: Comparación de los tiempos de respuesta de las consultas espacio-temporales de los datos QuikSCAT y RapidSCAT almacenados en MongoDB utilizando los índices Simple, Compuesto y 2D.	83
Figura 25: Comparación de los tiempos de respuesta de las consultas espacio-temporales de los datos QuikSCAT y RapidSCAT almacenados en MongoDB utilizando los índices Simple, Compuesto y 2D, en detalle.	84

Figura 26: Comparación de los tiempos de respuesta de las consultas espacio-temporales de los datos QuikSCAT y RapidSCAT almacenados en MongoDB utilizando los índices GeohashMod y GeohashModTime. ...	85
Figura 27: Web de descarga de Java.	91
Figura 28: Programa de instalación de java, inicio.	92
Figura 29: Programa de instalación de java, lugar de instalación.	93
Figura 30: Programa de instalación de java, publicidad.	94
Figura 31: Web de descarga HDF4.	95
Figura 32: Descarga de ejecutable de HDF4.	95
Figura 33: Descarga programa Anaconda.	96
Figura 34: Descarga de netCDF4.	97
Figura 35: Instalación de MongoDB.	98
Figura 36: Ejecutar consola de Windows 7.	99
Figura 37: Ejecución del servidor de MongoDB.	100
Figura 38: Servidor MongoDB esperando por conexiones.	101
Figura 39: Ejecución de programa Python.	101
Figura 40: Ejecución de programa Python, selección 1.	102
Figura 41: Pantalla de aplicación Java, Transformación.	104
Figura 42: Pantalla de aplicación Java, Cargar.	106
Figura 43: Pantalla de aplicación Java, Consultas.	107

Índice de algoritmos.

<u>Algoritmo 1: Transformación de los datos desde formato científico a CSV.</u>	49
<u>Algoritmo 2: Carga a la base de datos MongoDB.</u>	55
<u>Algoritmo 3: Creación de colección en la base de datos.</u>	57
<u>Algoritmo 4: Consulta de datos a la base de datos MongoDB.</u>	64
<u>Algoritmo 5: Cálculo de hash a partir de un punto en el espacio.</u>	66
<u>Algoritmo 6: Cálculo de bit según punto para índice GeohashMod.</u>	67
<u>Algoritmo 7: Codificación base32.</u>	67
<u>Algoritmo 8: Codificación base-32 del día.</u>	68

Resumen

El objetivo de este trabajo de final de grado es comparar la eficiencia de distintos tipos de índice en la base de datos NoSQL MongoDB para el caso de datos geoespaciales. En especial se utilizará el innovador tipo de codificación Geohash, para el cual crearemos lo necesario para su codificación, decodificación y consulta una vez lo hayamos cargado a la base de datos.

Se utilizan los datos de las misiones QuikSCAT y RapidSCAT de la NASA descargados desde su repositorio, los cuales almacenan datos meteorológicos oceánicos y marinos. Para el presente proyecto, estos datos se transforman de su formato inicial a un formato más simplificado. Posteriormente, y dado que el proyecto propone almacenarlos en una base de datos NoSQL MongoDB que permita su integración homogénea y rápido acceso, se realiza de nuevo una transformación dependiendo de las necesidades de cada tipo de índice implementado por MongoDB.

El proyecto estudia los índices más adecuados para almacenar datos geoespaciales implementados por MongoDB y se comparan. También se tiene en cuenta en la comparativa el índice Geohash que se ha construido específicamente en este trabajo. Los resultados del estudio realizado miden la eficiencia en el tiempo de respuesta a las consultas espaciales, temporales y espacio-temporales, siempre con los mismos datos almacenados en todas y cada una de las pruebas realizadas.

1.- Introducción.

Actualmente nos encontramos en la era de la información, donde todo está conectado a la red. Debido a esto es posible conseguir muchos datos de diversos tipos, como tendencias actuales, gustos, intereses.

Basándose en esta idea, se han creado conceptos como NoSQL o Big Data. Sin embargo, es necesario un paso más, todos estos datos no son más que una recopilación que pueden contener información.

Para obtener conocimiento más allá del almacenado de esta información desde un gran conjunto de datos se ha desarrollado el Data Mining, el cual intenta descubrir patrones en grandes volúmenes de datos, que después permitirá desarrollar estrategias de marketing, seguridad aérea y meteorológica o incluso estrategias financieras en la bolsa. Cuando el volumen de información sobre el que extraer conocimiento es enorme, proveniente de diversas fuentes, y el tiempo de respuesta requerido para poder extraer conclusiones sobre los datos debe ser mínimo, nos adentramos en lo que hoy en día conocemos como Big Data. Para el caso de los datos geoespaciales, dado su enorme valor informativo y la ingente cantidad de estos datos que se generan, el tiempo de respuesta necesario para realizar las consultas de las tareas de minado es vital. La manera tradicional de acelerar los tiempos de respuesta a las consultas es mediante la utilización de estructuras de datos específicas, los índices o de métodos algorítmicos basados en funciones de acceso a datos como es el hashing. El estudio de la indexación y del hashing para datos geolocalizados es el cometido del presente proyecto. Y de una manera mucho más específica nos centraremos en el método de Geohashing, una de las propuestas más eficientes que actualmente se utilizan.

Todo esto lo iremos estudiando más en detalle en los sucesivos apartados de este documento haciendo hincapié sobre todo, en el almacenado de datos y consultas y las herramientas escogidas para llevar a cabo este proyecto. Antes de plantear los objetivos del presente proyecto y los desarrollos propuestos, en este capítulo introductorio trataremos conceptos,

tipos de datos utilizados y las tecnologías de las que su conocimiento es básico para entender el alcance del trabajo realizado. Por ello, en el apartado 1.1 describiremos qué se entiende por Big Data, en el apartado 1.2 describiremos la naturaleza de los datos con los que se ha trabajado en el presente proyecto. Posteriormente, en el apartado 1.3 describimos qué se entiende por almacenamiento NoSQL y toda la tecnología que subyace bajo este tipo de bases de datos. Y, antes de abordar en el apartado 1.5 el alcance y objetivos del presente proyecto, en el apartado 1.4 describiremos la idea que subyace bajo la técnica del geohashing.

1.1.- Big Data.

Big Data (MongoDB, 2016a) es el primer término que nos atañe; sin él no sería necesario el buscar nuevas formas de almacenar datos, ni el filtrado y estudio de estos, tal y como lo hacen los conceptos de los que posteriormente comentaremos, como son el almacenamiento NoSQL y las técnicas de extracción de conocimiento o Data Mining.

El término Big Data se centra sobre todo en la gran cantidad de datos que actualmente generamos todos y que superan la capacidad actual del software convencional. Las principales diferencias entre el procesado habitual de datos y el Big Data, se resume básicamente en 3 conceptos: volumen, variedad y velocidad (MongoDB, 2016a).

Cuando estamos hablando de Big Data, no estamos hablando de la cantidad o volumen de datos que puede llegar a manejar, por ejemplo, una pequeña o mediana empresa, que quizá almacene varios Gigabytes con datos sobre sus productos o clientes. Cuando hablamos del volumen de Big Data, estamos tratando con Terabytes o Petabytes.

Además, estos datos proceden de distintas fuentes provenientes de objetivos distintos. Por ejemplo, podemos estar hablando de información obtenida de distintas redes sociales para la personalización o el almacenamiento de datos científicos para estudios biométricos. Este es el concepto de variedad al que nos referíamos antes.

Por último, este gran volumen de datos nos lleva a un problema: si queremos procesarlos todos para obtener información útil y valiosa, vamos a necesitar, o bien una gran cantidad de tiempo para poder analizar estos datos, o bien crear herramientas, conceptos, algoritmos, etc., para aumentar la velocidad de este procesamiento de datos.

Los datos geoespaciales con los que tratamos en el presente proyecto se involucran completamente en estos 3 conceptos: su volumen es enorme, su variedad también (se pueden obtener de muchos satélites orbitales, aeronaves de reconocimiento o de digitalizaciones históricas), y el tiempo de respuesta debe ser mínimo para poder dar respuesta eficazmente a las posibles tareas de análisis y minado de los propios datos registrados.

1.2.- Misión QuikSCAT y RapidSCAT.

Para la realización de este proyecto, era necesario conseguir datos que satisficieran el hecho de estar disponibles y que fuesen una gran cantidad. Estos datos, han sido obtenidos de las misiones QuikSCAT y RapidSCAT.

La misión QuikSCAT consistía en rellenar el hueco que había dejado la pérdida de datos del NASA Scatterometer (NSCAT) el 30 de junio de 1997, debido a un fallo de alimentación (PODAAC, 2016a). Insertando en el satélite QuikSCAT el instrumento SeaWinds, pretendía seguir con ese trabajo mediante un radar microondas específicamente diseñado, que medía la velocidad del viento cerca de la superficie y la dirección del viento en todas las condiciones meteorológicas.

El satélite fue lanzado ascendiendo 800 kilómetros sobre la superficie de la Tierra formando una órbita elíptica alrededor de la misma. Utilizando su antena, que irradiaba pulsos microondas cada 13,4 GHz, el instrumento SeaWinds recogía datos sobre los océanos, mares y lagos en una banda continua de 1.800 kilómetros de diámetro, haciendo aproximadamente 400.000 mediciones y cubriendo el 90% de la superficie de la Tierra al día.

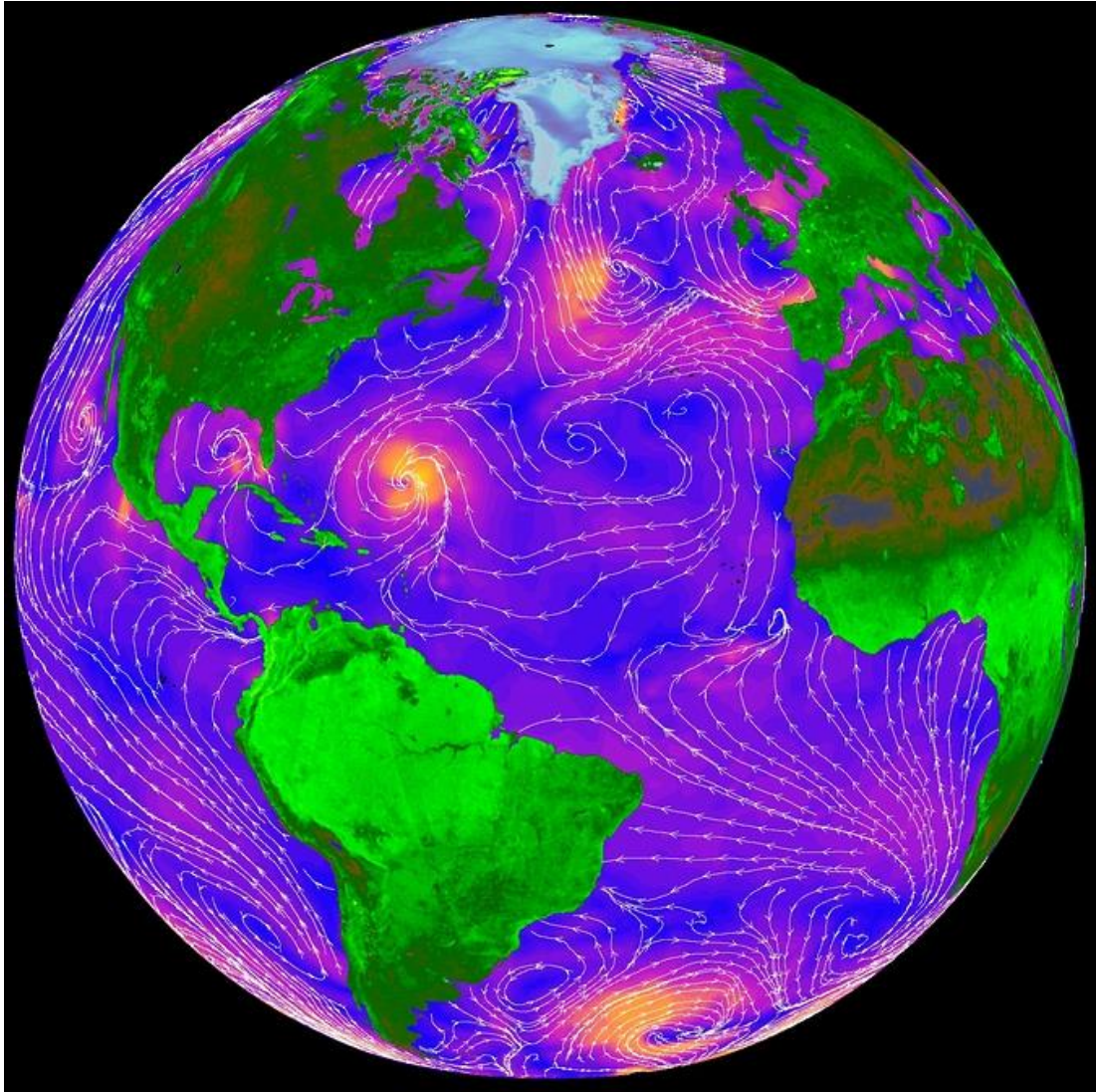


Figura 1: Resultado de las observaciones del QuikSCAT durante un periodo orbital (NASA/JPL, 2000).

En la figura 1 se visualiza de una forma sencilla, parte de las observaciones tomadas por el satélite QuikSCAT dando colores a los distintos valores. Esta toma es del 20 de septiembre de 1999 durante el huracán Gert, que podemos ver con esos colores amarillentos cerca de la costa de Florida.

Los datos de QuikSCAT se encuentran almacenados en un repositorio FTP gestionado por la NASA (PODAAC, 2016c). Los datos que trataremos se encuentran en el formato científico HDF (The HDF Group, 2011).

Algunos de los objetivos científicos de la misión eran (NASA/JPL, 2016a):

- Combinar datos de viento con las mediciones de los instrumentos científicos en otras disciplinas para ayudarnos a entender mejor los patrones de cambios climáticos globales y meteorológicos.
- Estudiar los cambios anuales y semestrales de la vegetación de la selva tropical.
- Estudiar los movimientos y cambios diarios y estacionales de los glaciares tanto del Ártico como del Antártico.
- Determinar los mecanismos forzados atmosféricos, la respuesta del océano y la interacción aire-mar en varias escalas espaciales y temporales.

La misión RapidSCAT (California Institute of Technology, 2016) reemplaza al satélite QuikSCAT por un fallo relacionado con la antigüedad del mismo; sin embargo, en lugar de albergarse en un satélite orbital, el ingenio se encuentra a bordo de la Estación Espacial Internacional (ISS), realizando las mismas acciones que QuikSCAT, de manera más precisas (mayor resolución de captura) pero cubriendo un área más reducida que la utilizada por QuikSCAT porque las órbitas de la ISS son diferentes: con una banda continua de 900 kilómetros y cubriendo la mayoría del océano entre 51,6 grados de latitud Norte y Sur.

Los datos recogidos en la misión RapidSCAT se encuentran también almacenados en un repositorio FTP gestionado por la NASA. El formato de estos datos es distinto, llamado netCDF (Unidata, 2016); una evolución del anterior formato HDF.

Algunos de los objetivos científicos de la misión RapidSCAT son (NASA/JPL, 2016b):

- Proporcionar una plataforma de calibración cruzada para mejorar la OVW (Ocean Vector Winds), lo que aumentará el valor de los datos y la estabilidad del QuikSCAT.

- Realizar estudios sobre el ciclo diurno y semi-diurno de los vientos sobre el océano y la evapotranspiración de la Tierra.
- Proporcionar datos útiles para la predicción de tormentas marinas y datos para completar el muestreo espacio-tiempo de la constelación OVW internacional.

1.3.- NoSQL.

Las siglas NoSQL (MongoDB, 2016b) se corresponde con *Not only Structured Query Language*, haciendo referencia a una gran cantidad de gestores de bases de datos que son distintos del sistema clásico de gestión de bases de datos relacionales o RDBMS (*Relational Data Base Management Systems*). Algunas de las diferencias de las bases de datos NoSQL respecto a las tradicionales bases de datos relacionales son (Wikipedia, 2016):

- No usan SQL como el principal sistema para realizar consultas a la base de datos.
- Los datos almacenados no requieren estructuras fijas como tablas (mostraremos más en profundidad los tipos de estructuras utilizados cuando veamos los tipos de bases de datos NoSQL).
- No soportan, normalmente, las operaciones JOIN.
- No garantizan completamente las características ACID (Atomicidad, Consistencia, Aislamiento y Durabilidad).
- Pueden escalar horizontalmente. Es decir, es posible aumentar el rendimiento agregando más nodos al sistema.

El termino NoSQL fue acuñado por Carlo Strozzi en 1998 para referirse a su propia base de datos, la cual seguía el modelo relacional pero no ofrecía una interface NoSQL. El término fue resucitado en 2009 por Eric Evans durante un evento para discutir bases de datos distribuidas de código abierto (Wikipedia, 2016).

Este tipo de bases de datos crecieron debido a problemas con el tratamiento de grandes cantidades de datos que las RDBMS (Sistemas gestores de

bases de datos relacionales) no solucionaban. Grandes empresas como Google, Amazon, Twitter o Facebook llegaron a la conclusión de que el rendimiento y sus propiedades en tiempo real, eran más importantes que la coherencia, haciendo ser, en estos casos, peores las bases de datos relacionales ya que dedicaban una gran cantidad de tiempo en procesos.

1.3.1.- Teorema CAP.

Eric Brewer, en el año 2000 durante un simposio, enunció este teorema haciendo hincapié en hecho de que era necesario empezar a dejar de preocuparse por la coherencia de los datos, porque si se quería una alta disponibilidad en estas aplicaciones distribuidas, entonces la consistencia era algo que no se podía garantizar (Julian Browne, 2009).

De esta forma, es posible elegir una base de datos que satisfaga 2 de las siguientes características, pero nunca las 3:

- Consistency (Consistencia): los datos guardados en nuestro almacén de datos deben ser correctos.
- Availability (Disponibilidad): es posible consultar estos datos almacenados por cualquier usuario en cualquier momento.
- Partition Tolerance (Tolerancia a fallos / a la partición): las peticiones seguirán funcionando sobre el almacén aunque un nodo falle, ya que otro dará la respuesta.

En la imagen de la figura 2 se muestran las distintas bases de datos más relevantes ubicadas según las características que cumplen del teorema CAP: como solo pueden asegurar dos de las tres propiedades enumeradas, las bases de datos se colocan en el lado del triángulo cuyos dos vértices de características satisfacen. La figura también muestra una breve descripción de los modelos de datos utilizados por estas bases de datos. Como vemos, MongoDB se encuentra en la parte inferior con un modelo de datos orientado a documentos y garantizando las características de consistencia y tolerancia a fallos.

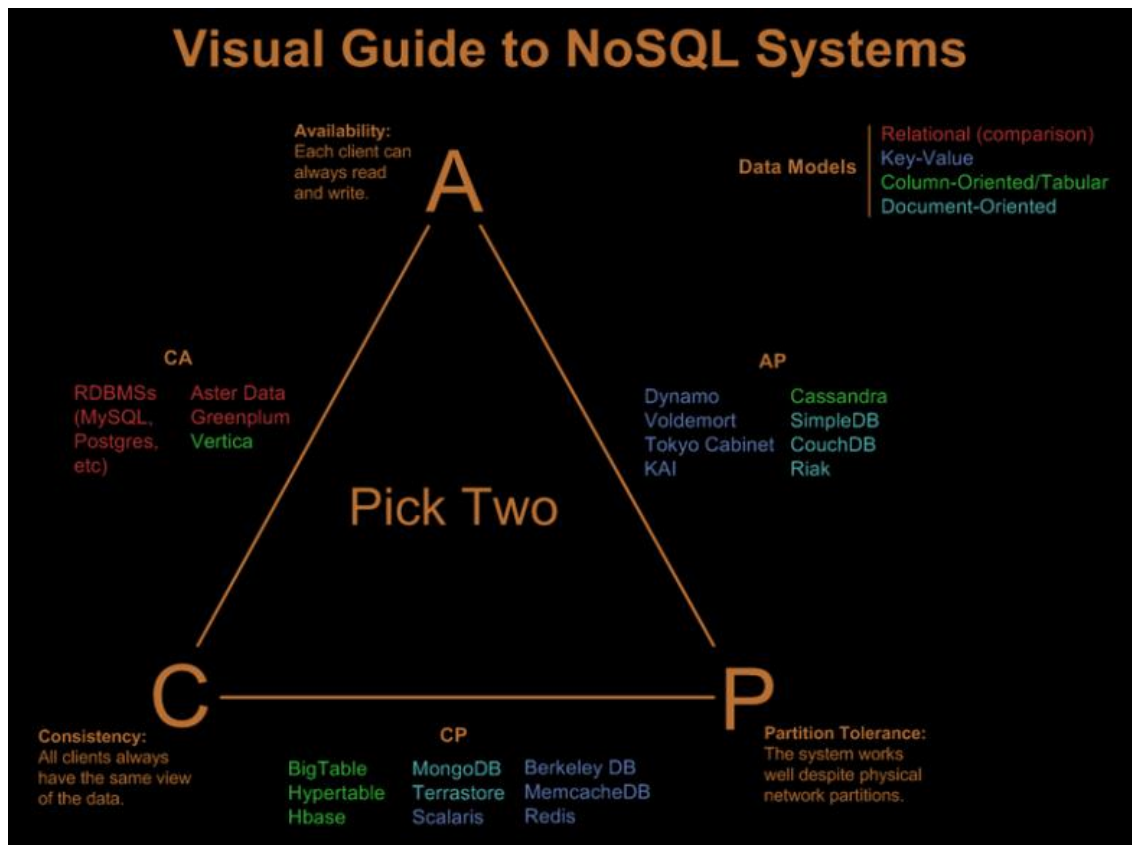


Figura 2: Teorema CAP y bases de datos que cumplen parte de las propiedades (Luis Miguel Gracia, 2013).

1.3.2.- Tipos de bases de datos NoSQL.

Podemos hacer una distinción de las bases de datos NoSQL en base al Teorema CAP y a como almacenan internamente los datos cada gestor de base de datos (Pramod Sadalage, 2014):

- Clave-valor: Cada valor almacenado es identificado por una clave. Son las más fáciles de usar desde el punto de vista de la API.
- Orientada a documentos: Se almacenan documentos con una estructura jerárquica en árbol, autodescriptivos ya que cada documento almacena sus propios pares clave-valor. Proporcionan un sistema de consulta poderoso y construcciones tales como índices.
- Orientada a columnas: Los datos son almacenados en familias de columnas como filas que tienen muchas columnas asociadas como clave de fila. Estas familias de columnas, son datos que a menudo se

consultan juntos. Este esquema es bueno a la hora de gestionar las cargas masivas de datos.

- Orientada a grafos: Su modelo de datos está compuesta por nodos que están conectados. Son buenas para modelar un dominio con forma de grafo ofreciendo un gran rendimiento cuando los datos están interconectados.

1.3.3.- MongoDB.

MongoDB (MongoDB, 2016c) es un sistema gestor de base de datos NoSQL orientado a documentos y desarrollado bajo el concepto de código abierto. MongoDB almacena estructuras de datos en documentos en un formato similar a JSON con un esquema dinámico, llamado BSON. Cumple con las características de consistencia y tolerancia a la partición del teorema CAP.

Algunas de sus características son (Wikipedia 2016b):

- Consultas Ad Hoc: soporta la búsqueda por campos, consultas de rangos y expresiones regulares. Las consultas pueden devolver un campo específico del documento, pero también puede ser una función JavaScript definida por el usuario.
- Indexación: cualquier campo puede ser indexado, y también es posible utilizar índices secundarios. El concepto de indexación es similar al manejo en las bases de datos relacionales.
- Replicación: soporta replicación maestro-esclavo. El maestro tiene la capacidad de ejecutar comandos de escritura y lectura, mientras que el esclavo solo copia lo almacenado en el maestro, puede leer o realizar copias de seguridad. A cada conjunto maestro-esclavos se le llama réplica set. Es posible que un esclavo cambie de maestro en caso de que este deje de responder.
- Equilibrio de carga: es posible distribuir la carga de almacenamiento entre los distintos servidores que formen la base de datos.
- Agregación: se pueden realizar operaciones similares a las que se obtienen con el comando SQL *"group by"*.

- Ejecución de JavaScript del lado del servidor: es posible realizar consultas utilizando JavaScript.

La elección de MongoDB como sistema NoSQL de almacenamiento para los datos extraídos del satélite QuikSCAT y de RapidSCAT se debe, primordialmente, a su facilidad de uso, a la eficiencia con la que opera a la hora de realizar consultas, y a la capacidad y variedad de tipos de índices que nos proporciona.

1.4.- Geohash.

Geohash (Phil Whelan, 2011) es un tipo de codificación geográfica, creada por Gustavo Niemeyer, que subdivide la superficie terrestre en formas rectangulares a las que se les asigna un código. La precisión de estas formas rectangulares depende de la cantidad de bits utilizados para la codificación, de forma que a mayor cantidad de bits utilizados, mayor será su precisión.

Para comprender el funcionamiento de este tipo de codificación, utilizaremos la Figura 3 para ayudarnos. El funcionamiento de la técnica es el siguiente: dado un punto cualquiera de la Tierra (identificado en la Figura 3 con un punto rojo), se divide todo el plano terrestre por su mitad (se suele tomar la mitad del eje de longitudes como primera división) y se observa donde se encuentra dicho punto. Si el punto se encuentra a la derecha de la división, se elige esa división y el primer bit será un 1. En caso contrario, se elegirá la división izquierda y el primer bit será un 0.



Figura 3: Muestra de datos recogidas por RapidSCAT (Phil Whelan, 2011).

Posteriormente, se vuelve a partir por la mitad el subplano terrestre donde se encuentra el punto en cuestión (esta vez en latitud). La Figura 4 muestra el subplano terrestre nuevamente dividido. Si el punto se encuentra en la parte superior, se sufixa al bit anterior un 0; en caso contrario se sufixa (concatena a la derecha) un 1.



Figura 4: Muestra de la segunda división Geohash (Phil Whelan, 2011).

A continuación, se vuelve a realizar el primer paso, dividiendo por la mitad en longitud el subplano donde se encuentra el punto e igualmente se concatenará un 1 a la cadena de bits que se va formando de las sucesivas subdivisiones en caso de elegir la división derecha (001 en nuestro caso), o un 0 en el caso contrario.

Todo este proceso se repite continuamente alternando los subplanos de división terrestre hasta llegar a la precisión establecida. Una vez obtenida la cadena, ésta se suele codificar utilizando una codificación de 32 bits.

Gracias a esta forma de codificación es posible describir y encontrar pequeñas zonas de la superficie terrestre de una forma rápida y sencilla. La sencillez del concepto y de su implementación, unido a las respuestas de coste constante $O(1)$ en las consultas nos ha llevado a experimentar con

esta solución en el presente proyecto. Mostraremos cuán eficaz es este tipo de codificación comparándolo con diversos métodos de indexación ofrecidos por MongoDB para los datos geolocalizados.

Cabe destacar que la codificación Geohash utilizada en este proyecto se ha decidido llamarla GeohashMod ya que su implementación se ha realizado íntegra y específicamente para este proyecto.

1.5.- Alcance y Objetivos del Proyecto.

Una vez descritas las técnicas básicas necesarias y la naturaleza de los datos geolocalizados utilizados en el presente proyecto, podemos pasar a enumerar de manera concisa los objetivos y el alcance del presente Trabajo Final de Grado.

De manera resumida son tres:

1. Conseguir extraer y efectuar las transformaciones necesarias para adecuar el formato actual de los datos geoespaciales QuikSCAT y RapidSCAT disponibles en los repositorios de la NASA.
2. Transformar los datos anteriores para adecuarlos a algún tipo de índice específico de la base de datos NoSQL MongoDB de forma que sea posible cargar y consultar los datos desde una interfaz similar para todos los tipos de índice. Además, se planea implementar codificación por Geohash para crear un índice basado en esta técnica.
3. Realizar un estudio de la eficiencia y la eficacia para las consultas de los diversos modos de indexación proporcionados por MongoDB para datos geolocalizados. Asimismo, el estudio realizado comparará los distintos modos de indexación con la técnica de Geohashing.

2.- Materiales utilizados: Entorno, herramientas y librerías.

En este capítulo se documentarán las herramientas hardware y software utilizados para el desarrollo del proyecto.

2.1.- Software.

A continuación, se detalla el software utilizado en el proceso de desarrollo del proyecto, así como las distintas librerías utilizadas.

2.1.1.- Windows 7.

El sistema operativo utilizado ha sido Windows 7, en su versión *Ultimate Service Pack 1* (Microsoft, 2016). Sobre este sistema operativo se han descargado los archivos que contienen los datos geoespaciales del repositorio de la NASA y se ha realizado la transformación de dichos archivos desde su formato científico en HDF-4 o netCDF4, dependiendo de si los datos son QuikSCAT o RapidSCAT respectivamente, a un formato unificado y sencillo como son los archivos de valores separados por comas o CSV (*Comma-Separated Values*).

Sobre este sistema operativo se ha instalado tanto el sistema gestor de base de datos, como las distintas librerías para la utilización de los distintos lenguajes de programación para el desarrollo de las aplicaciones realizadas.

2.1.2.- Java.

Se ha utilizado la versión 1.8 de la máquina virtual del lenguaje de programación Java (Oracle, 2016a) para el desarrollo de la aplicación que interactúa con el sistema gestor de base de datos MongoDB. Se han utilizado distintas APIs (*Application Programming Interfaces*) para el desarrollo de la aplicación, las cuales son:

- Mongo Java Driver: se ha utilizado la versión 3.0.2. Esta API ha sido utilizada para la comunicación entre el programa desarrollado en Java y el sistema gestor de base de datos MongoDB. Se puede descargar desde su página web de forma gratuita:

<https://docs.mongodb.com/ecosystem/drivers/java/> (MongoDB, 2008-2016a).

- Jcalendar: se ha utilizado la versión 1.4. Esta API ha sido empleada para la inserción de fechas a la hora de realizar consultas desde la aplicación desarrollada. Se puede descargar gratuitamente desde su página web <http://toedter.com/jcalendar/> (Kai Tödter, 2016).

2.1.3.- Python.

Se ha utilizado la versión 2.7 de este lenguaje de programación para la transformación de datos en formato científico netCDF4 de los archivos de RapidSCAT a CSV.

Para instalar el intérprete del lenguaje podemos hacerlo desde su página web <http://toedter.com/jcalendar/> (Python Software Foundation, 2001-2016).

2.1.4.- MongoDB.

La versión utilizada ha sido la 3.2.7 de este gestor de base de datos no relacionales, con el que se han almacenado los datos QuikSCAT y RapidSCAT extraídos del repositorio de la NASA. La razón de utilizar este gestor de base de datos, ha sido porque resultaba una nueva experiencia realizar las distintas operaciones sobre esta base de datos, disponer de la capacidad de tener índices geoespaciales ya que es la característica principal de los datos extraídos del repositorio y la rapidez de resultado de las consultas, siempre que se utilicen adecuadamente los índices.

Se puede descargar desde la página web de la organización <https://www.mongodb.com/download-center#community> (MongoDB, 2016d).

2.1.5.- Eclipse.

Ha sido utilizada la versión Mars 1 de este entorno de desarrollo integrado o IDE (*Integrated Development Environment*), que contiene herramientas de programación que han sido muy útiles para el desarrollo de la programación en Java. Además, se ha utilizado el plugin WindowBuilder para el desarrollo de la interfaz gráfica de la aplicación. Ambos pueden descargarse desde el

sitio web de eclipse <https://eclipse.org/downloads/packages/release/Mars/1> (The Eclipse Foundation, 2016).

2.1.6- Librerías utilizadas para la transformación de datos.

Es necesaria la instalación de algunas librerías para la transformación de los datos al formato CSV:

- Para la transformación de datos QuikSCAT es necesaria la instalación de las librerías HDF4 que permiten la lectura y transformación de los datos desde su formato científico a CSV. Esta transformación se realiza desde una aplicación Java desarrollada. La librería se puede obtener desde su sitio web <https://www.hdfgroup.org/release4/obtain.html> (The HDF Group, 2016).
- Para la transformación de datos RapidSCAT es necesaria la instalación de las librerías netCDF-4 para Python, las cuales permiten la lectura y tratamiento de datos NetCDF. Se pueden obtener desde el sitio web <https://pypi.python.org/pypi/netCDF4/1.0.7> (Python Software Foundation, 1990-2016). Se necesita además la instalación del paquete de librerías Numpy (Numpy, 2016) para la computación científica con Python y el tratamiento de datos realizado con NetCDF. Este paquete puede obtenerse desde el sitio web <http://www.scipy.org/scipylib/download.html> (SciPy developers, 2016).

2.2.- Hardware.

El desarrollo del proyecto, así como las pruebas con los distintos índices, ha sido realizados en un ordenador con un procesador i5-4210h a 2.9 GHz, con 8 GB de memoria RAM y un disco duro de 500 GB.

Ha de comentarse que este entorno no es el perfecto para este tipo de procesamiento a gran escala, debido al alto volumen de datos que es necesario procesar y guardar. Sin embargo, y dado que uno de los objetivos primordiales del proyecto es el de comparar las diversas técnicas de acceso

eficiente a MongoDB, con utilizar una muestra significativa de datos para el estudio comparativo, se puede alcanzar una idea cercana al comportamiento real con el enorme volumen de información que supondría la carga completa de los datos meteorológicos marinos de QuikSCAT y RapidSCAT.

3.- Desarrollo del proyecto.

3.1.- Estudio de viabilidad del proyecto.

Como es necesario utilizar distintos tipos de software, la viabilidad del proyecto depende de varios factores. El proceso resumido fue el siguiente: descargar los datos de las fuentes ya comentadas, la transformación de ellos a un formato más simple, su carga en una base de datos NoSQL con una serie de distintos índices y finalmente, una serie de consultas de prueba para apuntar tiempos de respuesta y analizarlos.

Para esta empresa fue necesaria la descarga de los datos desde el repositorio de datos de la NASA mediante el programa de gestión de descargas Filezilla, ya que facilita la bajada de datos desde FTP.

Posteriormente se transformaron los datos desde su formato inicial a un formato más simple como es CSV, para ello se utilizó el programa QSreadx64 (Ruiz Romero, Fco, 2012), el cual se encapsuló, para el caso de los datos provenientes de la misión QuikSCAT, en un programa Java desarrollado para facilitar su uso y para los provenientes de la misión RapidSCAT, en un programa en Python 2.7 desarrollado por Daniel Teomiro (Teomiro Villa, Daniel, 2015), al que fue necesario realizar una serie de cambios para adecuarlo a una salida común con la de los datos QuikSCAT.

Es necesario instalar unos paquetes de librerías para ambos programas: HDF4 para QSreadx64 y netCDF4, además de Numpy para el programa en Python, cuya instalación puede ser un problema porque necesita que se instalen otros paquetes secundarios para su correcta instalación.

La descarga e instalación de MongoDB no es muy complicada, ya que proporciona un sencillo asistente de configuración. Sin embargo, sí que fue necesario efectuar un estudio sobre las distintas herramientas de MongoDB para poder realizar las cargas y las consultas sobre esta base de datos.

El programa de carga también debe ejecutar una transformación de los datos para adecuarlos al índice que se requiera. Además, la consulta

necesitará realizar su transformación de nuevo a un formato común para que el análisis de tiempos cobre sentido.

3.1.1.- Tareas realizadas.

En este apartado se hará una descripción de las tareas realizadas y su tiempo estimado. Presentamos dos tablas y dos figuras. La primera y segunda tabla representan, respectivamente, el tiempo estimado de desarrollo del proyecto y el tiempo real que se ha empleado en realizarlo. Las dos figuras siguientes muestran esta información a través de diagramas de Gantt.

Tiempo estimado		
Área	Tareas	Tiempo (días)
Análisis y diseño	Estudio MongoDB	36
	Estudio Geohash	6
	Estudio Python	8
	Estructura del proyecto	3
Preparación	Instalación de programas	4
	Descarga de los datos	4
Implementación y Pruebas	Programa de transformación	2
	Cambios programa Python	3
	Carga por índices	10
	Consulta por índices	13
Estudio de rendimiento	Transformación de los datos	3
	Carga de los datos	8
	Consultas	7
	Análisis de resultados	5
	Documentación	27

Tabla 1: Tiempo estimado para la realización del proyecto en días.

Tiempo real		
Área	Tareas	Tiempo (días)
Análisis y diseño	Estudio MongoDB	36
	Estudio Geohash	8
	Estudio Python	8
	Estructura del proyecto	4
Preparación	Instalación de programas	6
	Descarga de los datos	4
Implementación y Pruebas	Programa de transformación	5
	Cambios programa Python	3
	Carga por índices	13
	Consulta por índices	16
Estudio de rendimiento	Transformación de los datos	3
	Carga de los datos	8
	Consultas	7
	Análisis de resultados	6
	Documentación	41

Tabla 2: Tiempo utilizado para la realización del proyecto en días.

Los tiempos empleados podemos verlos con más exactitud en los siguientes diagramas de Gantt (figuras 5 y 6).

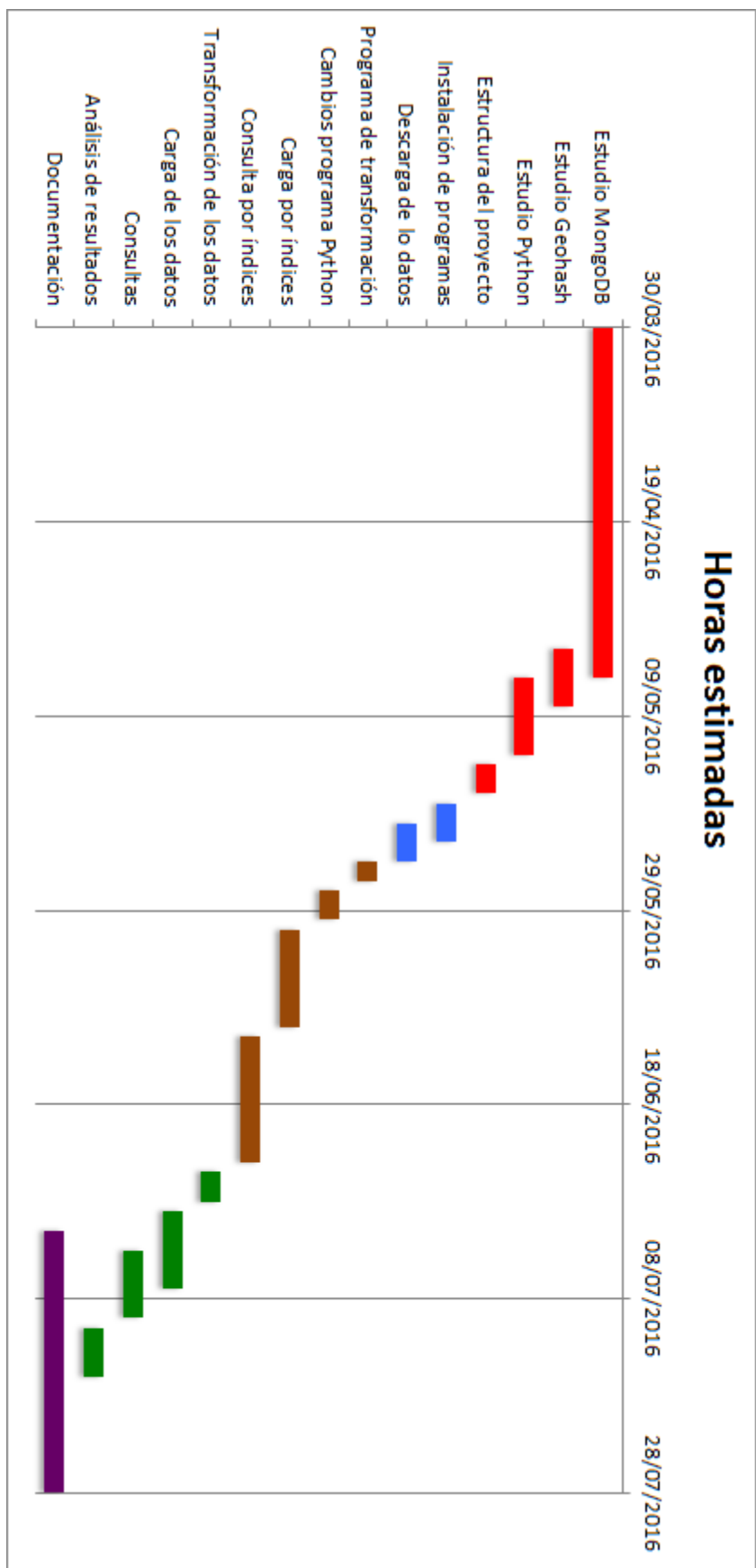


Figura 5: Diagrama Gantt tiempo estimado.

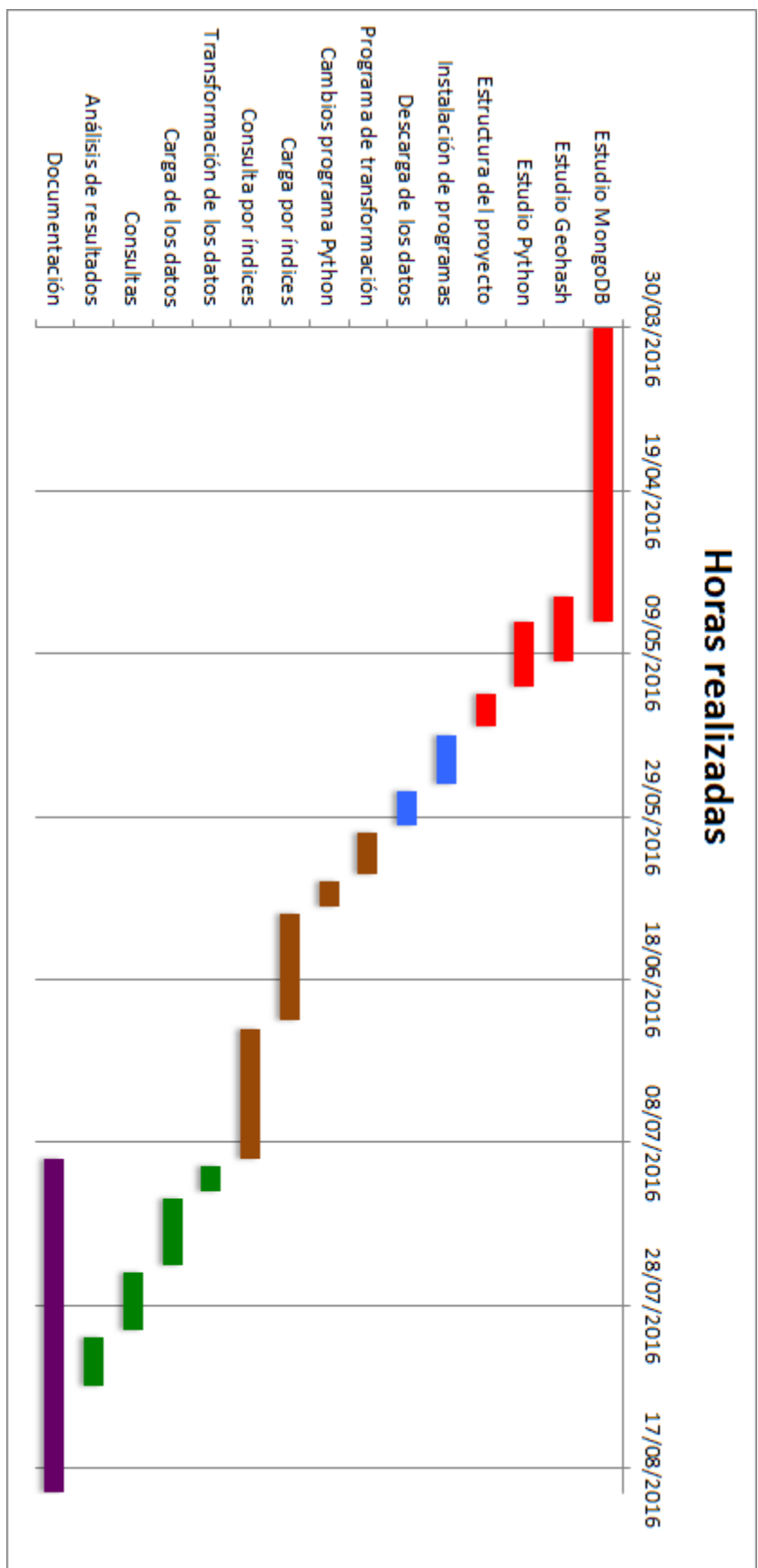


Figura 6: Diagrama Gantt tiempo utilizado.

3.2.- Análisis del sistema desarrollado.

En este apartado veremos en diferentes puntos, las decisiones tomadas a la hora de implementar el programa desarrollado. Para empezar, como es necesario utilizar distintos programas para poder realizar las pruebas, en el diagrama de flujo de la figura 7 se presentan las acciones necesarias para poder realizar las pruebas de tiempos finales.

Como vemos el proceso es bastante sencillo: simplemente hay que descargar los datos desde el repositorio FTP, que por su formato científico, será necesario transformar.

Para la transformación habrá que tener en cuenta que datos nos hemos descargado: si los datos son de tipo QuikSCAT usaremos el programa desarrollado en Java y si los datos son de tipo RapidSCAT usaremos el programa desarrollado en Python.

Una vez obtengamos los datos transformados habrá que cargarlos en la base de datos mediante el programa Java, el cual transformará y adecuará los datos al tipo de índice elegido en este mismo proceso.

Posteriormente, una vez los datos hayan sido cargados, utilizaremos el programa para realizar una serie de consultas y obtener los tiempos de respuesta.

Finalmente se hará un análisis y comparación de estos tiempos para poder tener unos resultados y sacar una conclusión.

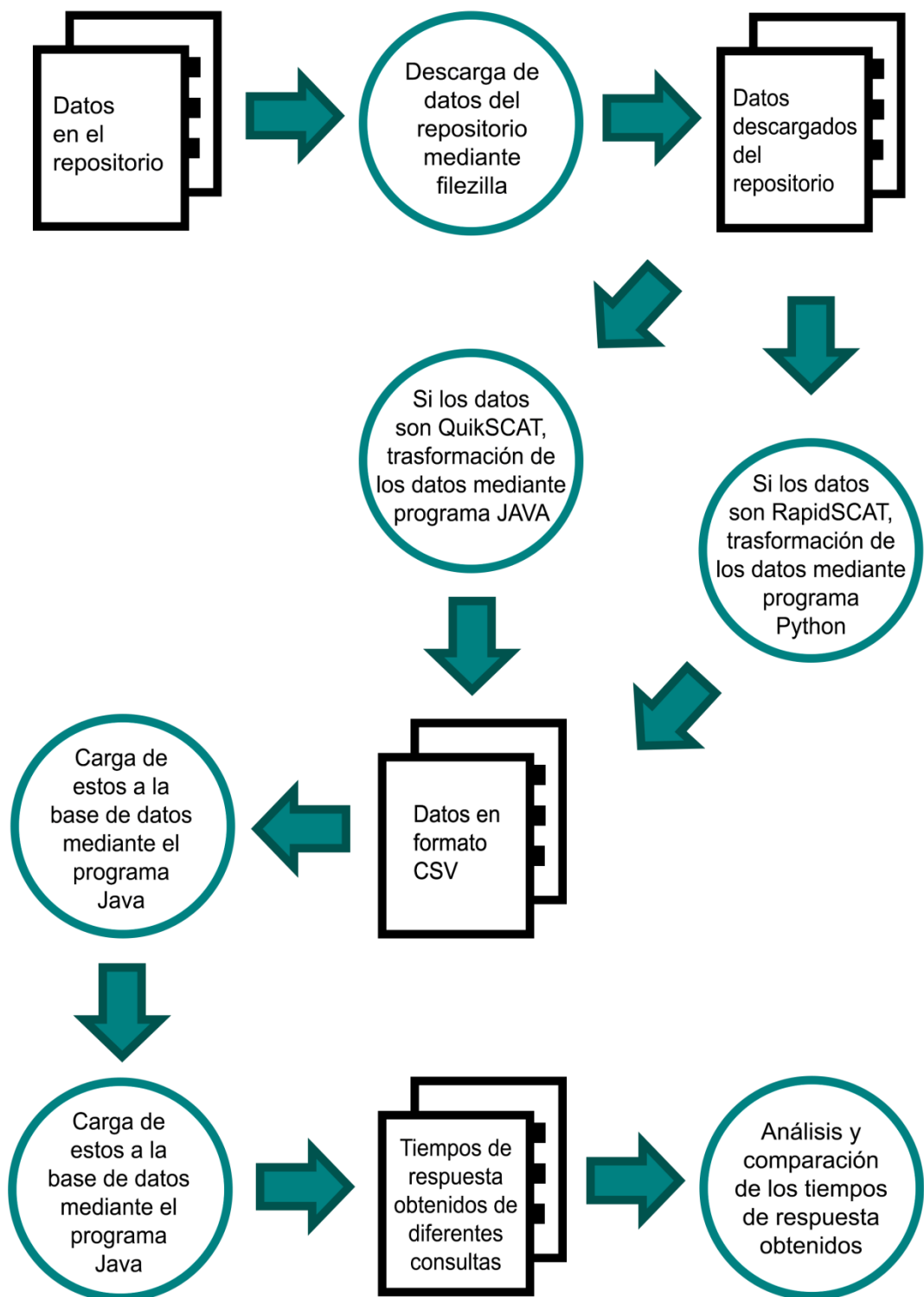


Figura 7: Diagrama de acciones para del proyecto.

3.2.1.- Extracción de los datos del repositorio.

El primer paso a realizar es la obtención de los datos, como ya se ha dicho en apartados anteriores, se han elegido los datos QuikSCAT y RapidSCAT desde el repositorio FTP de la NASA (PODAAC, n.d.2) por su fácil acceso.

Los datos QuikSCAT utilizados corresponden a su versión L2Bv2 (PODAAC, 2016c). Esta versión contiene los datos por cada observación realizada con una resolución de 25 km². Los datos existentes son los correspondientes a las observaciones meteorológicas realizadas desde 1999 hasta 2009. Para las pruebas se han tomado todos los datos del año 2008, por ser el último año completo de observaciones.

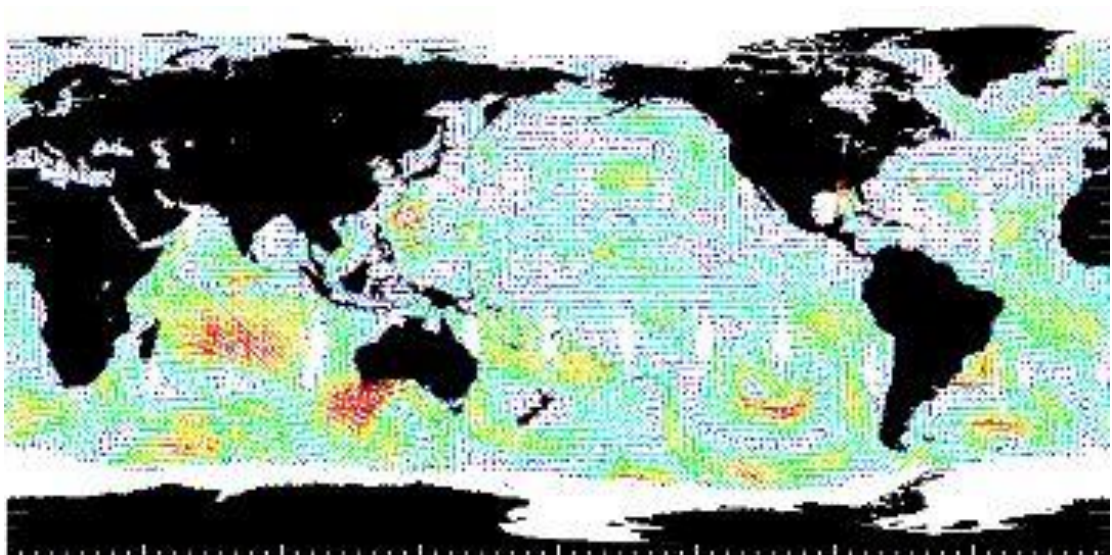


Figura 8: Muestra de datos QuikSCAT L2Bv2 (PODAAC, 2016d)

Los datos RapidSCAT que se han seleccionado son los correspondientes a la versión L2B12v1.2 (PODAAC, 2016e), con una resolución de observaciones meteorológicas cada 12,5 km². Para evaluar el proyecto se extrajeron únicamente los datos del año 2015.

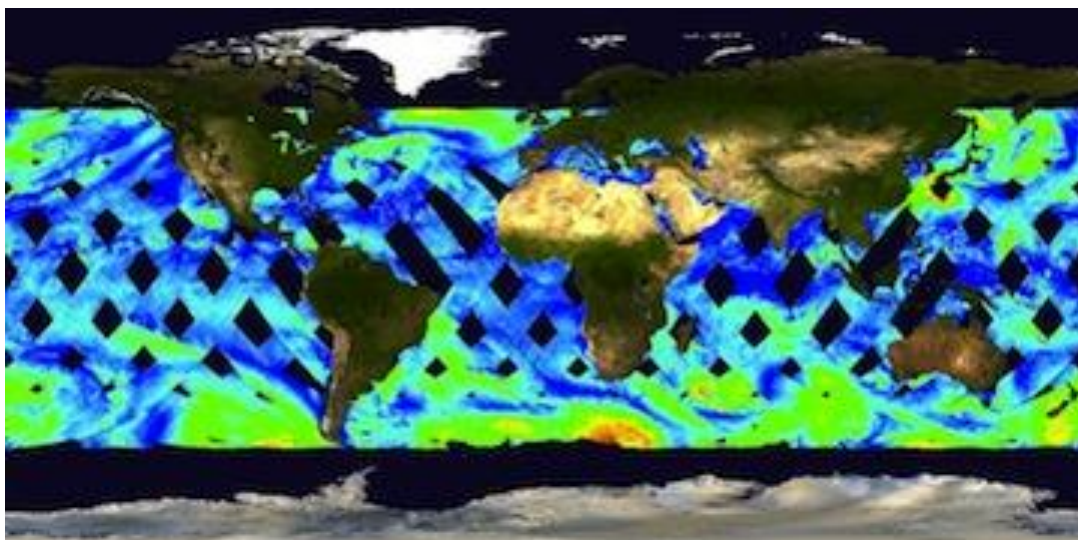


Figura 9: Muestra de datos RapidSCAT L2B12v1.2 (PODAAC, 2016f)

Dado el volumen de los datos que es necesario extraer desde el repositorio, lo que llevará bastante tiempo descargarlos, lo idóneo es utilizar algún tipo de aplicación que permita pausar estas descargas. Por esta razón hemos empleado Filezilla para este fin. Con Filezilla es tan simple como agregarle la dirección del sitio FTP y navegar por el desde las distintas ventanas del programa.

3.2.2.- Estructura del proyecto.

Tras un estudio de lo necesario para realizar el proyecto, se llegó a la conclusión de que había que desarrollar 3 grandes funcionalidades:

- Transformación: para facilitar la transformación de los datos y conseguir que se almacenarán por días completos.
- Cargar por índices: sería necesario transformar los datos del paso anterior para adecuarlos a los distintos índices y posteriormente cargarlos a la base de datos.
- Consulta por índices: una plataforma de consultas que dejase realizar consultas en rangos de fecha, longitud y latitud, las cuales a su vez dependerían del índice elegido.

Además, para facilitar la utilización de estas funcionalidades, se decidió también desarrollar una interfaz gráfica. En sucesivos apartados veremos las diferentes tareas de cada clase.

El siguiente listado representa, en líneas generales, la función de cada clase:

- **MongoGUI:** crea la interfaz gráfica.
- **CargaListener:** escucha los eventos relacionados con la carga de datos en la interfaz.
- **ConsultaListener:** escucha los eventos relacionados con la carga de datos en la interfaz.
- **Lanzador:** crea un hilo que ejecutara la funcionalidad deseada. Sirve para evitar que la interfaz se bloquee cuando se está realizando alguna acción.
- **InterfazGUIMongo:** realiza las funciones de “traductor” entre las clases encargadas de la interfaz y las clases encargadas de las funcionalidades.
- **Transformacion:** se encarga de realizar la transformación de los datos desde el formato científico inicial a CSV.
- **Indices:** clase padre que se encarga de guardar todo lo relacionado con los distintos índices y realizar funcionalidades con la base de datos en base a estos índices.
 - **IndiceSimple:** desarrolla las funcionalidades de la clase padre para el tipo de índice simple.
 - **IndiceCompuesto:** desarrolla las funcionalidades de la clase padre para el tipo de índice compuesto.
 - **Indice2D:** desarrolla las funcionalidades de la clase padre para el tipo de índice 2D.
 - **IndiceGeohashMod:** desarrolla las funcionalidades de la clase padre para el tipo de índice GeohashMod y GeohashModTime.
- **Consultas:** se encarga de realizar las consultas necesarias en la base de datos, filtrar los datos extraídos y transformarlos a un formato general para todos los índices.
- **Tiempo:** guarda marcas de tiempo y duraciones según se le indique.

- OperacionesGeohashMod: contiene las funcionalidades necesarias para calcular una consulta rectangular para GeohashMod o GeohashModTime.
- Singleton: clase que desarrolla el patrón Singleton, que es una clase que permite instanciar atributos de forma única y desde cualquier archivo que lo importe. De esta forma, conseguimos tener un acceso al mismo parámetro desde distintas clases manteniendo los cambios que hubiese padecido anteriormente. Para esta ocasión se guardaron una instancia de la clase Tiempo y una instancia de la clase JTextArea.

En el diagrama de la figura 10 se pueden ver tanto las relaciones como los atributos y métodos de cada clase empleada en el desarrollo del proyecto.

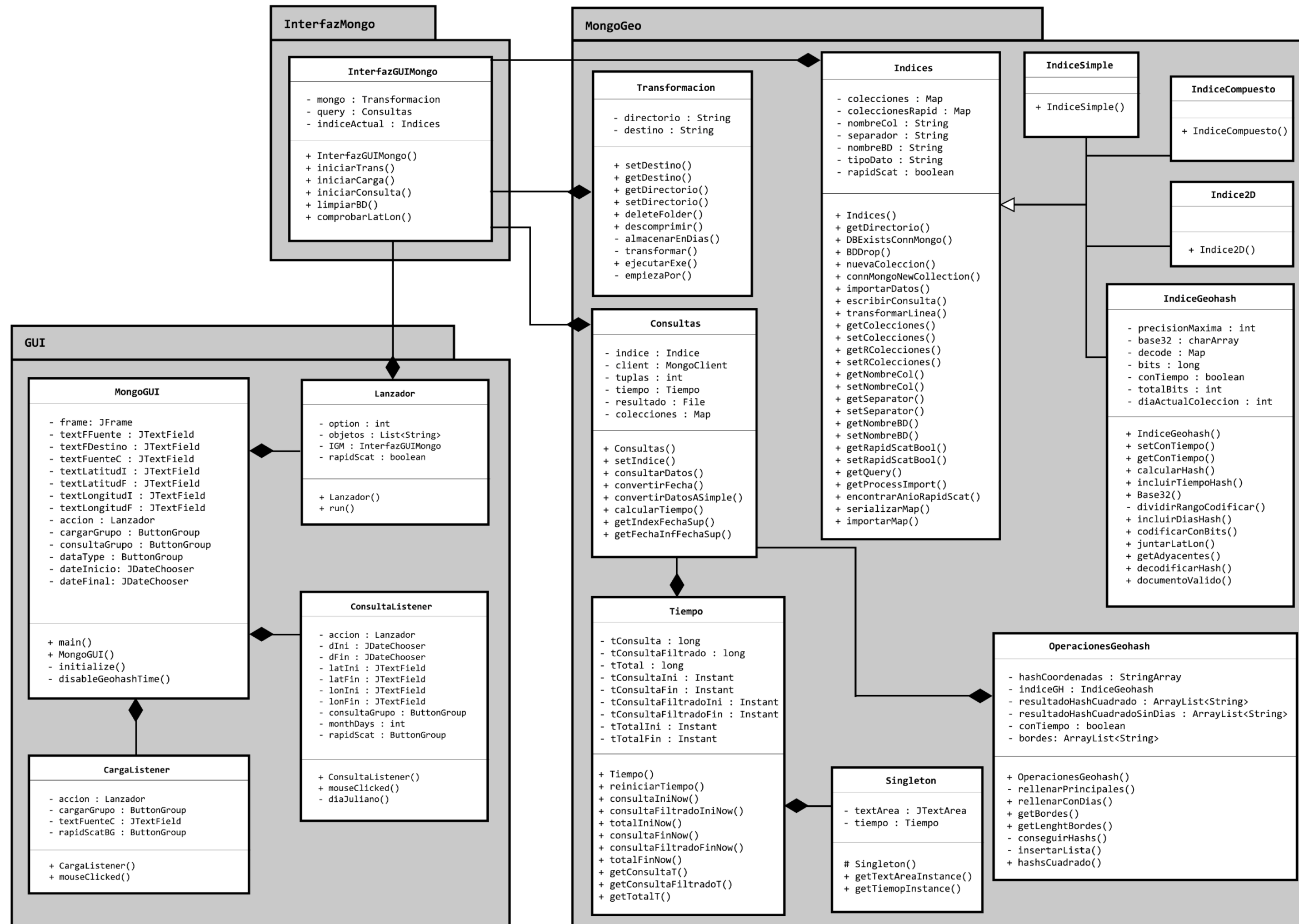


Figura 10: Diagrama de clases de diseño.

3.2.3.- Estructura de la base de datos.

La estructura de la base de datos es un punto importante de este proyecto porque, dependiendo de cómo sea, también formará parte de las tomas de tiempo sobre las consultas.

Se tomaron las siguientes decisiones durante la estructuración de la base de datos:

- Para cada tipo de índice, se guardará en una base de datos diferente, bajo el nombre **“geospatialTipoIndice”** que fácilmente nos permitirá saber que índice usa esta BD. Por otra parte, cada BD podrá contener tanto datos QuikSCAT como datos RapidSCAT, siempre que utilicen el mismo tipo de índice.
- El almacenamiento de los datos por colección se realiza según la cantidad de los días, a fin de tener mayor cantidad de puntos espaciales por cada colección y de esta forma comprobar con mayor eficacia que índice espacial resulta más eficiente. Los datos QuikSCAT almacenan 8 días como máximo por colección y en los datos RapidSCAT se almacenan dos días como máximo por colección. Esta diferencia se debe a la densidad de los datos, siendo los datos RapidSCAT mucho más pesados que los datos QuikSCAT.
- Las colecciones recibirán el nombre de **“TipoDatoSdataTipoIndiceNumCol”** siendo **“TipoDato”** la inicial del tipo de dato a utilizar, **“TipoIndice”** el tipo de índice que utiliza esta colección y **“NumCol”** el número de colección que se utilizará para saber qué días contiene, por ejemplo: QSdataSimple0. El número de colección es dependiente del tipo de dato, es decir, para los datos QuikSCAT se empezará de 0 hasta un número X dependiendo de los días que se carguen, y para los datos RapidSCAT, de nuevo se empezará de 0 hasta un número Y dependiendo de los días que se carguen.
- Utilizando la aplicación en cada base de datos solo se podrá cargar dos veces los datos, una para QuikSCAT y otra para los datos RapidSCAT.

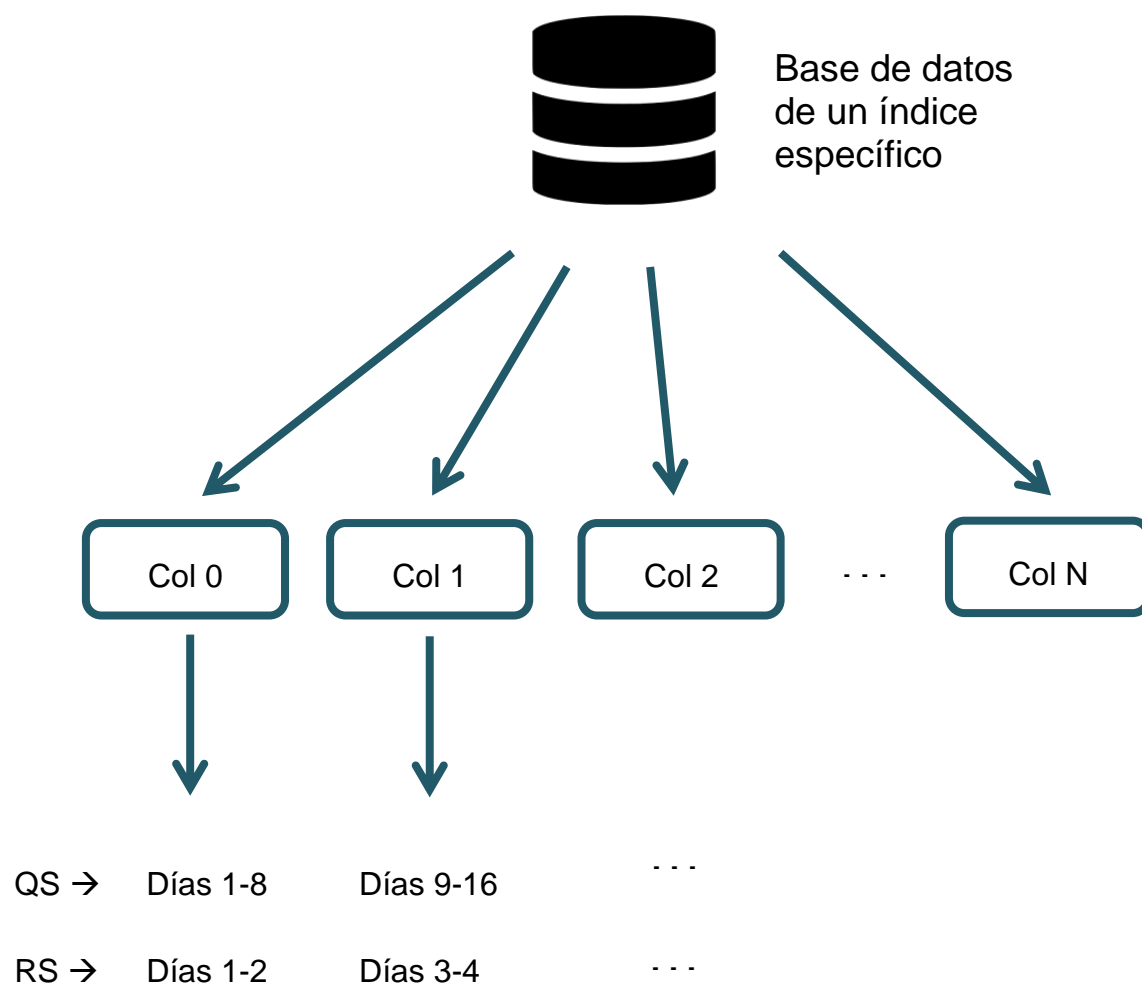


Figura 11: Esquema de diseño de la base de datos.

En el esquema anterior se muestra de una forma rápida como está organizada la base de datos para un índice específico y el contenido de cada colección.

3.2.4.- Transformación de los datos extraídos.

Para transformar los datos QuikSCAT L2Bv2 fue necesario utilizar el programa auxiliar QSreadx64 (Ruíz Romero, Fco., 2012) citado con anterioridad. Para que esta tarea fuese más sencilla, el programa QSreadx64 se encapsuló en nuestra aplicación Java.

Proporcionando el directorio donde se encuentren los datos QuikSCAT descargados por días (tal y como se están en el repositorio FTP de la NASA) y otro directorio donde almacenar la salida, se podrá iniciar un proceso que automáticamente proporcione los datos necesarios al programa QSreadx64

y su salida se irá almacenando en los archivos CSV por días en un subdirectorio llamado “datos” de la ubicación de salida.

Estos ficheros de salida incorporarán en su nombre el tipo de datos que son, así como el año y día que contienen, los cuales servirán para el siguiente paso, la carga a la base de datos. Dentro de estos archivos tenemos los datos de velocidad del viento, dirección del viento y ratio de precipitaciones para una fecha, latitud y longitud, como podemos ver en el siguiente ejemplo:

```
Aaaadddhmmss (Tiempo), latitud, longitud, velocidad del viento, dirección  
del viento, ratio de lluvias
```

```
2008002002845,-64.95,120.94, 7.74, 0.00, 0.00
```

```
2008002002845,-64.85,121.50, 7.43, 0.00, 0.00
```

```
2008002002845,-64.41,122.76, 6.17, 0.00, 0.00
```

```
2008002002845,-64.29,123.17, 6.41, 0.00, 0.00
```

```
2008002002845,-64.14,123.58, 6.51, 0.00, 0.00
```

Para transformar los datos RapidSCAT L2Bv1.2 se ha utilizado el programa en Python desarrollado por Daniel Teomiro (Teomiro Villa, Daniel, 2015), siendo necesario realizar algunos cambios para adecuarlo a los demás datos.

En este programa habrá que realizar el proceso de transformación en dos pasos. Para empezar, suministrándole la ubicación de los datos descargados por días y una ubicación salida, se transformarán los datos desde su formato científico a CSV, para posteriormente unificarlos por días en la ubicación de salida. También es posible realizar esta segunda función independientemente de la transformación en caso de que fuese necesario.

Los documentos de salida unificados por días contienen en su nombre el día de los datos que recopilan. Los datos siguen el siguiente patrón:

Aaaadddhmmss (Tiempo), latitud, longitud, velocidad del viento, dirección del viento, ratio de lluvias

2015231054813,-43.81,15.36,0.00,0.00,0.00

2015231054813,-43.93,15.36,0.00,0.00,0.00

2015231054813,-44.04,15.36,0.00,0.00,0.00

El formato de los distintos valores de ambos tipos de datos cumple las siguientes reglas:

- **Tiempo:** año, concatenado a día y mes juliano, concatenado a hora, minuto y segundo en el que se tomó la muestra. Sin decimales.
- **Latitud:** sus límites son inferior y superior son -90 y 90, respectivamente. Puede tener hasta 2 decimales.
- **Longitud:** sus límites inferior y superior son 0 y 360, respectivamente. Puede tener hasta 2 decimales.
- **Velocidad del viento:** es positivo y puede tener hasta 2 decimales.
- **Dirección del viento:** es positivo y puede tener hasta 2 decimales.
- **Ratio de lluvias:** es positivo y puede tener hasta 2 decimales.

En el algoritmo 1 podemos ver como se realiza la lectura, descompresión y transformación de los datos:

```
función transformar() crea excepciones de entrada y salida y de interrupción
    archivo directorioFuente = nuevo archivo(pathDirectorioFuente)

    si directorioFuente.listarArchivos es distinto de nulo entonces
        printAreaDeTextoInterfaz "Inicio de transformación de datos.\n"
        archivo directorioDatosDestino = nuevo archivo(pathDestino + "\\datos")
        si directorioDatosDestino no existe entonces
            directorioDatosDestino.crearDirectorio()
        fin si
    sino
        directorioDatosDestino.borrar()
        directorioDatosDestino.crearDirectorio()
    fin sino
    archivo transformado = nuevo archivo(".\\QSdata.csv")
    archivo[] directorios = directorioFuente.listarArchivos()
```



```

para cada dir de directorios con índice i hacer
    si dir.esDirectorio() es verdadero entonces
        archivo[] archivos = directorio.listarArchivos()
        para cada arch de archivos con índice j hacer
            si empiezaPor(arch.conseguirNombre()) es verdadero entonces
                si transformado existe entonces
                    transformado.borrar()
                fin si
                cadena[] comando = {"QSreadx64.exe", "0", "1", "0"}
                Proceso p
                cadena pathArch = arch.conseguirPath()
                si pathArch termina en ".gz" entonces
                    pathArch = descomprimir(pathArch)
                fin si
                comando[1] = pathArch
                ConstructorDeProcesos pb = nuevo
                    ConstructorDeProcesos(comando)
                p = pb.empezar()
                p.esperarQueTermine()
                archivo descomprimido = nuevo archivo(pathArch)
                si descomprimido existe entonces
                    descomprimido.borrar()
                fin si
                almacenarEnDias()
            fin si
        fin para
    fin si
fin para
printAreaDeTextoInterfaz "Transformación de datos terminada.\n"
fin si
fin función

```

Algoritmo 1: Transformación de los datos desde formato científico a CSV.

Como vemos, lo primero que ocurre es la creación del directorio “**datos**” dentro del directorio destino, posteriormente se crea el archivo auxiliar “**QSdata.csv**” que almacenará los del fichero que se haya transformado.

Después se insertarán en el programa “**QSreadx64.exe**” los parámetros necesarios, así como la ubicación del archivo que debe transformar, que previamente se ha descomprimido, en caso de que fuese necesario. El archivo descomprimido se borra después de la transformación.

Estos datos se leerán y guardarán en el directorio destino con el nombre de “**QSdata.añodia.csv**”, de forma que este fichero se vaya rellenando hasta

que se lea un día diferente al guardado en el nombre. De esta forma evitamos encontrar filas con datos de distintos días, ya que los datos almacenados en el repositorio no aseguran contener exactamente los días que estos indican.

3.2.5.- Índices utilizados.

En los siguientes apartados hablaremos de los distintos tipos de índices que se han utilizado. Para que sea más sencillo entender las decisiones tomadas sobre estos índices, vamos a definir cuál es el objetivo de cada índice, así como los detalles específicos de cada uno y los distintos tipos utilizados.

Teniendo en cuenta la estructura de los datos, los candidatos para formar una clave única son: tiempo, latitud y longitud. Sin embargo, por si solas no son únicas, pues es posible encontrar varios valores de tiempo similares, como también de latitud y longitud. Por lo tanto, estos índices se basarán en jugar con estos datos para conseguir claves únicas que ayuden a realizar búsquedas más eficientes.

Índice simple.

El objetivo de este índice es crear un valor único que actúe como clave sobre el índice concatenando los valores de tiempo, latitud y longitud en un único número, indexándose con el tipo de índice “Index Field” de MongoDB (MongoDB, 2008-2016b).

Índice compuesto

Este índice pretende crear su valor único creando un índice que indexe a la vez los valores de tiempo, latitud y longitud, utilizando el tipo de índice “Compound Index” de MongoDB.

Índice 2D

Este tipo de índice usa el tipo de índice geoespacial “2D” de MongoDB para los valores de latitud y longitud, y un índice de tipo “Index Field” para el tiempo.

Cabe destacar que este índice utiliza internamente codificación Geohash (MongoDB, 2008-2016c); sin embargo, éste difiere del utilizado en los siguientes índices y del explicado en el apartado 1.4, en el hecho de que divide en longitud y latitud simultáneamente, en lugar de realizarlo en dos pasos alternativos. Por tanto, la precisión elegida siempre debe ser par. La precisión máxima tomada es 20 debido a que genera unos cuadrantes lo suficientemente pequeños para contener pocos datos, y lo suficientemente grandes para contener más de un dato.

Índice GeohashMod

Para crear este índice se utilizó el modelo de codificación comentado en el apartado 1.4 de esta documentación. Se eligió una precisión de 20 bits, utilizando una codificación basada en base-32, que simplemente recopila 5 bits del código y los transforma en un carácter. En la base de datos se crearon dos índices de tipo “Index Field” de MongoDB, uno sobre este código, que se incluyó a cada documento como un nuevo campo bajo el nombre de “Hash” y el otro para la fecha.

Índice GeohashModTime

Este índice se apoya en el anterior para crearse, basándose en la codificación Geohash concatenado con la fecha codificada. La codificación de las coordenadas se realiza de la misma manera que en el caso anterior, mientras que para el día se codifica de una forma distinta, pero utilizando también

base-32. De esta forma se creó un nuevo campo para cada fila denominado Hash, y se incluyó este código con un índice “Index Field”.

3.2.6.- Carga de datos por índices.

La carga de datos a la base de datos se realiza en dos pasos:

1. Se lee un fichero de la ubicación fuente, se transforman los datos según el índice deseado y se guarda en la ubicación del ejecutable de la aplicación.

2. Se envía dicho fichero a la aplicación “mongoimport.exe” con una serie de opciones de MongoDB y éste cargará el fichero en la base de datos en la colección indicada.

Tras un estudio de capacidad se llegó a la conclusión de que las colecciones que almacenasen datos QuickSCAT con un tamaño de 1 GB podrían tener hasta 8 ficheros de datos en total. Para los datos RapidSCAT serían necesarios 2 GB para almacenar tan solo 2 ficheros de datos, dada el volumen tan grande de datos a almacenar.

Para identificar la colección en la que se encuentran los datos de un grupo de días, se ha utilizado un mapa de manera que guarde como clave el primer día que contiene una colección determinada y como valor el nombre de dicha colección. Posteriormente, este mapa se serializa (conversión a formato texto) y se guarda en un fichero para poder utilizarlo en futuras ocasiones.

Es importante que los datos que se carguen estén almacenados en una ruta donde el último directorio se llame “datos”, ya que ésta ha sido la forma que se ha ideado para identificar dónde se encuentran los datos en formato CSV.

En el algoritmo 2 que sigue, se muestra cómo se realiza la transformación y almacenado de los datos en la base de datos:

```
función importarDatos()
    booleano datosRapidScat = conseguirBooleanoRapidScat()
    booleano encontrado = falso
    archivo[] documentos = nuevo archivo(conseguirDirectorioDeCarga).listarArchivos()
    si documentos no es nulo entonces
        para cada documento de documentos y no se ha encontrado hacer
            si documento es igual a "datos" entonces
                documentos = nuevo archivo(conseguirDirectorioDeCarga +
                    "\\datos").listarArchivos()
                encontrado = verdadero
            fin si
        fin para
        entero contador = 0
        cadena fechaDeFichero = ""
        archivo archivoObjetivo = nulo
        cadena extension = nulo
        booleano usandoloIndice2D = falso
```

```

booleano usandoIndiceGhModT = falso
si la clase que invoco el método es instancia de Indice2D entonces
    extension = ".json"
    usandoIndice2D = verdadero
fin si
sino si la clase que invoco el método es instancia de IndiceGeohashMod y
    se usará IndiceGeohashModTime entonces
    extension = ".csv"
    usandoIndiceGHModT = verdadero
fin sino fin si
sino
    extension = ".csv"
fin sino
si datosRapidScat entonces
    cadena[] nombreInicial = documentos[0].conseguirNombre().separar(11)
    fechaDeFichero = nombreInicial[1].subcadena(0,7)
    archivoObjetivo = nuevo archivo("QSdataMongo"+extension)
fin si
sino
    fechaDeFichero = encontrarAñoDeArchivoRapidScat(documentos[0])
    archivoObjetivo = nuevo archivo("RSdataMongo" + extension)
fin sino
almacenarColeccionEnMapa(fechaDeFichero)
crearNuevaColeccionEnBD()
proceso p
constructorDeProcesos pb = conseguirProcesoDeImportacion(archivoObjetivo)
bufferLectura dentro
cadena linea
entero numLinea = 0
intentar
    si archivoObjetivo existe entonces
        archivoObjetivo.borrar()
        archivoObjetivo.crear()
    fin si
fin intentar
capturar excepción de entrada salida
    print "Error al crear el fichero de carga."
devolver
fin capturar
printAreaDeTextoInterfaz "Iniciando carga de datos a la base de datos, con índice
    "+ nombreIndiceUsado +".\n"
para cada documento en documentos con indice indiceDocumentos hacer
    intentar
        flujoEscritura fuera = nuevo flujoEscritura(archivoObjetivo)
        dentro = nuevo bufferLectura(nuevo lecturaArchivo(documento))
        booleano inicioDeFichero = verdadero
        mientras dentro lea líneas en línea hacer
            si numLinea mayor que 1 entonces

```

```

    si usandoIndiceGhModTime es verdadero entonces
        linea = linea + "," + (cadena) contador
    fin si
    linea = transformarLinea(linea)
    si usandoIndice2D es verdadero entonces
        si inicioFichero es verdadero entonces
            linea = linea.subcadena(1)
            linea = "[" + linea
        fin si
        inicioFichero = falso
    fin si
    fuera.escribir(linea)
fin si
    numLinea = numLinea + 1
fin mientras
entero archivoRestantes = documentos.longitud - indiceDocumentos
print "Archivo restantes: " + restantes
p = pb.empezar()
booleano nuevaColeccion = falso
si el proceso p ha acabado correctamente entonces
    print "El archivo se insertó correctamente."
    si indiceDocumentos es menor que documentos.longitud entonces
        contador = contador + 1
        si datosRapidScat es falso entonces
            cadena[] nombre = documentos[
                indiceDocumentos + 1].conseguirNombre().separar("\l")
            fechaFichero = nombre[1].subcadena(0,7)
            si contador es igual a 8 entonces
                si usandoIndiceGeohashModificado es verdadero
                    entonces
                        ponerDiaActualColeccion((entero)
                            fechaFichero.subcadena(4))
                fin si
                nuevaColeccion = verdadero
                contador = 0
            fin si
        fin si
    sino
        si documentos.longitud es mayor que indiceDocumentos + 1
            entonces
                fechaFichero = encontrarAñoDeArchivoRapidScat(
                    documentos[indiceDocumentos+1])
                si contador es igual a 2 entonces
                    si usandoIndiceGeohashModificado es verdadero
                        entonces
                            ponerDiaActualColeccion((entero)
                                fechaFichero.subcadena(4))
                    fin si
                nuevaColeccion = verdadero
            fin si
        fin si
    fin si

```

```

                                contador = 0
                                fin si
                                fin si
                                fin sino
                                si nuevaColeccion es verdadero entonces
                                    nuevaColeccion = falso
                                    almacenarColeccionEnMapa(fechaDeFichero)
                                    crearNuevaColeccionEnBD()
                                    constructorDeProcesos pb =
                                        conseguirProcesoDelImportacion(archivoObjetivo)
                                fin si
                                fin si
                                fin si
                                sino
                                    print "Error al importar el archivo."
                                    devolver
                                fin sino
                                numLinea = 0
                                fuera.cerrarFlujo()
                                dentro.cerrarFlujo()
                                fin intentar
                                capturar excepción de entrada salida
                                    print excepción
                                    print "Error al abrir el archivo de carga"
                                    devolver
                                fin capturar
                                capturar excepción de interrupción
                                    print "Se ha producido un error durante la carga de datos a la base de
datos"
                                devolver
                                fin capturar
                                fin para
                                printAreaDeTextoInterfaz "¡Carga de datos finalizada!"
                                serializarMapaYGuardarEnFichero()
                                si archivoObjetivo existe entonces
                                    archivoObjetivo.borrar()
                                fin si
                                fin si
                                sino
                                    printAreaDeTextoInterfaz "La ruta indicada no es un directorio."
                                fin sino
                                fin función

```

Algoritmo 2: Carga a la base de datos MongoDB.

Como podemos ver, el proceso es bastante sencillo: se crea la colección donde se almacenarán los datos para posteriormente comenzar a leer los archivos transformados y realizar las transformaciones necesarias para el

índice indicado. Estos datos se guardan en un fichero auxiliar llamado “QSdataMongo.csv” o “RSdataMongo.csv” (“.json” en caso del índice 2D) dependiendo de si son datos de QuikSCAT o de RapidSCAT. A continuación se crea la llamada al proceso con las opciones necesarias y se espera que este proceso acabe. Una vez haya finalizado, se cuenta la cantidad de ficheros cargados a la colección actual, de forma que, si se llega al límite de días almacenados por colección, se crea otra nueva colección y se almacena en el mapa correspondiente. Finalmente, una vez el proceso iterativo acaba, se guarda en un fichero el mapa serializado para poder utilizarlo en las consultas.

La creación de nuevas colecciones en la base de datos se realiza según se indica en el algoritmo 3 de la siguiente manera:

```
función crearNuevaColeccionEnBD()
    ClienteMongo clienteMongoDB = nuevo ClienteMongo("localhost", 27017)
    BaseDeDatosMongo bd
    bd = clienteMongoDB.conseguirBaseDeDatos(nombreBaseDeDatos)
    OpcionesColeccion opciones = nuevo OpcionesColeccion()
    opciones.limitarTamañoColeccion(verdadero)
    si datosRapidScat es verdadero entonces
        opciones.tamañoEnBytes(2147483648L)
    fin si
    sino
        opciones.tamañoEnBytes(1073741824)
    fin sino
    bd.crearNuevaColeccion(nombreColeccion, opciones)
    ColeccionMongo<DocumentoBson> mongoCol =
        bd.conseguirColeccion(nombreColeccion)
    entero orden = 1
    según nombreBaseDeDatos hacer
        caso "geospatialSimple"
            mongoCol.crearIndice(nuevo ObjetoBDBasico(
                "YearDayLatitudeLongitudTime", orden))
        fin caso
        caso "geospatialCompound"
            mongoCol.crearIndice(nuevo ObjetoBDBasico("Time",orden).añadir(
                "Latitude", orden).añadir("Longitud", orden))
        fin caso
        caso "geospatial2D"
            mongoCol.crearIndice(nuevo ObjetoBDBasico("Time",orden))
            mongoCol.crearIndice(nuevo ObjetoBDBasico("Loc", orden),
                nuevo OpcionesDeIndice().bits(20))
        fin caso
```



```

caso "geospatialGeohashMod"
    mongoCol.crearIndice(nuevo ObjetoBDBasico("Time", orden))
caso "geospatialGeohashModTime"
    mongoCol.crearIndice(nuevo ObjetoBDBasico("Hash", orden))
fin caso
fin según
clienteMongoDB.cerrar()
fin función

```

Algoritmo 3: Creación de colección en la base de datos.

El proceso se sirve de los distintos atributos de la clase, los cuales almacenan los nombres de la base de datos, así como de la colección que se están utilizando actualmente para realizar una llamada a la base de datos y crear la colección que se utilizará. A esta colección se le caracteriza limitando el tamaño de datos que puede almacenar.

Posteriormente se crea el índice indicado según el nombre de la base de datos en la colección que se acaba de crear.

A continuación, se explicarán las transformaciones que han sido necesarias para cada tipo de índice utilizado.

Índice simple

Como se pretendía crear un valor único que actuase de clave única, fue necesario transportar la latitud de su formato inicial (desde -90° hasta 90°) a un formato únicamente positivo (de 0° a 180°) y multiplicar por 100 tanto latitud como longitud para no almacenar el punto decimal. El patrón obtenido sería el siguiente:

aaaadddhmmssllllllll (entiéndase l como latitud y L como longitud)

Sin embargo, este patrón presentaba un problema, ya que para aumentar la efectividad de la base de datos se pretendía tomar este valor como si fuese un número. Teniendo en cuenta que éste alcanzaba la cifra de 10^{23} , abarcaba mucho más que las de cualquier número entero o entero largo posible, aproximadamente $9,223 * 10^{19}$; por lo cual, tras varias pruebas para intentar crear un índice válido y único se llegó a idear siguiente patrón:

aadddIIIIIIIIIIhmm (entiéndase I como latitud y L como longitud y para evitar pérdida de datos, los segundos se guardaron en otra columna)

Se transformaron los años de manera que, teniendo en cuenta que los datos QuikSCAT están entre los años 1999 y 2009, éstos fuesen representados por un número único que abarque desde el 1 para el año 1999 hasta el 11 para el año 2009.

Para los datos RapidSCAT, teniendo en cuenta que abarcan desde el 2015 hasta el 2016, se codificaron a 7 y 8 respectivamente, utilizando la misma codificación utilizada para QuikSCAT.

Índice compuesto

Dada la estructura de los datos, simplemente fue necesario transportar la longitud desde su rango de valores de 0° a 360° al rango -180° a 180° para llegar a un formato homogéneo y así realizar las consultas de manera más sencilla.

Índice 2D

Puesto que éste índice necesita que los valores de latitud y longitud abarquen valores negativos y positivos (para latitudes desde -90° hasta 90° y para la longitudes desde -180° a 180°) fue necesario transformar la longitud.

Además, es necesario crear una cadena que almacene los valores de latitud y longitud bajo un nombre dentro de la propia fila de los datos. Este tipo de estructuras no son posibles en ficheros con formato CSV, por lo que fue necesario transformar cada fichero que se fuese a cargar a JSON.

Así pues, la latitud y la longitud se almacenan de la siguiente manera:

Loc:{ lat: xx.xx, lon:xxx.xx}

Índice GeohashMod

Para este caso también fue necesario transformar la longitud desde su formato inicial al formato de -180° a 180° grados. Además, hubo que crear un nuevo campo por documento, llamado Hash, el cual contiene el código explicado.

Índice GeohashModTime

Apoyándose en el índice GeohashMod, es necesario transportar la longitud al formato de -180° a 180° grados. Posteriormente se realiza la codificación GeohashMod como en el caso anterior y después se codifica el día como explicamos a continuación. Puesto que los datos QuikSCAT almacenan 8 días por colección, simplemente se ha de tener en cuenta cual sería la posición de un día en una colección determinada (desde el primero hasta el octavo puesto). Este dato se pasa a binario, pudiendo abarcar valores desde 000 hasta 111. Por último, se incluyen otros 2 bits a su derecha y se codifica el resultado en base-32.

Para los datos RapidSCAT se realiza una operación similar, aunque en este caso solo se almacenan dos días; lo que en binario se traduce como un 0 o un 1. Además, en este caso hubo que incluir 4 bits a su derecha en lugar de 2 como en el caso anterior.

3.2.7.- Consulta de datos.

Desde la interfaz gráfica se reciben los datos de fecha inicial, fecha final, latitud inicial, latitud final, longitud inicial y longitud final. Con estos datos se pretende realizar consultas desde la fecha inicial hasta la fecha final y el rectángulo formado por la línea los valores de latitud inicial y longitud inicial, hasta la latitud final y longitud final (formando esta línea, una diagonal del rectángulo de búsqueda). Esta consulta, será almacenada en un fichero con el nombre del tipo de índice utilizado. Se intenta que las consultas sean lo más exactas posibles a los datos que se han pedido; sin embargo, en algunos casos hay que realizar un filtrado previo de los datos consultados, como es el caso de los índices Simple, GeohashMod y GeohashModTime.

En el algoritmo 4 podemos ver cómo se realizan las consultas para cualquier tipo de índice y tipo de dato.

```
función consultarDatos(cadena[] valoresInferiores, cadena[] valoresSuperiores)
    booleano datosRapidScat = conseguirBooleanoRapidScat()
    tiempo.reiniciarTiempo()
    tiempo.empezarMedirTiempoTotal()
    areaDeTexto texto = conseguirAreaDeTextoInterfaz()
    texto.añadir("Iniciando consulta.\n")
    archivo resultado = nuevo archivo("./Resultado"+indice.conseguirNombreSeparador())
si resultado existe entonces
        resultado.borrar()
        intentar
            resultado.crearNuevoArchivo()
        fin intentar
        capturar excepción entrada salida
            print "Error al crear el fichero de consulta "
                +indice.conseguirNombreSeparador()+". "
        devolver
        fin capturar
fin si
```

```
clienteMongoDB = nuevo clienteMongoDB("localhost", 27017)
cadena fechaInferior = valoresInferiores.año + valoresInferiores.dia
cadena fechaSuperior = valoresSuperiores.año + valoresSuperiores.dia
```

```
si datosRapidScat es falso entonces
    colecciones = indice.conseguirColeccionesQuik()
fin si
sino
    colecciones = indice.conseguirColeccionesRapid()
fin sino
cadena[] fechasDeColecciones =
    conseguirFechaInferiorFechaSuperiorColecciones(fechaInferior, fechaSuperior)
fechaInferior = fechaDeColecciones[0]
fechaSuperior = fechaDeColecciones[1]
entero indiceMapaColecciones = (entero) fechaInferior
ObjetoBD consulta = nulo
BaseDeDatosMongo bd =
    clienteMongoDB.conseguirBaseDeDatos(indice.conseguirNombreBaseDeDatos)
cadena[] parametrosDeEscritura = nulo
si indice es instancia de IndiceSimple entonces
    cadena limiteInferior, limiteSuperior
    limiteInferior = convertirDatosASimple((entero) valoresInferiores.año +
        valoresInferiores.dia, valoresInferiores.latitud, valoresInferiores.longitud)
    entero fechaSuperiorAuxiliar = (entero) fechaSuperior
    fechaSuperiorAuxiliar = conseguirIndiceFechaSuperior(fechaSuperiorAuxiliar,
        datosRapidScat)
```

```

    limiteSuperior = convertirDatosASimple(fechaSuperiorAuxiliar,
        valoresSuperiores.latitud, valoresSuperiores.longitud)
    consulta = indice.conseguirConsulta(limiteInferior, limiteSuperior)
    parametrosDeEscritura = nuevo cadena[{valoresInferiores.latitud,
        valoresInferiores.longitud, valoresSuperiores.latitud,
        valoresSuperiores.longitud}]
fin si
sino si indice es instancia de IndiceGeohashMod entonces
    op = nuevo OperacionGeohashMod(indice)
    lista<cadena> hashes = op.calcularHashCuadrado((real)valoresInferiores.latitud,
        (real) valoresInferiores.longitud,(real) valoresSuperiores.latitud,
        (real) valoresSuperiores.longitud)
    hashesConsulta = hashes
    si no se usara IndiceGeohashModTime entonces
        cadena[] fecha = convertirFecha(indiceFechaInferior,
            conseguirIndiceFechaSuperior(indiceFechaInferior, datosRapidScat))
        fecha[0] = valoresInferiores.año + valoresSuperiores.dia + "000000"
        hashes.añadir(fecha[0])
        hashes.añadir(fecha[1])
    fin si
    sino
        si datosRapidScat es verdadero entonces
            hashes = op.rellenarConDias((entero) valoresInferiores.dia -
                ((entero) ((cadena) indiceFechaInferior).subcadena(4,7)), 8)
        fin si
        sino
            hashes = op.rellenarConDias((entero) valoresInferiores.dia - ((entero)
                ((cadena) indiceFechaInferior).subcadena(4,7)), 8)
        fin sino
    fin sino
    consulta = indice.conseguirConsulta(hashes.convertirAAray())
    parametrosDeEscritura = nuevo cadena[op.conseguirLongitudBordes() + 4]
    cadena[] hashesBordes = op.conseguirBordes()
    mientras entero i = 4 sea menor que parametrosDeEscritura.longitud hacer
        parametrosDeEscritura[i] = hashBordes[i-4]
    fin mientras
    parametrosDeEscritura[0] = valoresInferiores.latitud
    parametrosDeEscritura[1] = valoresSuperiores.latitud
    parametrosDeEscritura[2] = valoresInferiores.longitud
    parametrosDeEscritura[3] = valoresSuperiores.longitud
fin sino fin si
sino
    cadena[] fecha
    fecha = convertirFecha(indiceFechaInferior,
        conseguirIndiceFechaSuperior(indiceFechaInferior, datosRapidScat))
    fecha[0] = valoresInferiores.año + valoresInferiores.dia + "000000"
    consulta = indice.conseguirConsulta(fecha[0], fecha[1], valoresInferiores.latitud,
        valoresSuperiores.latitud, valoresInferiores.longitud,
        valoresSuperiores.longitud)
fin sino

```

```

booleano dentro = falso
booleano primeraVuelta = verdadero
si colecciones.conseguir((cadena) indiceFechaInferior) es distinto de nulo entonces
    mientras indiceFechaInferior - (entero) fechaSuperior sea distinto de 0 y salir sea
    falso hacer
        si colecciones.conseguir((cadena) indiceFechaInferior) es distinto de nulo
        entonces
            tiempo.empezarMedirTiempoConsulta()
            ColeccionMongo<DocumentoBson> coleccionMon =
                bd.conseguirColeccion(colecciones.conseguir(
                    (cadena) indiceFechaInferior))
            CursorMongo<DocumentoBson> cursor =
                coleccionMon.encontrar((Bson) consulta).iterador()
            tiempo.finMedirTiempoConsulta()
            tuplas = tuplas + indice.escribirConsulta(cursor, parametrosDeEscritura,
                resultado)
            indiceFechaInferior = conseguirIndiceFechaSuperior(indiceFechaInferior,
                datosRapidScat)
        si indice es instancia de IndiceSimple entonces
            cadena limiteInferior, limiteSuperior
            limiteInferior = convertirDatosASimple(indiceFechaInferior,
                valoresInferiores.latitud, valoresInferiores.longitud)
            limiteSuperior = convertirDatosASimple(
                conseguirIndiceFechaSuperior(indiceFechaInferior,
                datosRapidScat),valoresSuperiores.latitud,
                valoresSuperiores.longitud)
            consulta = indice.conseguirConsulta(limiteInferior, limiteSuperior)
        fin si
    sino si indice es instancia de IndiceGeohashMod entonces
        lista<cadena> hashes = hashesConsulta
        si no se usara IndiceGeohashModTime entonces
            cadena[] fechas
            fechas = convertirFecha(indiceFechaInferior,
                conseguirIndiceFechaSuperior(indiceFechaInferior,
                datosRapidScat))
            hashes.añadir(fechas[0])
            hashes.añadir(fechas[1])
        fin si
    sino si datosRapidScat es verdadero y primeraVuelta es verdadero
    entonces
        hashes = op.rellenarConDias(0,2)
        primeraVuelta = falso
    fin sino fin si
    sino si primeraVuelta es verdadero entonces
        hashes = op.rellenarConDias(0,8)
        primeraVuelta = falso
    fin sino fin si
    consulta = conseguirConsulta(hashes.convertirAArray)
fin sino fin si
sino
    cadena[] fechas

```

```

        fechas = convertirFecha(indiceFechaInferior,
                                conseguirIndiceFechaSuperior(indiceFechaInferior,
                                                                datosRapidScat))
        consulta = indice.conseguirConsulta(fechas[0], fechas[1],
                                            valoresInferiores.latitud, valoresSuperiores.latitud,
                                            valoresInferiores.longitud,
                                            valoresSuperiores.longitud)

    fin sino
fin si
dentro = verdadero
fin mientras
si indice es instancia de IndiceSimple entonces
    cadena limiteInferior, limiteSuperior
    limiteSuperior = convertirDatosASimple((entero) (valoresSuperiores.año +
                                                    valoresSuperiores.dia), valoresSuperiores.latitud,
                                                    valoresSuperiores.longitud))
    si dentro es falso entonces
        limiteInferior = convertirDatosASimple( (entero)(valoresInferiores.año +
                                                    valoresInferiores.dia), valoresInferiores.latitud,
                                                    valoresInferiores.longitud)
    fin si
    sino
        limiteInferior = convertirDatosASimple(indiceFechaInferior,
                                                valoresInferiores.latitud, valoresInferiores.longitud)
    fin sino
    consulta = indice.conseguirConsulta(limiteInferior, limiteSuperior)
fin si
sino si indice es instancia de IndiceGeohashMod entonces
    lista<cadena> hashes = nuevo lista<cadena>()
    si no se usara IndiceGeohashModTime entonces
        si dentro es falso entonces
            hashes = op.rellenarConDias((entero) valoresInferiores.dia - ((entero)
                                                                ((cadena) indiceFechaInferior).subcadena(4,7)),
                                        (entero) valoresSuperiores.dia -
                                        (entero) ((cadena) indiceFechaInferior).subcadena(4,7)))
        fin si
        sino
            hashes = op.rellenarConDias(0, (entero) valoresSuperiores.dia -
                                        (entero) ((cadena) indiceFechaInferior).subcadena(4,7)))
        fin sino
    fin si
    sino
        cadena[] fechas = nuevo cadena[2]
        si dentro es falso entonces
            fechas[0] = valoresInferiores.año + valoresInferiores.dia + "000000"
        fin si
        sino
            fechas = convertirFecha(indiceFechaInferior,
                                    conseguirIndiceFechaSuperior(indiceFechaInferior,
                                                                    datosRapidScat))
        fin sino

```

```

        fechas[1] = valoresSuperiores.año + valoresSuperiores.día + "235959"
        hashes = hashesConsulta
        hashes.añadir(fechas[0])
        hashes.añadir(fechas[1])
    fin sino
    consulta = indice.conseguirColeccion(hashes.convertirAArray)
fin sino fin si
sino
    cadena[] fechas = nuevo cadena[2]
    si dentro es falso entonces
        fechas[0] = valoresInferiores.año + valoresInferiores.día + "000000"
    fin si
    sino
        fechas = convertirFecha(indiceFechaInferior,
                                conseguirIndiceFechaSuperior(indiceFechaInferior,
                                                                datosRapidScat))
    fin sino
    fechas[1] = valoresSuperiores.año + valoresSuperiores.día + "235959"
    consulta = conseguirConsulta(fechas[0], fechas[1], valoresInferiores.latitud,
                                valoresSuperiores.latitud, valoresInferiores.longitud,
                                valoresSuperiores.longitud)
    fin sino
    tiempo.empezarMedirTiempoConsulta()
    ColeccionMongo<DocumentoBson> coleccionMon =
        bd.conseguirColeccion(colecciones.conseguir(
            (cadena) indiceFechaInferior))
    CursorMongo<DocumentoBson> cursor =
        coleccionMon.encontrar((Bson) consulta).iterador()
    tiempo.finMedirTiempoConsulta()
    tuplas = tuplas + indice.escribirConsulta(cursor, parametrosDeEscritura, resultado)
    tiempo.finMedirTiempoTotal()
    calcularTiemposYMostrar()
fin si
sino
    PrintAreaDeTextoInterfaz "No se ha podido encontrar la fecha suministrada en la
base de datos.\n"
    fin sino
    clienteMongoDB.cerrar()
fin función

```

Algoritmo 4: Consulta de datos a la base de datos MongoDB.

Por cómo están almacenados los datos en la base de datos, la mayoría de las consultas tendrán la latitud y longitud como valores fijos, mientras que los días o años será necesario ir cambiándolos para adecuarlos a los datos almacenados en la colección que se vaya a consultar.

Como podemos ver, el proceso tiene tres partes que son distintas: la primera consulta, la última consulta y el resto.

Para la primera consulta hay que tener en cuenta la fecha inicial que se desea consultar, la cual la recibiremos desde los parámetros de entrada. Desde esa fecha, se toma la colección a la que pertenece y se calcula el último día al que pertenece la colección, consultándose el rectángulo formado por las variables de longitud y latitud.

La consulta final tiene en cuenta la fecha final que se desea consultar para, de forma similar a la primera consulta, conseguir la colección a la que pertenece y realizar la consulta del rectángulo desde el primer día hasta el día indicado por el parámetro en esta colección.

El resto de las consultas toman como límites inferior y superior de la fecha, el primer y último día almacenados en la colección a la que pertenecen.

En caso de que los datos solo abarquen una única colección, se toma como si fuese una “última consulta” y se pone como límite inferior el obtenido desde los parámetros.

3.2.8.- Índices GeohashMod y GeohashModTime.

Estos índices se han creado específicamente para este caso siendo el GeohashModTime el más concreto para el tipo de datos utilizado.

La intención de estos índices es crear unos valores hash que identifiquen una zona geográfica del planeta en el cual se almacenarán los datos de las misiones QuikSCAT y RapidSCAT. Para ello se toma la idea de la codificación Geohash mencionada en el capítulo 1 apartado 4.

De esta forma, para el índice GeohashMod fue necesario implementar un codificador/decodificador de forma que podamos obtener el hash relacionado con un área específica del mapa mundial y viceversa, y un ayudante que calcularía los distintos códigos de los cuadrantes para poder realizar una consulta rectangular.

El índice GeohashModTime, apoyándose en todo lo anterior, subsanaría el problema que tiene la utilización de únicamente el índice Geohash sobre este tipo de datos: el tiempo. Ya que la formación de una clave única para los datos del repositorio es posible solamente, si tenemos en cuenta tanto el tiempo, como la latitud y la longitud. Por lo tanto, el índice GeohashModTime contendría tanto zonas de latitud y longitud como zonas temporales en un código hash.

En los algoritmos 5 y 6 siguientes presentamos cómo se realiza el proceso de cálculo del hash asignado a una zona geográfica.

función devuelve cadena calcularHash(real latitud, real longitud, cadena fecha)

```

    real largo[] rangoLatitud = {-90,90}
    real largo[] rangoLongitud = {-180,180}
    booleano bitImpar = verdadero
    mientras totalBits sea menor que precisionMaxima hacer
        si bitImpar es verdadero entonces
            dividirRangoCodificar(longitud, rangoLongitud)
        fin si
        sino
            dividirRangoCodificar(latitud, rangoLatitud)
        fin sino
    fin mientras
    bits <- mover 44 bits hacia la izquierda
    si no se usara IndiceGeohashModTime entonces
        devolver Base32()
    fin si
    sino
        devolver incluirTiempoHash(Base32(), fecha)
    fin sino
fin función

```

Algoritmo 5: Cálculo de hash a partir de un punto en el espacio.

función dividirRangoCodificar(real largo valor, real largo[] rango)

```

    real largo medio = (rango[0] + rango[1]) / 2
    si valor es mayor o igual que medio entonces
        totalBits = totalBits + 1
        bits <- desplazar 1 bit hacia la izquierda
        bits = (bit izquierdo de bits) or 0x1
        rango[0] = medio
    fin si
    sino
        totalBits = totalBits + 1
        bits <- desplazar un bit hacia la izquierda

```

```

        rango[1] = medio
    fin sino
fin función

```

Algoritmo 6: Cálculo de bit según punto para índice GeohashMod.

Las variables “bits”, “totalBits” y “precisionMaxima” almacenan, respectivamente, la cadena de bits que forma el hash, la cantidad de bits que hasta ahora se han insertado en la variable “bits” y la cantidad máxima que habrá.

El algoritmo 5 otorga turnos para las divisiones de latitud y longitud, y el algoritmo 6 realiza dichas divisiones y almacena el bit correspondiente dependiendo de dónde se encuentre el punto proporcionado en el buffer de bits. Una vez este proceso acaba, simplemente hay que codificarlo a base-32 y devolverlo, o bien devolver la cadena codificada incluyendo el día, dependiendo de si utilizamos el índice GeohashMod o GeohashModTime. La codificación la vemos desarrollada en el algoritmo 7.

```

función devuelve cadena Base32()
    ConstructorDeCadena buffer = nuevo ConstructorDeCadena
    entero largo primeros5Bits = 0xf800000000000000l
    entero largo copiaBits = bits
    entero trozosParciales = totalBits / 5
    entero i = 0
    mientras i sea menor que trozosParciales hacer
        entero puntero = (entero) ((copiaBits and primeros5Bits) -> desplazar 59 bits hacia
la derecha)
        buffer.añadir(base32[puntero])
        copiaBits <- desplazar 5 bits hacia la izquierda
        i = i + 1
    fin mientras
    totalBits = 0
    bits = 0
    devolver buffer.convertirACadena()
fin función

```

Algoritmo 7: Codificación base32.

Por último, en caso de que hubiésemos seleccionado la opción de incluir el tiempo en la codificación sería necesario que la cadena que contiene el hash que se ha calculado pasase por la siguiente función.

```

función devuelve cadena incluirTiempoHash(cadena hash, cadena diaColeccion)

```

```

entero diaH = (entero) diaColeccion
si datosRapidScat es verdadero entonces
    diaH <- desplazar 4 bits hacia la izquierda
fin si
sino
    diaH <- desplazar 2 bits hacia la izquierda
fin sino
hash = hash + base32[diaH]
devolver hash
fin función

```

Algoritmo 8: Codificación base-32 del día.

4.- Resultados.

A continuación, se hará una introducción sobre cómo se realizarán las consultas para la tomas de tiempos de rendimiento y se mostrarán los distintos resultados de las consultas realizadas sobre la base de datos MongoDB en diferentes apartados con graficas de líneas para comparar los resultados.

4.1.- Preparación de pruebas de rendimiento.

Una vez que tenemos creado el sistema que nos proporciona el acceso a un sencillo almacenamiento y que permite la consulta de los datos a nuestro antojo, solo nos quedaba explorar como realizar las pruebas de rendimiento, las cuales suponen el fin último de este proyecto.

Para ello se han tomado las siguientes combinaciones de consultas, tipo de datos e índices:

- Consultas: sobre cada combinación de tipo de datos e índices se realizan tres grupos distintos de consultas: temporales, espaciales y espacio-temporales. Las consultas temporales nos devuelven todos los datos que se encuentren en un rango de dos fechas. Las consultas espaciales devuelven los datos contenidos en una ventana espacial (los contenidos en el rectángulo formado por dos puntos del espacio) de una determinada fecha. Y las consultas espacio-temporales combinan ambas consultas espacial y temporal; es decir, devuelve todos los datos de una ventana espacial en un rango temporal determinando.
- Tipo de datos: cada combinación de consultas e índices se realizan tanto para los datos QuikSCAT como para RapidSCAT.
- Índices: para cada combinación de consultas y tipos de datos se utilizan los cinco índices elegidos y explicados en el apartado anterior de este mismo capítulo.

También se comparará el tiempo de respuesta de cada tipo de consulta para cada tipo de índice, con lo que podemos tener una idea bastante aproximada

del comportamiento de cada índice. Además, se tomarán mediciones del tiempo de filtrado, simplemente por si es necesario hacer referencia a él en algún momento, aunque no se reflejará en las gráficas.

Más específicamente, los datos que se utilizan para realizar las consultas son:

- Consultas temporales: todos los datos almacenados entre las latitudes -90° y 90° y las longitudes entre -180° y 180° . Se incrementarán los días en 1 hasta un máximo de 12. Estos días son los siguientes
 1. 1 de enero de 2008.
 2. Del 2 al 3 de febrero de 2008.
 3. Del 3 al 5 de marzo de 2008.
 4. Del 4 al 7 de abril de 2008.
 5. Del 5 al 9 de mayo de 2008.
 6. Del 6 al 11 de junio de 2008.
 7. Del 7 al 13 de julio de 2008
 8. Del 8 al 15 de agosto de 2008.
 9. Del 9 al 17 de septiembre de 2008.
 10. Del 10 al 19 de octubre de 2008.
 11. Del 11 al 21 de noviembre de 2008.
 12. Del 12 al 23 de diciembre de 2008.
- Consultas espaciales: todos los datos almacenados provenientes de las observaciones del día 16 de julio de 2008, incrementando las ventanas de consulta espacial 5° en 5° , a excepción de la primera consulta en la que se toman 4° de latitud y de longitud, con el fin de adecuar las diferencias entre consultas a múltiplos de 5:
 1. Latitudes entre 48° y 49° y Longitudes entre -16° y -15° .
 2. Latitudes entre 43° y 48° y Longitudes entre -21° y -16° .
 3. Latitudes entre 33° y 43° y Longitudes entre -31° y -21° .
 4. Latitudes entre -21° y -6° y Longitudes entre 86° y 101° .
 5. Latitudes entre -41° y -21° y Longitudes entre 66° y 86° .
 6. Latitudes entre -7° y -32° y Longitudes entre -93° y -118° .

7. Latitudes entre -62° y -32° y Longitudes entre -148° y -118° .

- Consultas espacio-temporales: se utilizarán los mismos datos de latitud y longitud que las espaciales, así como las fechas de los datos utilizados en las consultas temporales: 1, 3, 5, 6, 7, 9 y 11.

Los datos QuikSCAT utilizan estos parámetros mientras que para los RapidSCAT se consultan desde los mismos puntos espaciales en latitud y longitud a excepción de aquellas latitudes que excedan en valor absoluto a 56° , ya que esos son los límites de toma de estos de datos por la Estación Espacial Internacional, en cuyo caso se utilizan 56° o -56° según el caso. Para las consultas temporales se utilizan los datos almacenados de las fechas siguientes:

1. 26 de agosto de 2015.
2. Del 28 al 29 de agosto de 2015.
3. Del 2 al 4 de septiembre de 2015.
4. Del 6 al 9 de septiembre de 2015.
5. Del 11 al 15 de septiembre de 2015.
6. Del 17 al 22 de septiembre de 2015.
7. Del 24 al 30 de septiembre de 2015.
8. 15 de octubre de 2015 como día fijo para las consultas espaciales.

Las consultas se realizan de forma separada dependiendo del tipo de dato debido a su independencia entre sí: al estar almacenados por días, esto crea un gran agujero a la hora de realizar las consultas teniendo en cuenta que los datos QuikSCAT son del año 2008 y los datos RapidSCAT son del 2015.

Cabe destacar que la cuantía de datos que hay en el repositorio de RapidSCAT del año 2015 parecen bastante inferiores a los del 2008 de QuikSCAT ya que las tomas de RapidSCAT para el 2015 apenas contienen un tercio de dicho año y están limitadas en latitud, mientras que las del QuikSCAT son del año y el mapa completos. No obstante, el volumen de datos obtenidos por RapidSCAT cuadruplica a los del QuikSCAT. Por esto, aunque las consultas parecen inferiores a las que se utilizan para los datos

QuikSCAT, realmente son mucho más pesadas en volumen de datos recuperados.

4.2.- Datos QuikSCAT.

4.2.1.- Consultas temporales.

Como se puede observar en la figura 12, los índices Compuesto y 2D son los claros candidatos para ser los más eficientes para los parámetros suministrados en las consultas. Sin embargo, en la gráfica de la figura 12 no se encuentran las consultas con el índice GeohashModTime debido a que dichas consultas excedían el tamaño máximo de documento de MongoDB, siendo este de 16 MB.

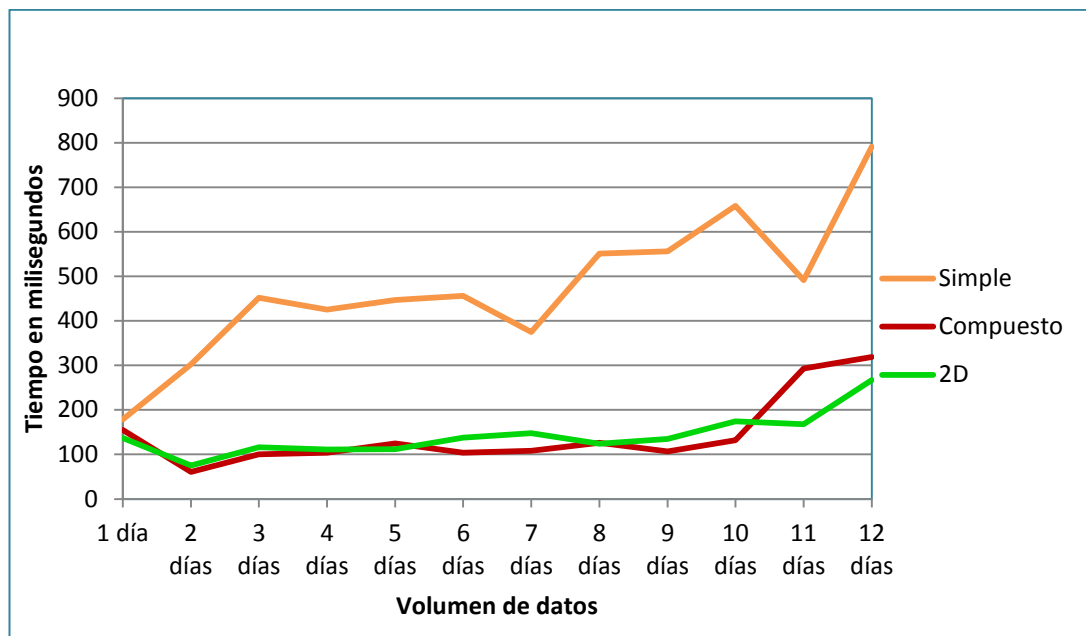


Figura 12: Comparación de los tiempos de respuesta de las consultas temporales de los datos QuikSCAT almacenados en MongoDB utilizando los índices Simple, Compuesto y 2D.

Si se reducen 10 grados la ventana de consultas realizadas, quedando los intervalos entre -80° y 80° de latitud y entre -170° y 170° de longitud, sí que es posible realizar las consultas temporales. El resultado puede comprobarse en la figura 13.

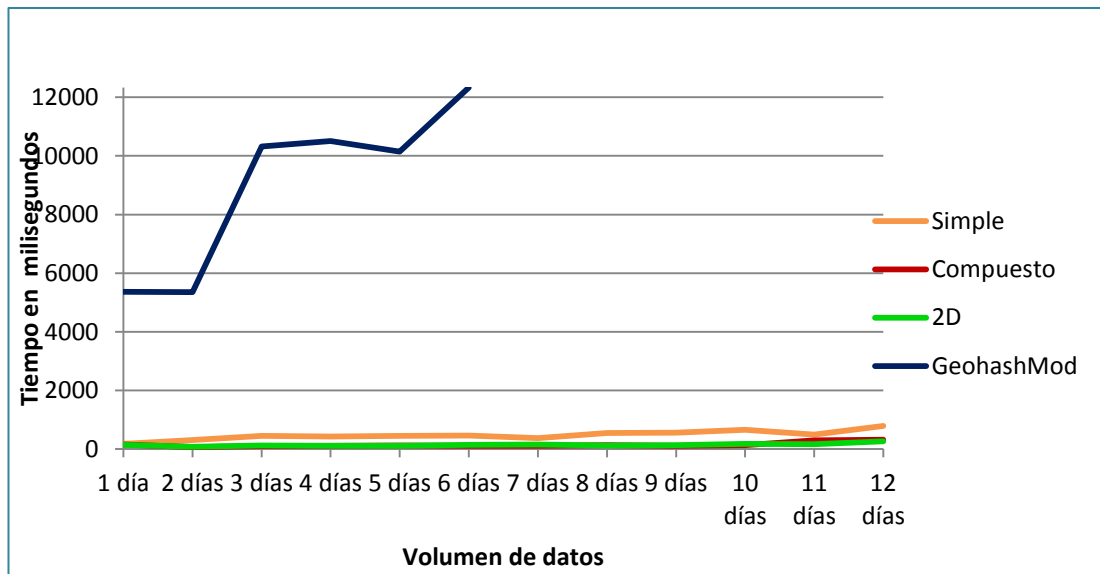


Figura 13: Comparación de los tiempos de respuesta de las consultas temporales de los datos QuikSCAT almacenados en MongoDB utilizando los índices Simple, Compuesto, 2D y GeohashMod, y con una reducción de los intervalos de consulta en 10°.

La figura 13 nos sugiere que incluir el índice GeohashModTime no es nada competitivo con respecto a los índices proporcionados por MongoDB para este tipo de consultas.

4.2.2.- Consultas espaciales.

Como puede observarse en la figura 14, el índice 2D proporcionado por MongoDB no produce un buen resultado para consultas de rangos (o ventana) espacial pequeños, de pocos grados de intervalo. Esto es bastante extraño, sobre todo para el índice 2D, el cual es un índice específico de MongoDB para datos geoespaciales. El resultado para las consultas espaciales de un grado cuadrado de latitud y longitud es de 2 segundos y 793 milésimas haciendo que se salga de la gráfica. En este estudio comparativo también podemos ver el comportamiento de los dos índices creados para este proyecto. Según se observa, llegan a ser tan eficaces como los implementados por MongoDB.

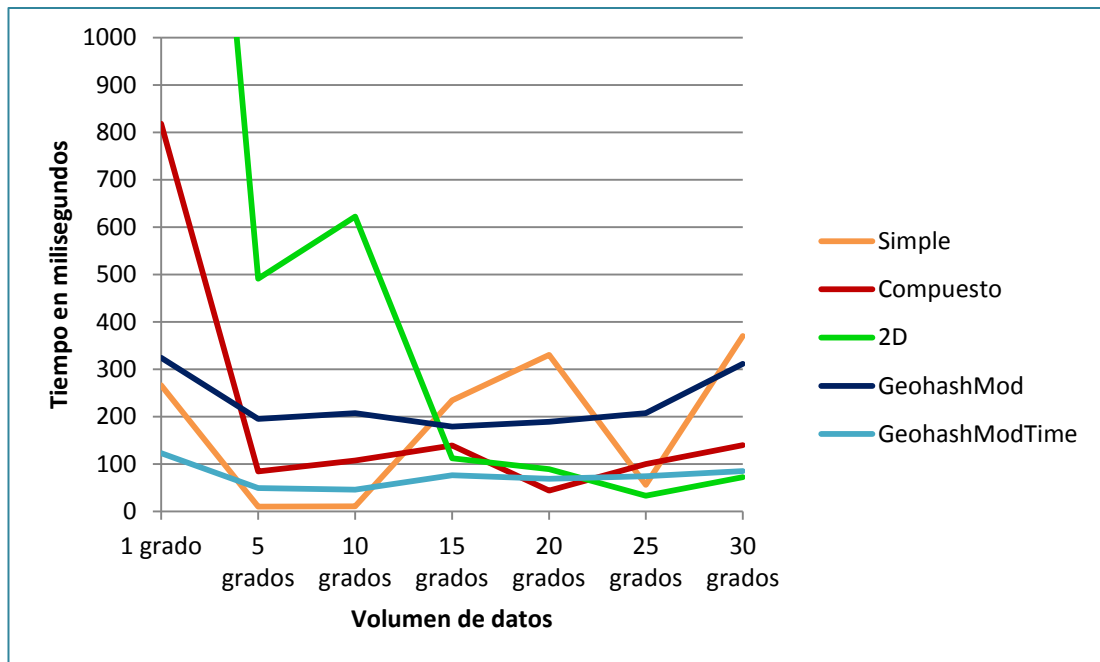


Figura 14: Comparación de los tiempos de respuesta de las consultas espaciales de los datos QuikSCAT almacenados en MongoDB para todos los tipos de indexación seleccionados.

También comprobamos en la gráfica de la figura 14 que en prácticamente todos los resultados hay una gran diferencia entre la primera consulta y las sucesivas. ¿Por qué ocurre esto? Se debe a que MongoDB utiliza la memoria interna para guardar las consultas que se han realizado recientemente (*buffering*), utilizando por defecto la mitad de la memoria para ello. Ello agiliza mucho las consultas que son iguales o muy similares sobre una base de datos y colección.

4.2.3.- Consultas espacio-temporales.

Para el caso de las consultas espacio-temporales, la figura 15 de nuevo muestra que el índice 2D tiene problemas para consultas de regiones pequeñas de datos geoespaciales, por lo que el empleo de los índices GeohashMod y GeohashModTime es la elección más eficiente para estos casos.

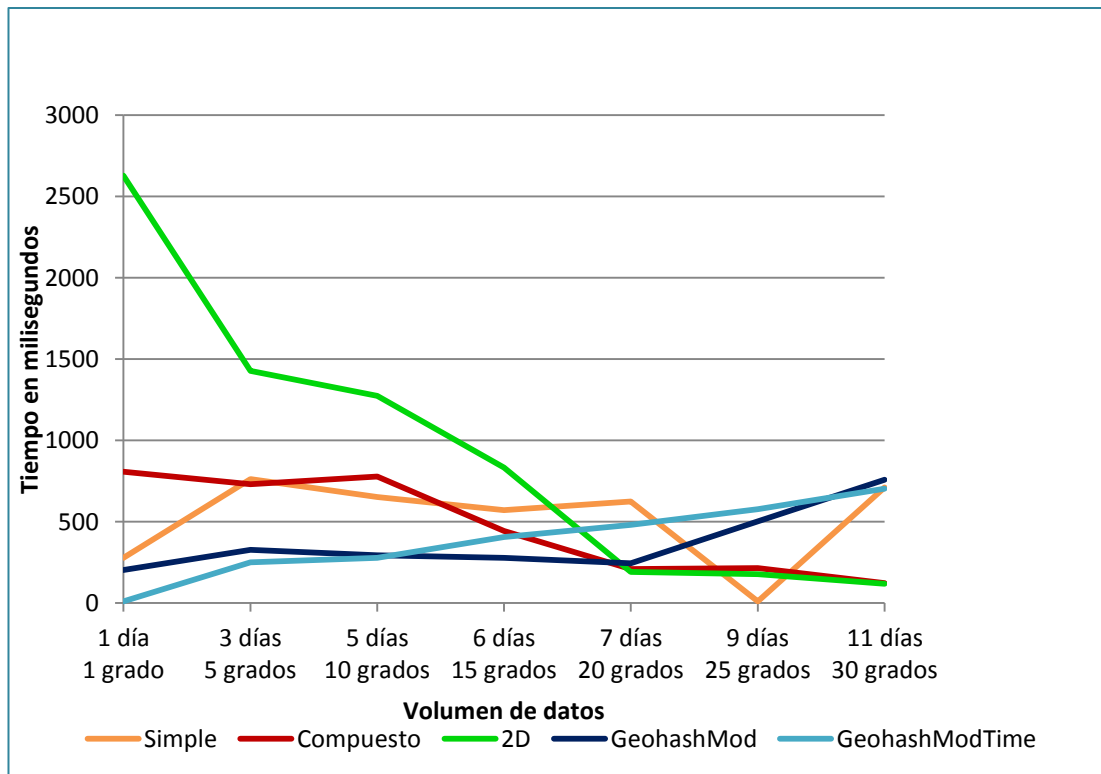


Figura 15: Comparación de los tiempos de respuesta de las consultas espacio-temporales de los datos QuikSCAT almacenados en MongoDB para todos los tipos de indexación seleccionados.

Al aproximarse dichos índices hacia consultas más densas, empiezan a requerir más tiempo de respuesta, siendo los índices 2D y Compuesto los más rápidos.

El índice Compuesto, además, tiene un comportamiento bastante eficiente para las consultas espacio-temporales, aunque no sea el mejor. Sin lugar a dudas esta serie de resultados parecen indicar que no hay un índice específico al que podamos erigir como el más eficiente para cualquier tipo de consulta sobre el tipo de datos.

4.3.- Datos RapidSCAT.

4.3.1.- Consultas temporales.

Al igual que para el caso de los datos QuikSCAT, no se ha podido realizar sobre el índice GeohashModTime debido a los límites máximos de documento establecidos por MongoDB.

En este caso el comportamiento de los índices es bastante similar al caso de los datos QuikSCAT: el índice Compuesto y el 2D siguen más o menos la misma línea, mientras que el Simple se despega de ellos al final.

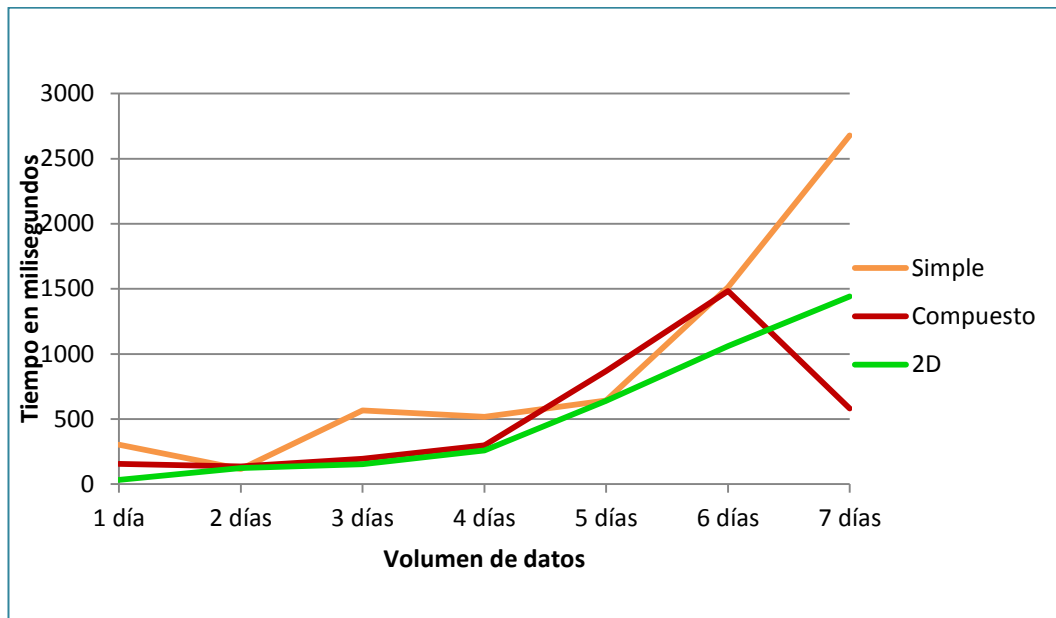


Figura 16: Comparación de los tiempos de respuesta de las consultas temporales de los datos RapidSCAT almacenados en MongoDB utilizando los índices Simple, Compuesto y 2D.

En la gráfica de la figura 16 se refleja la densidad de los datos comparados con los de QuikSCAT, ya que todas las líneas superan el tiempo de respuesta de las consultas para ese caso. De nuevo, en la gráfica no se representa al índice GeohashMod, el motivo es el mismo al del apartado de datos QuikSCAT.

Debido a como está construido el índice GeohashModTime, imposibilita la realización de estas consultas, porque el tamaño de la consulta necesaria para conseguir este volumen de datos es superior a la que ofrece MongoDB como máximo. En el caso de GeohashMod, es posible realizarlas ya que no excede dicho límite, sin embargo la diferencia es tan abrumadora que hace inútil esta comparación.

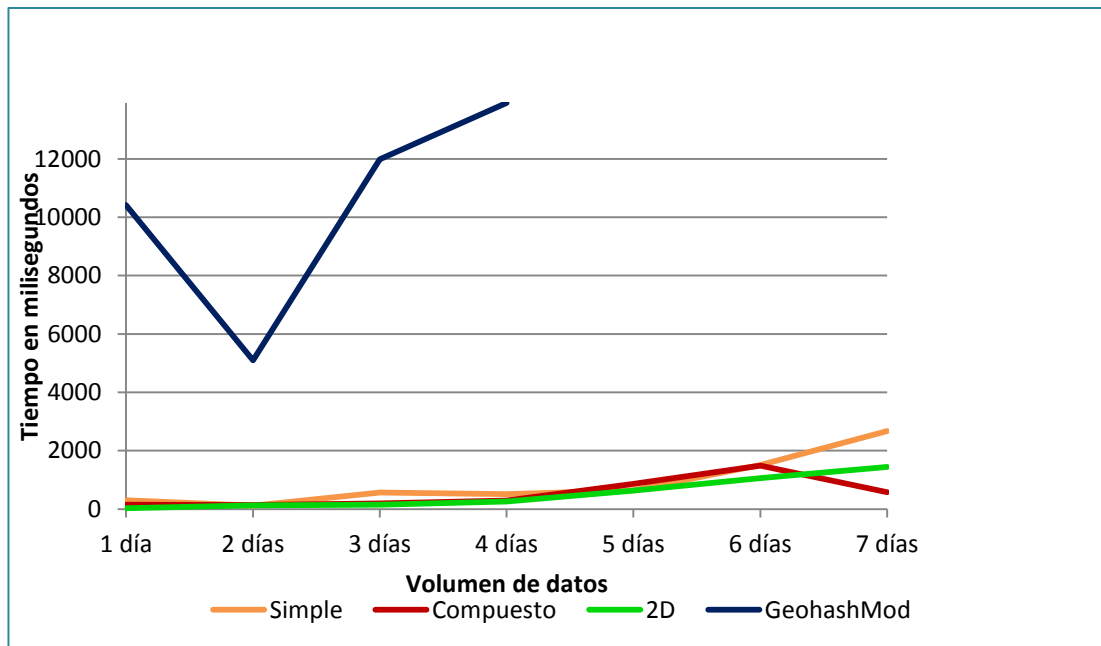


Figura 17: Comparación de los tiempos de respuesta de las consultas temporales de los datos RapidSCAT almacenados en MongoDB utilizando los índices Simple, Compuesto, 2D y GeohashMod, y con una reducción de los intervalos de consulta en 10°.

4.3.2.- Consultas espaciales.

Para las consultas espaciales de regiones pequeñas, el índice 2D vuelve a comportarse de forma muy poco eficiente. El índice 2D llega a alcanzar 19 segundos y 334 milisegundos en las consultas de 1° cuadrado de latitud y longitud, 8 segundos y 104 milisegundos en la de 5°, y 7 segundos y 839 milisegundos en las de 10°, haciéndolo terriblemente ineficiente para zonas de menos de 10° cuadrados de rango. Además, las respuestas vuelven a tener un gran retraso tras superar los 20° cuadrados de rango o ventana de consulta.

Los índices Simple y Compuesto, también tienen un comportamiento ineficiente, llegando a tener un tiempo de respuesta superior a 2 segundos para la primera consulta. Y al contrario, los índices GeohashMod y GeohashModTime tienen un comportamiento, en cuanto a tiempo de respuesta, bastante parecido siendo GeohashMod algo más ineficiente al inicio pero aun así, siendo mejor que cualquiera de los índices que no utilizan GeohashMod.

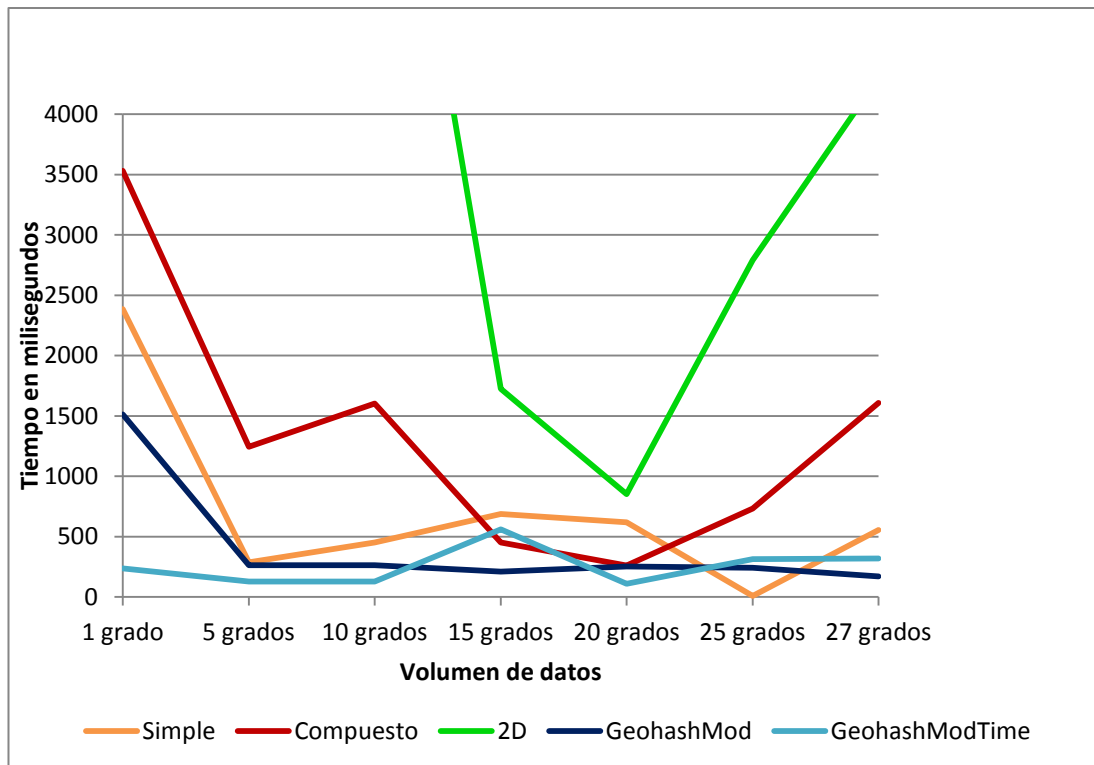


Figura 18: Comparación de los tiempos de respuesta de las consultas espaciales de los datos RapidSCAT almacenados en MongoDB para todos los tipos de indexación seleccionados.

4.2.3.- Consultas espacio-temporales.

Por ultimo para este tipo de consulta, el índice 2D vuelve a sufrir en el tiempo de respuesta, siendo el más alto de todos para todas las consultas realizadas para este caso.

Viéndolo más en detalle en la figura 20 podemos ver que son de nuevo los índices GeohashMod y GeohashModTime los que resultan más rápido a la hora de otorgar su respuesta.

El índice Compuesto se dispara hacia la mitad de la gráfica volviendo a los límites inferiores al final de esta, mientras que el Simple alcanza los tiempos de los índices GeohashMod y GeohashModTime rápidamente después de la primera consulta.

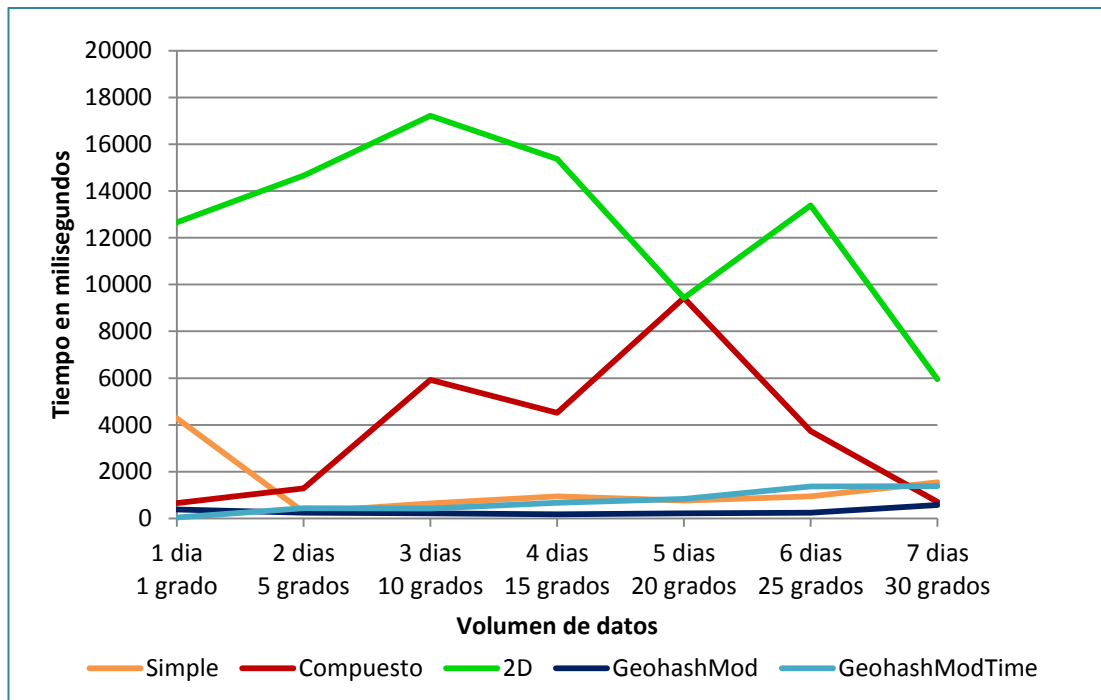


Figura 19: Comparación de los tiempos de respuesta de las consultas espacio-temporales de los datos RapidSCAT almacenados en MongoDB para todos los tipos de indexación seleccionados.

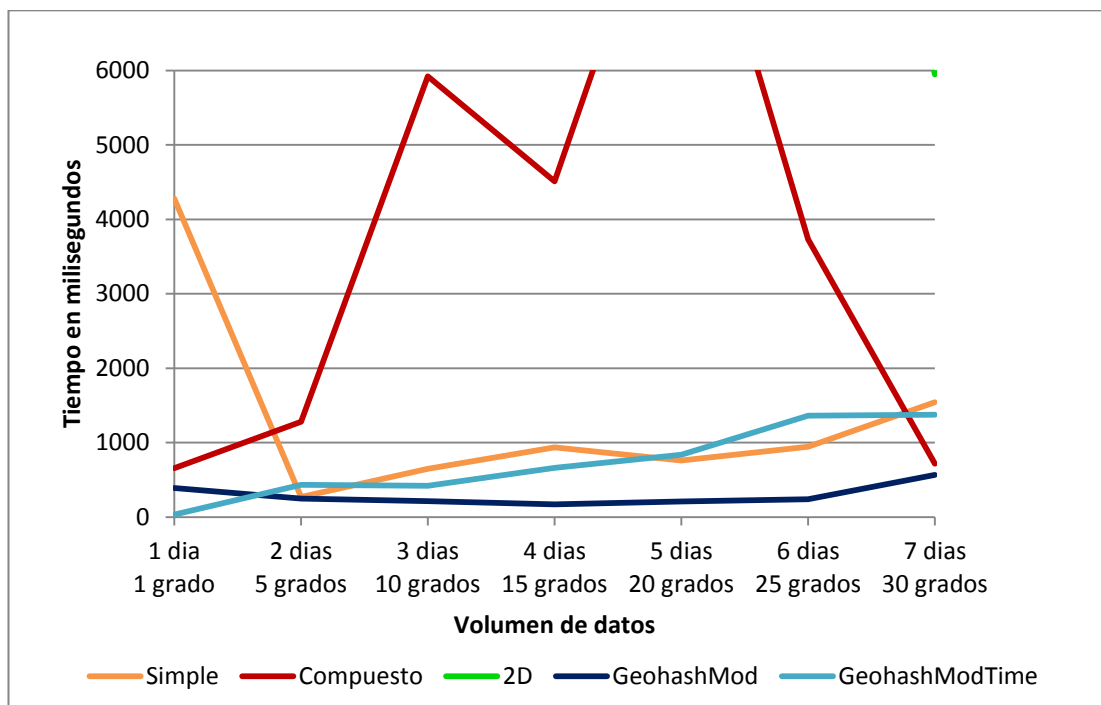


Figura 20: Comparación de los tiempos de respuesta de las consultas espacio-temporales de los datos RapidSCAT almacenados en MongoDB para todos los tipos de indexación seleccionados., en detalle.

4.4.- Datos QuikSCAT y RapidSCAT.

Estas graficas son la unión de las 2 anteriores, con la intención de comparar diferencias entre los tipos de datos.

4.4.1.- Consultas temporales.

Como ya se vio en los apartados anteriores, los índices GeohashMod y GeohashModTime son terriblemente ineficientes en este ámbito siendo incluso imposible realizar consultas (de más de un día) para el índice GeohashModTime. Además, la inclusión de demasiadas líneas en las gráficas puede dar lugar a confusión, por lo que las líneas del índice GeohashMod que se vieron en apartados anteriores para este tipo de consultas, no serán incluidas en estas gráficas.

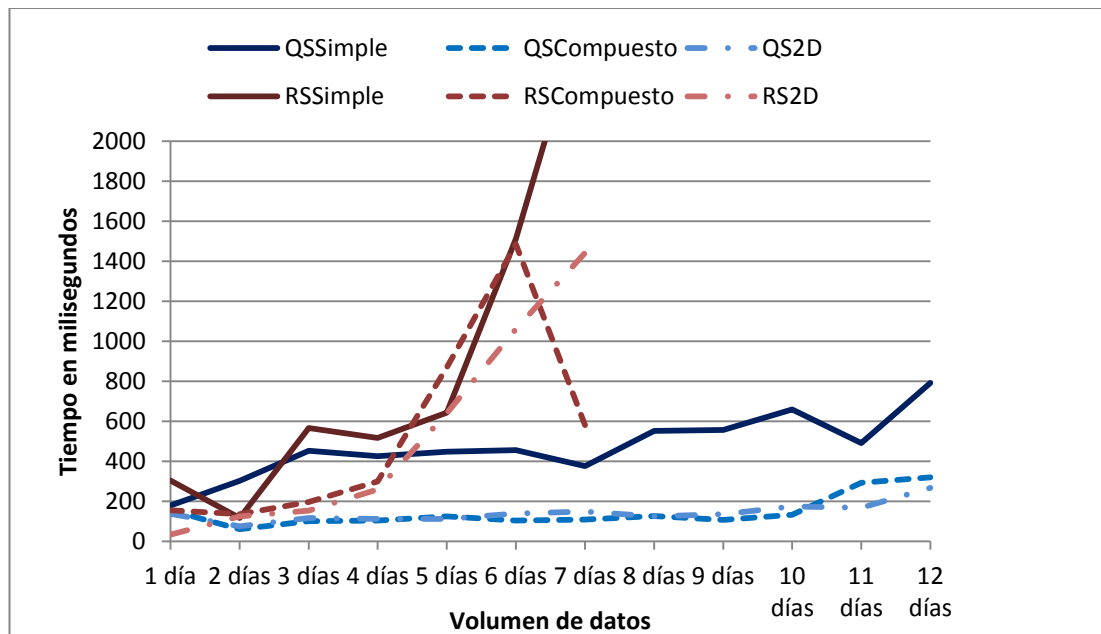


Figura 21: Comparación de los tiempos de respuesta de las consultas temporales de los datos QuikSCAT y RapidSCAT almacenados en MongoDB utilizando los índices Simple, Compuesto y 2D.

Podemos observar que la densidad de datos hace mella en el tiempo de respuesta haciendo que los índices sobre los datos RapidSCAT sufran al intentar responder sobre las consultas. Sin embargo, para las consultas de menor volumen de datos, la respuesta es muy parecida entre los tipos de datos.

4.4.2.- Consultas espaciales.

En la siguiente grafica se mostrarán los índices Simple, Compuesto y 2D de ambos tipos de datos y posteriormente veremos otra gráfica con los índices GeohashMod y GeohashModTime. Se ha decidido emparejarlos de esta forma, primero, porque todos en una única gráfica forman un galimatías ilegible, y segundo porque la principal utilidad de estas gráficas es la comparación entre los tipos de índice.

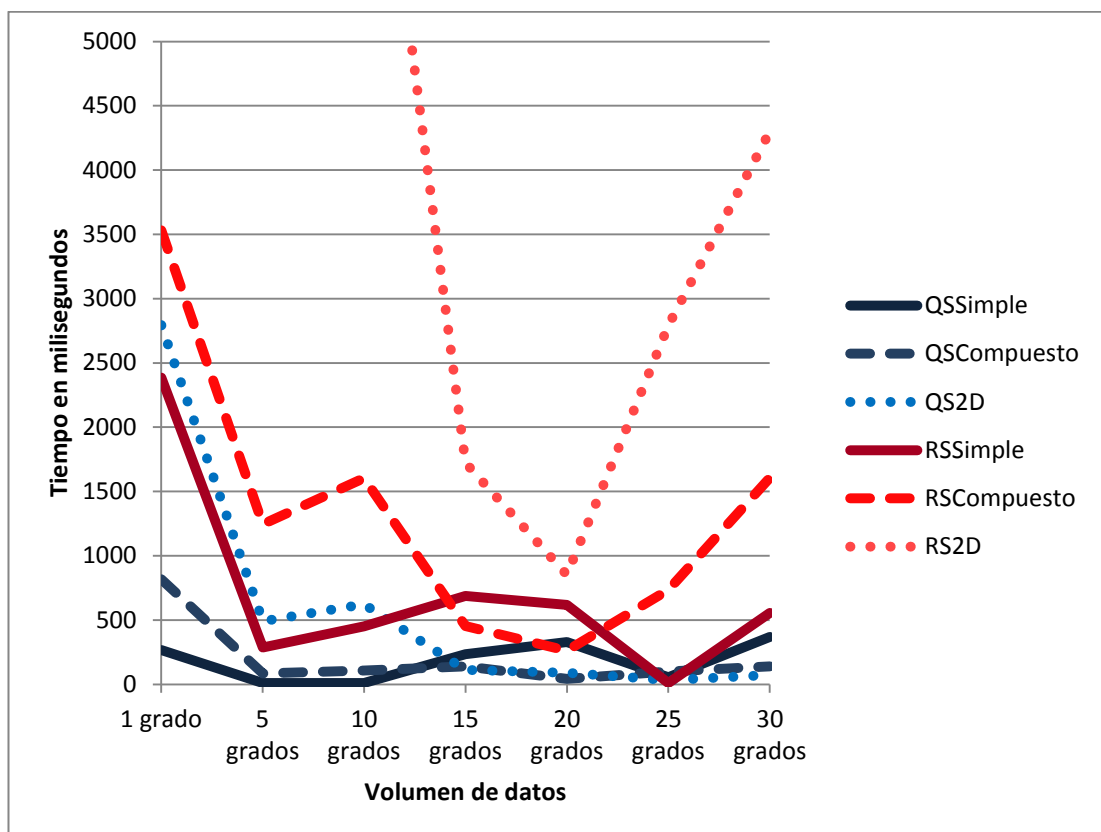


Figura 22: Comparación de los tiempos de respuesta de las consultas espaciales de los datos QuikSCAT y RapidSCAT almacenados en MongoDB utilizando los índices Simple, Compuesto y 2D.

De nuevo podemos vislumbrar la diferencia de tiempos por la densidad de los datos. El índice 2D comparte un gran retraso al principio en ambos tipos de datos, siendo incluso el de los datos QuikSCAT bastante parecido al inicio del índice Simple para datos RapidSCAT en cuanto a tiempo de respuesta y posteriormente realizando una gran bajada en este tiempo.

Los índices Simple y Compuesto comparten un comportamiento parecido para ambos tipos de datos, simplemente la diferencia se debe, de nuevo, a la densidad de los datos.

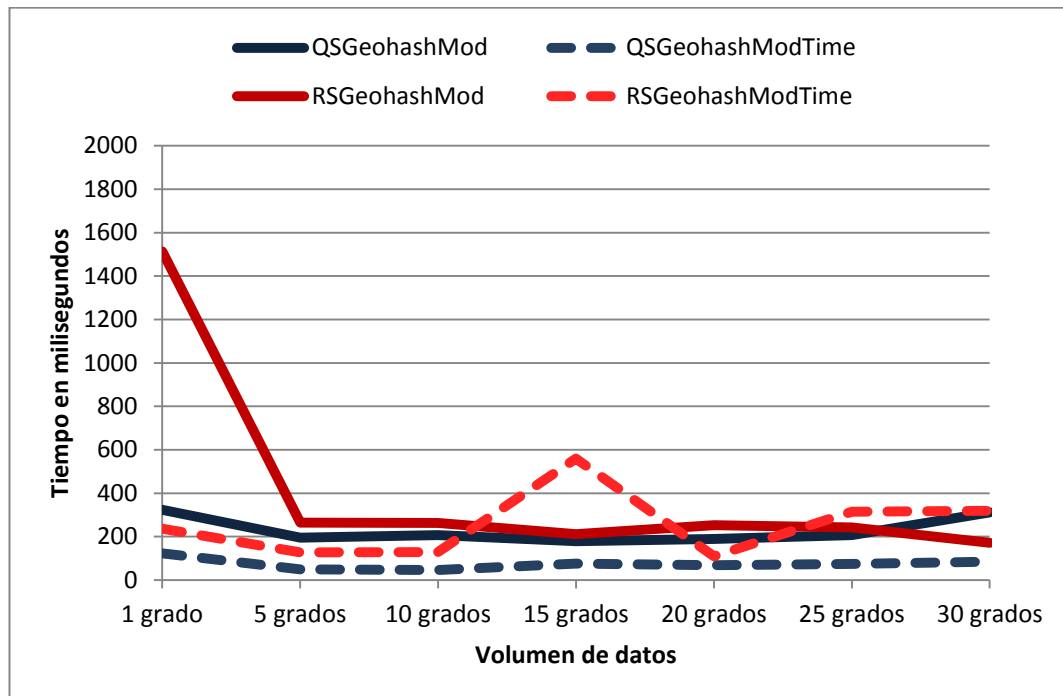


Figura 23: Comparación de los tiempos de respuesta de las consultas espaciales de los datos QuikSCAT y RapidSCAT almacenados en MongoDB utilizando los índices GeohashMod y GeohashModTime.

En este caso el comportamiento de los índices es claramente muy parecido entre sí, teniendo pocas diferencias incluso entre los dos tipos de datos, a excepción de algunos puntos. Según parece, a este par de índices la densidad de los datos les afecta levemente comparado con los índices Simple, Compuesto o 2D.

4.4.3.- Consulta espacio-temporales.

Para este caso ha sido necesario de nuevo, realizar una separación de las líneas en las figuras a fin de poder leerlas de una forma más sencilla. Debido a que para este tipo de consulta, a los datos QuikSCAT se les aplicó intervalos distintos a los de los datos RapidSCAT por su densidad, se representan de forma que la primera línea de los puntos señalados en el eje horizontal de la gráfica representa el intervalo de días para los datos

QuikSCAT, la segunda línea son los intervalos de días para los datos RapidSCAT y la última representa los grados tomados en cada caso.

Por último, señalar que la comparación podría haberse acertado más la gráfica haciendo coincidir los días de forma simétrica. Sin embargo, he procedido así porque creo que es la forma más correcta de representarlos ya que, aun que las consultas QuikSCAT son más amplias, el volumen de los datos es siempre inferior al de las consultas RapidSCAT.

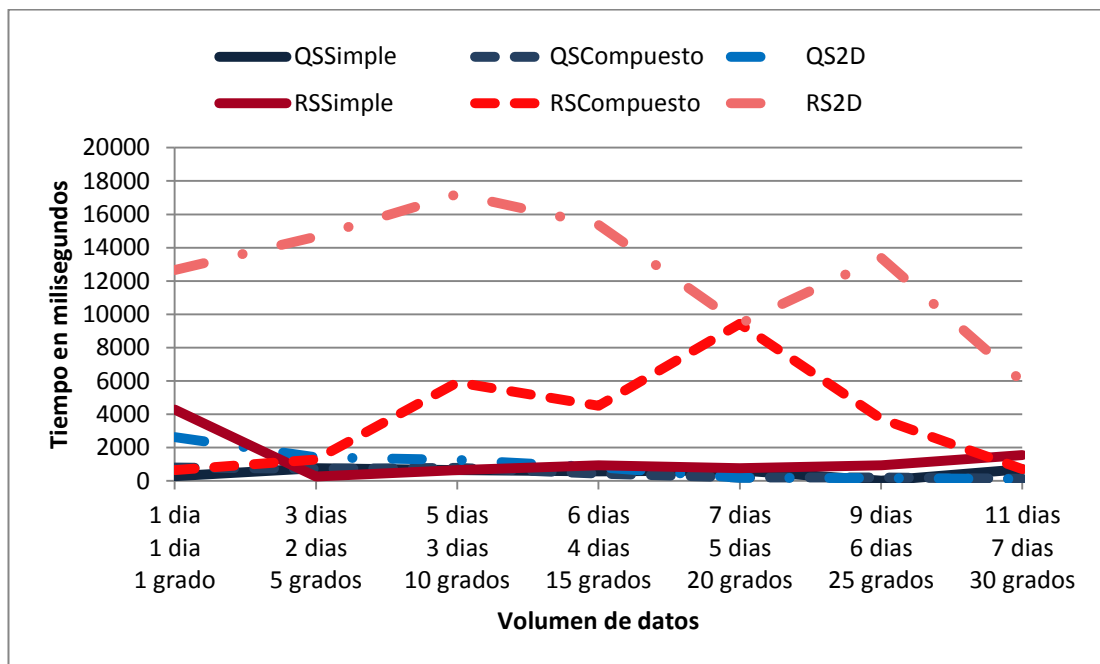


Figura 24: Comparación de los tiempos de respuesta de las consultas espacio-temporales de los datos QuikSCAT y RapidSCAT almacenados en MongoDB utilizando los índices Simple, Compuesto y 2D.

Esta es la gráfica completa con los tres índices, Simple, Compuesto y 2D. Como la visión de esta se hace bastante confusa, en la siguiente podemos ver con más detalle, las líneas de los índices a excepción de la línea de RS2D.

Como podemos comprobar en la figura 25, el único índice que para la mayoría de las consultas tiene unos tiempos de respuesta parecidos a las líneas de los índices para los datos QuikSCAT, es el índice Simple de RapidSCAT.

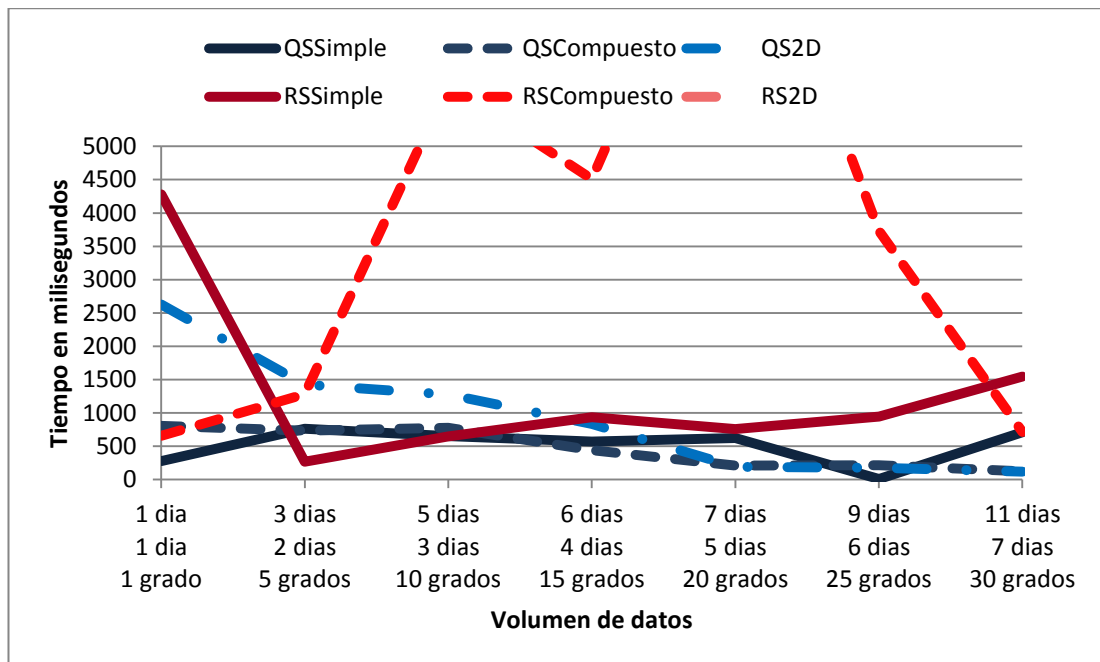


Figura 25: Comparación de los tiempos de respuesta de las consultas espacio-temporales de los datos QuikSCAT y RapidSCAT almacenados en MongoDB utilizando los índices Simple, Compuesto y 2D, en detalle.

Sin embargo, todos los índices sobre los datos RapidSCAT, incluido el RS2D que no se representa en esta última gráfica, tienen unos tiempos de respuesta más bajos en las últimas consultas, algo bastante extraño ya que estas son más pesadas, en cuanto a cantidad de datos.

Por último, en los índices GeohashMod y GeohashModTime, en este caso, sí que parece afectarles el aumento de la cantidad de datos recogida. Comparando esta gráfica con la figura 23, podemos llegar a la conclusión de que los cambios de colección y fecha, afectan a los tiempos de respuesta negativamente, esto probablemente se deba a la cache de MongoDB que hablamos de ella en el apartado 4.2.2.

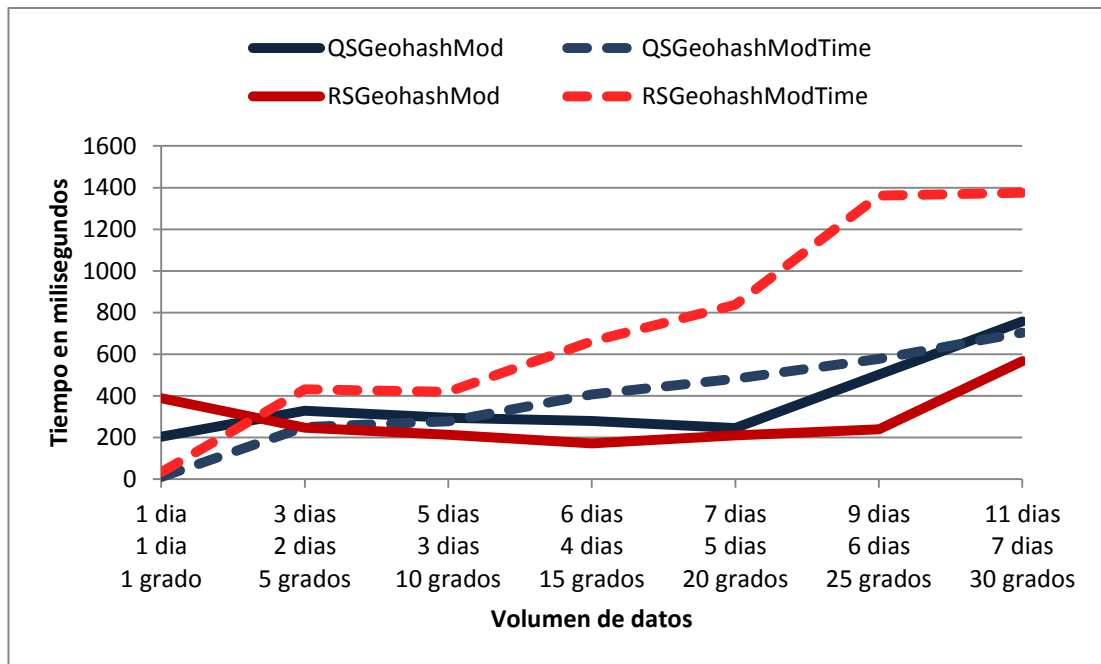


Figura 26: Comparación de los tiempos de respuesta de las consultas espacio-temporales de los datos QuikSCAT y RapidSCAT almacenados en MongoDB utilizando los índices GeohashMod y GeohashModTime.

5.- Conclusiones y Trabajo Futuro.

5.1.- Conclusiones según los resultados obtenidos en el proyecto.

Las conclusiones sobre el análisis de los tiempos de respuesta para cada tipo de índice es, que el mejor índice depende significativamente del tipo de aplicación de los datos.

El índice Simple parece tener un comportamiento bastante errático, probablemente debido a como está construido, ya que para cualquier tipo de consulta (por pequeña que sea) se realizará una consulta de prácticamente, el planeta al completo, que posteriormente se filtrará para dar el resultado que se había pedido en un principio. Esto, por supuesto, ralentiza muchísimo la obtención de los datos y no está reflejado en las gráficas.

El índice Compuesto tiene un comportamiento bastante “normal”. Simplemente realiza bien su trabajo, sin ser necesariamente el más rápido, pero tampoco el más lento en muchos casos.

El índice 2D tiene grandes problemas para adquirir datos de zonas pequeñas del globo terrestre, mientras que tiene un comportamiento bastante adecuado a la hora de realizar consultas de todo el planeta para diferentes días.

El índice GeohashMod presenta grandes problemas en las consultas sobre todo el espacio (el planeta completo), teniendo graves repercusiones sobre el tiempo de respuesta. La causa primordial es la forma en la que se ha construido. Sin embargo, comparte junto al índice GeohashModTime muy buenos tiempos de respuesta a la hora de realizar consultas sobre regiones específicas de la Tierra.

Y por último, el GeohashModTime tiene el grave problema de no poder realizar consultas demasiado amplias, pues rápidamente sobrepasa el buffer máximo de consulta. Pero para consultas pequeñas o medianas tiene un comportamiento muy bueno.

Por todo ello, específicamente, ninguno de los índices presentados realmente puede adquirir la categoría del más eficiente para cualquier tipo de consulta: todos los índices están muy ligados al tipo de consulta que se realiza. Por ejemplo, si deseamos realizar consultas de regiones muy pequeñas, podemos utilizar los índices específicamente creados en ese proyecto, GeoHashMod y GeoHashTime, o similares; mientras que para regiones grandes de la Tierra, los índices 2D y Compuesto son una mejor opción.

Otra opción es, aprovechando que las consultas se han realizado sobre una base de datos NoSQL que tolera la partición y suponiendo que se tiene una gran cantidad de espacio para almacenar los datos, crear una plataforma que tras leer los parámetros de tiempo, latitud y longitud, envíe la consulta a una base de datos con un índice específico que se muestre claramente más eficiente que los demás para ese tipo de consulta. De esta forma mantendríamos la mayor eficiencia independientemente del tipo de consulta realizada.

5.2.- Conclusiones personales.

Este Trabajo Final de Grado ha sido muy provechoso personalmente porque he aprendido a utilizar numerosas herramientas y tecnologías desconocidas para mí y a hacer un uso correcto de las mismas.

Se ha realizado un acercamiento a datos científicos, siendo necesario entenderlos bien para tratarlos de forma correcta haciendo uso de distintas librerías que posibilitarían la utilización de estos datos de una forma más sencilla.

Se ha realizado un gran acercamiento hacia la base de datos NoSQL MongoDB, que conocía por haber visto en otras asignaturas pero no había tratado en profundidad. Además de realizar un tratamiento para la base de datos con Java, que ha supuesto un incremento de mis aptitudes para este lenguaje y por supuesto, para la librería que maneja la conexión con MongoDB.

También se ha realizado un acercamiento a Python 2.7, un lenguaje totalmente nuevo para mí y que actualmente se encuentra en alza, ya que cada vez son más personas las que lo utilizan.

Se ha estudiado e incluso se ha implementado este sistema innovador de codificación de coordenadas geográficas llamado Geohash que agiliza el tratamiento de datos de carácter geográfico, creando un sistema paralelo que incluye la fecha para agilizar más aún las consultas para este tipo de datos.

Este último, probablemente es mejorable cambiando la forma en que se codifica el día o quizás incluyendo la codificación de la hora en el sistema, pero sin lugar a dudas ha sido muy provechoso este acercamiento.

5.3.- Trabajos futuros sugeridos.

Actualmente las búsquedas con datos con contenido geoespacial son muy frecuentes, ya que podemos encontrar diversas aplicaciones para este tipo de datos. Por lo que conseguir un acceso rápido a estos datos, bien para realizar Data Mining sobre ellos o para utilizarlos directamente con algún tipo de propósito.

Existen también, una gran cantidad de bases de datos NoSQL (Edlich, 2016) por lo que comprobar la eficiencia de índices geoespaciales en otras bases de datos NoSQL en comparación con las pruebas realizadas en este proyecto podría dar salida a resultados mejores pudiendo encontrar un sistema gestor de base de datos que actúe de la forma más eficiente posible ante este tipo de datos.

La implementación del índice Geohash modificado que se ha realizado en este proyecto, también es mejorable. Actualmente, existen técnicas que realizan las divisiones de forma no equiespacial para crear el hash de una zona en especial. Si estas divisiones las orientamos correctamente hacia los datos que hemos tratado en este proyecto, podemos conseguir que la eficiencia hacia la mayoría de las posibles consultas que se realicen sea aún mayor, a contrapunto de una minoría de consultas que serían más lentas.

6.- Anexo 1: Manual del programador.

En este apartado veremos, específicamente, el proceso que es necesario para poder ejecutar el programa en nuestro ordenador.

Necesitaremos instalar Java para poder ejecutar la aplicación de transformación de datos QuikSCAT (para lo que también necesitaremos instalar las librerías de HDF4 para esta funcionalidad), carga de datos a MongoDB y consulta de datos a MongoDB (necesitaremos MongoDB para este par de funcionalidades).

6.1 Java.

Es necesario tener instalado en nuestra maquina al menos el paquete de Java Runtime Environment (Techopedia Inc, 2016) con el que ejecutar la aplicación Java. Para este proyecto, se ha utilizado la versión 8 de dicho paquete de herramientas. Este paquete es bastante usual tenerlo ya previamente instalado, porque es necesario para ejecutar cualquier aplicación que necesite Java. En cualquier caso, es posible descargarlo de forma gratuita desde la página web de la organización. (Oracle, 2016)

Simplemente haciendo clic “Descarga gratuita de Java” como se puede ver en la imagen inferior.



Figura 27: Web de descarga de Java.

Tras esto, deberemos leer y aceptar el acuerdo de licencia de usuario final para poder descargar. Una vez descargado, lanzamos el ejecutable. Es necesario dar permisos de administrador de nuestro ordenador a este ejecutable, para poder instalarlo.

Una vez en el instalador, podremos seleccionar si queremos cambiar el directorio destino, posteriormente seleccionaremos “Instalar >”.

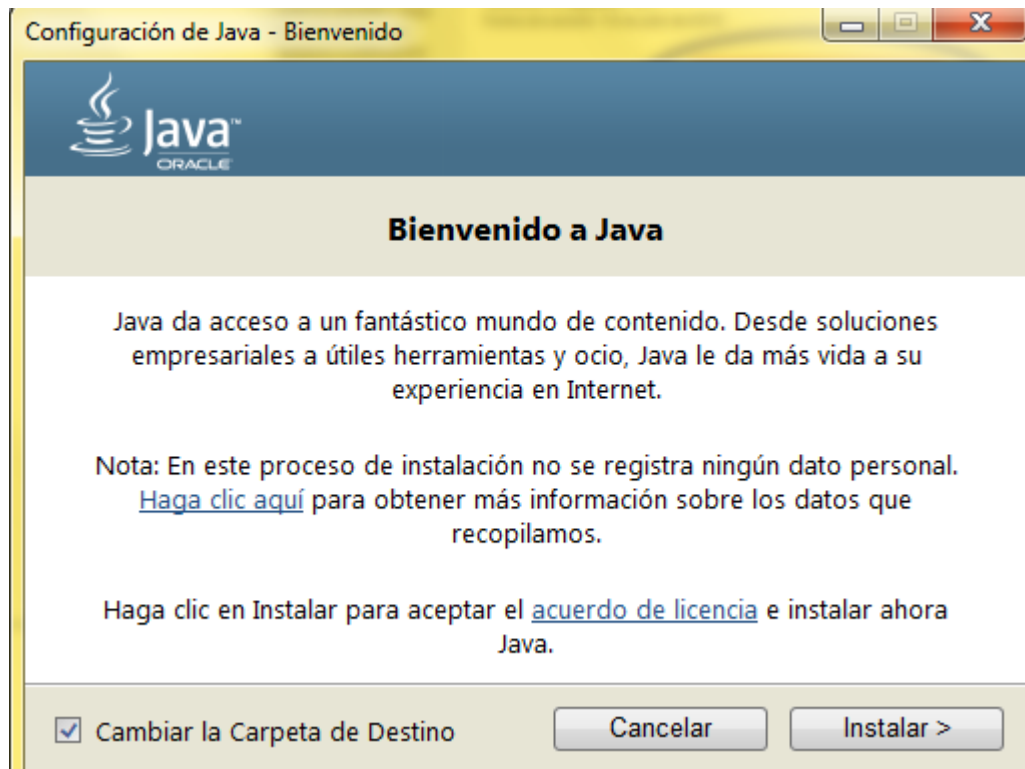


Figura 28: Programa de instalación de java, inicio.

En el siguiente menú, podremos seleccionar el directorio destino en caso de que así lo queramos. Después seleccionaremos “Siguiente >”.

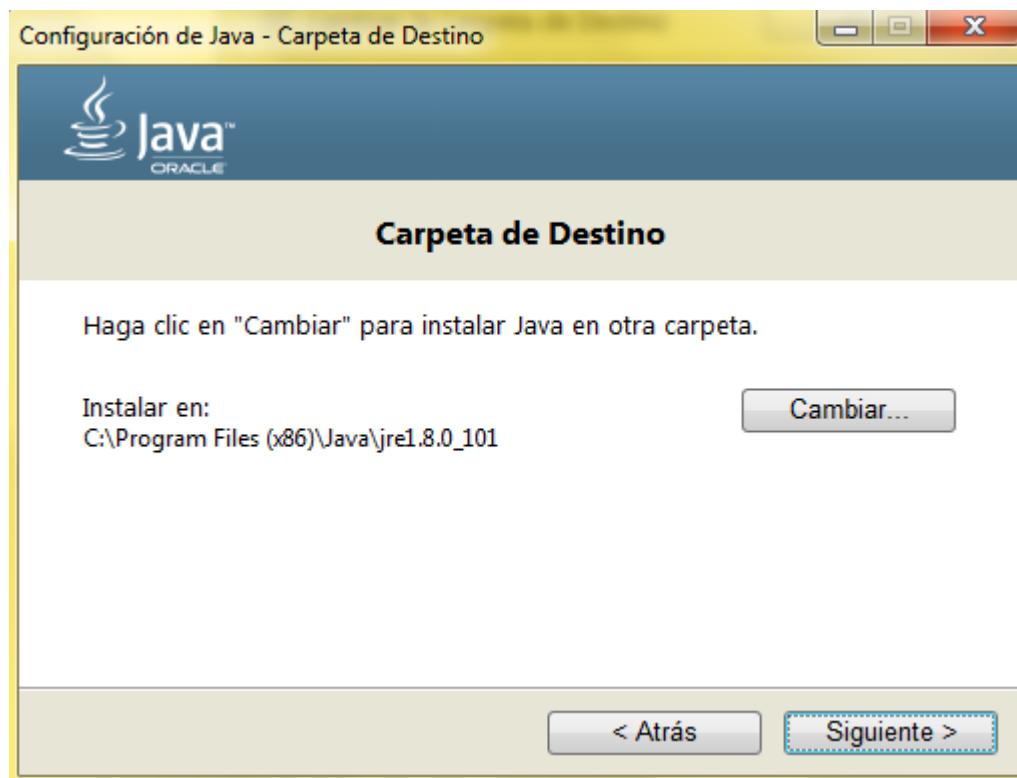


Figura 29: Programa de instalación de java, lugar de instalación.

Después de esto, nos preguntara si deseamos establecer Yahoo como página principal y motor de búsqueda para todos los navegadores predeterminados. Esto no es algo necesario para la instalación de Java, recomiendo desmarcar la casilla. Presionaremos en el botón "Siguiete >".

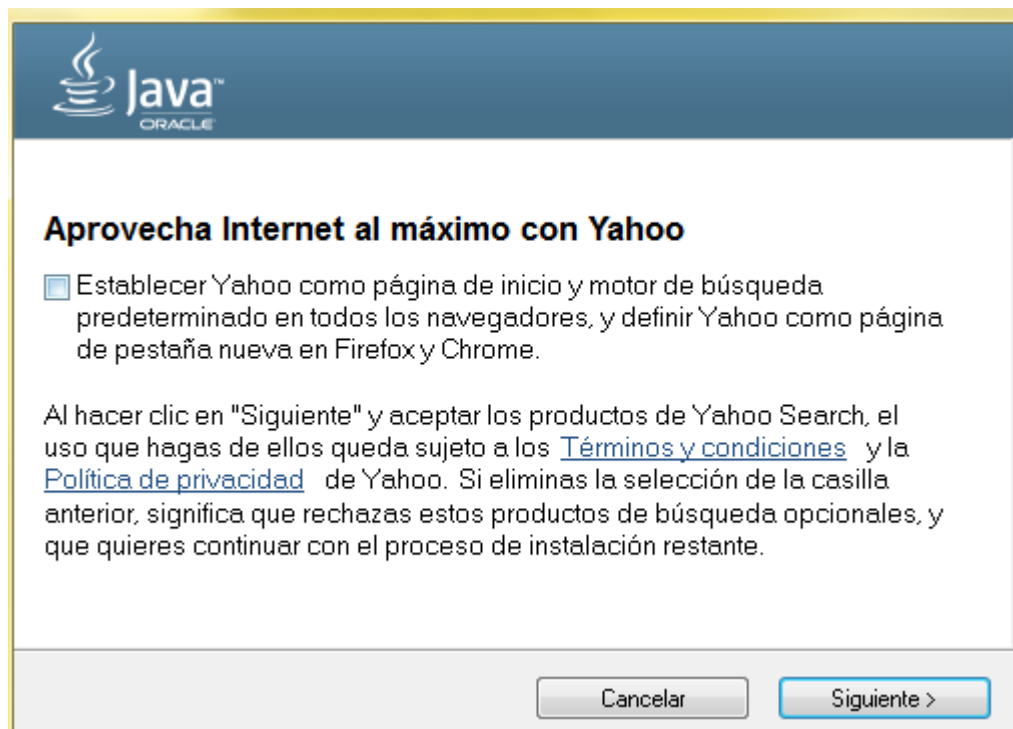


Figura 30: Programa de instalación de java, publicidad.

Posteriormente empezara la instalación. Una vez termine y se haya instalado correctamente, se nos abrirá el navegador para verificar la versión de Java y buscar nuevas actualizaciones. Con esto ya podremos ejecutar la aplicación del proyecto en Java.

6.2 HDF4.

El paquete de librerías HDF4, es posible descargarlo desde el sitio web de forma oficial y gratuita (The HDF Group, 2016). Simplemente desde el apartado de descarga, y seleccionando el botón "Download HDF4".

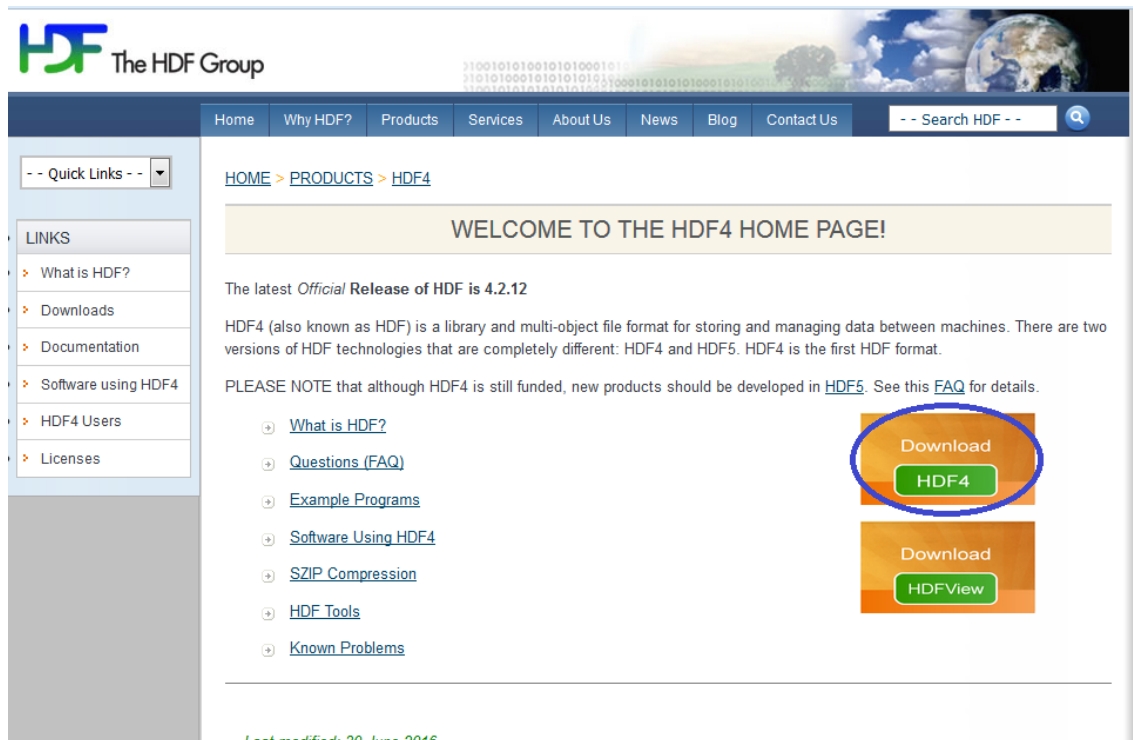


Figura 31: Web de descarga HDF4.

En la parte inferior de la siguiente pantalla se muestran las distribuciones binarias de este paquete. Descargamos la versión IVF 16 para Windows (para elegir entre 64 bits o 32 bits, debemos saber qué tipo de sistema operativo tenemos en nuestra maquina).

Platform	
Linux 3.10 CentOS 7 x86_64 Compilers: gcc & gfortran 4.8.5, jdk 1.7 or greater	Individual Utilities
Linux 2.6 CentOS 6 x86_64 Compilers: gcc & gfortran 4.4.7, jdk 1.7 or greater	Individual Utilities
Linux 2.6 CentOS 6 x86_64 (32-bit) Compilers: gcc & gfortran 4.4.7	Individual Utilities
Windows (64-bit) Compilers: CMake VS 2013 C++, IVF 15, jdk 1.7 or greater	
Windows (32-bit) Compilers: CMake VS 2013 C++, IVF 15, jdk 1.7 or greater	
Windows (64-bit) Compilers: CMake VS 2015 C++, IVF 16, jdk 1.7 or greater	
Windows (32-bit) Compilers: CMake VS 2015 C++, IVF 16, jdk 1.7 or greater	

Figura 32: Descarga de ejecutable de HDF4.

Una vez lo hayamos descargado, deberemos descomprimirlo y ejecutar el archivo con extensión "msi". La ejecución es como cualquier otro asistente de configuración, tendremos que leer un acuerdo de licencias y aceptarlo.

para poder instalarlo. Nos preguntará que paquetes queremos instalar y la ubicación de destino, en este caso, instalaremos todos los paquetes y la ubicación, podemos dejar el lugar por defecto o elegir otro y finalmente se iniciará la instalación.

A continuación, podremos utilizar la funcionalidad de transformación de datos QuikSCAT de la aplicación Java.

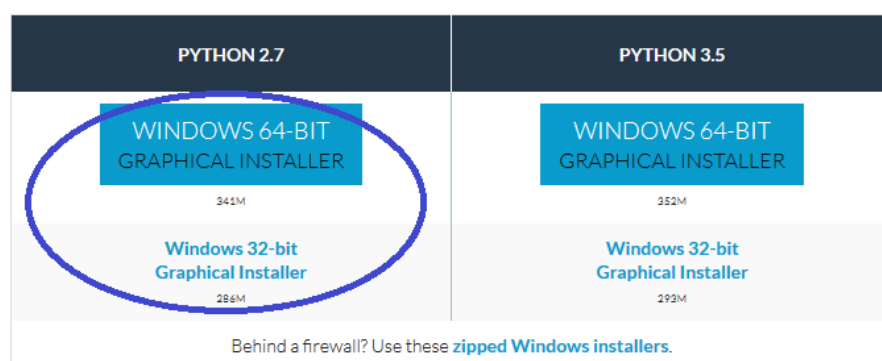
6.3.- Python, Numpy y netCDF4.

En este punto explicaremos como instalar Python, junto con los paquetes Numpy y netCDF4, para poder realizar la transformación de los datos RapidSCAT.

6.3.1 Python y Numpy.

La forma más rápida y sencilla de instalar Python y Numpy es instalando el programa Anaconda (Continuum Analytics, 2016), que instalara Python en nuestro ordenador, Numpy y algunos paquetes adicionales. Podemos descargar un instalador con asistente de configuración desde la página oficial, como se indica en la siguiente imagen, en 32 o 64 bits dependiendo de nuestro sistema operativo.

Anaconda for Windows



Windows Anaconda Installation

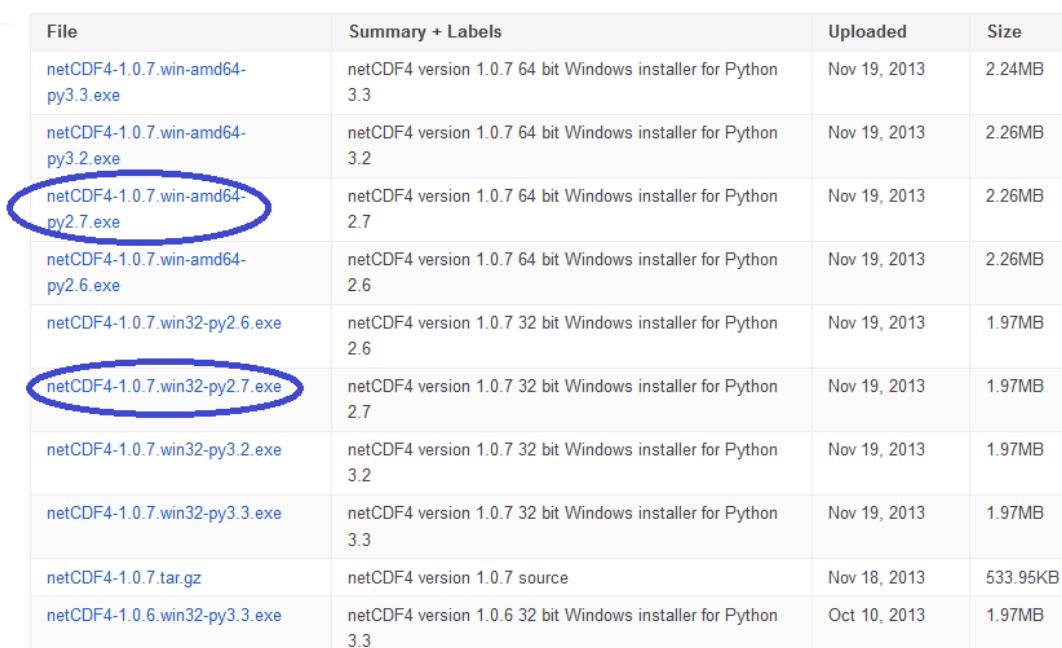
Figura 33: Descarga programa Anaconda.

Una vez descargado, habrá que ejecutarlo y seguir las instrucciones del asistente. Cuando lleguemos al punto del asistente que nos pregunta sobre “Advanced options” deberemos seleccionar ambos casilleros.

Una vez finalizada la instalación, ya lo tenemos todo preparado para instalar netCDF4.

6.3.2.- netCDF4.

Para instalar este paquete, será necesario haber realizado el paso anterior. Podemos descargar las librerías de netCDF4 desde la página oficial de Python (Python Software Foundation, 1990-2016). La versión 1.0.7 de netCDF4, incluye un ejecutable que facilita bastante el proceso de instalación. Debemos tener en cuenta que necesitamos el paquete para Python 2.7 además de descargar el indicado para nuestro sistema operativo.



File	Summary + Labels	Uploaded	Size
netCDF4-1.0.7.win-amd64-py3.3.exe	netCDF4 version 1.0.7 64 bit Windows installer for Python 3.3	Nov 19, 2013	2.24MB
netCDF4-1.0.7.win-amd64-py3.2.exe	netCDF4 version 1.0.7 64 bit Windows installer for Python 3.2	Nov 19, 2013	2.26MB
netCDF4-1.0.7.win-amd64-py2.7.exe	netCDF4 version 1.0.7 64 bit Windows installer for Python 2.7	Nov 19, 2013	2.26MB
netCDF4-1.0.7.win-amd64-py2.6.exe	netCDF4 version 1.0.7 64 bit Windows installer for Python 2.6	Nov 19, 2013	2.26MB
netCDF4-1.0.7.win32-py2.6.exe	netCDF4 version 1.0.7 32 bit Windows installer for Python 2.6	Nov 19, 2013	1.97MB
netCDF4-1.0.7.win32-py2.7.exe	netCDF4 version 1.0.7 32 bit Windows installer for Python 2.7	Nov 19, 2013	1.97MB
netCDF4-1.0.7.win32-py3.2.exe	netCDF4 version 1.0.7 32 bit Windows installer for Python 3.2	Nov 19, 2013	1.97MB
netCDF4-1.0.7.win32-py3.3.exe	netCDF4 version 1.0.7 32 bit Windows installer for Python 3.3	Nov 19, 2013	1.97MB
netCDF4-1.0.7.tar.gz	netCDF4 version 1.0.7 source	Nov 18, 2013	533.95KB
netCDF4-1.0.6.win32-py3.3.exe	netCDF4 version 1.0.6 32 bit Windows installer for Python 3.3	Oct 10, 2013	1.97MB

Figura 34: Descarga de netCDF4.

Finalizada la descarga, la instalación se realizara mediante un asistente de configuración. Llegado un punto de la instalación, nos pedirá la ubicación donde hemos instalado Python, que se encuentra en la en la misma ubicación que Anaconda. Por norma general lo encontrará automáticamente. Tras esto empezará la instalación.

Una vez finalizado, ya podremos utilizar la aplicación Python de transformación de datos RapidSCAT.

6.4.- MongoDB.

Necesitaremos MongoDB para almacenar los datos QuikSCAT y RapidSCAT y poder realizar las pruebas pertinentes para conocer la eficiencia de los índices.

Para instalar MongoDB, podemos descargarlo desde la página oficial. El ejecutable lanzará un asistente de configuración del que seguiremos sus instrucciones. Para poder cambiar la ubicación de la instalación, habrá que elegir el tipo de instalación "Custom".

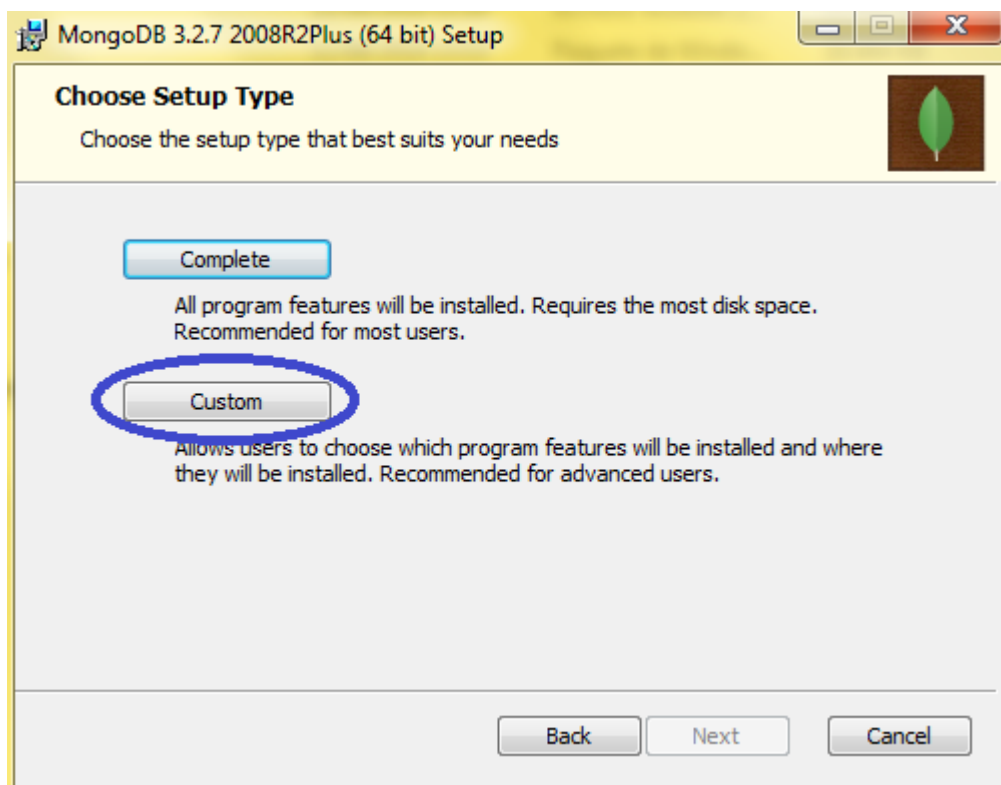


Figura 35: Instalación de MongoDB.

Una vez haya acabado el asistente de instalación, ya estará listo para poder usarse.

7.- Anexo 2: Manual de usuario.

En este anexo se explicarán las diversas funcionalidades del programa en cuestión, como ejecutarlo y los procesos necesarios para que sea posible el correcto funcionamiento.

Se supondrá que se han seguido los pasos explicados en el anexo 1 sobre la instalación de los programas necesarios para la ejecución correcta de la aplicación.

7.1.- MongoDB.

Para que la aplicación funcione correctamente, será necesario tener en todo momento el servidor de MongoDB activo, para que las llamadas de carga y consulta puedan obtener respuesta.

Para realizar esta acción, abriremos la consola de Windows, escribiendo **cmd** en el apartado “Buscar programas y archivos” del menú de inicio.

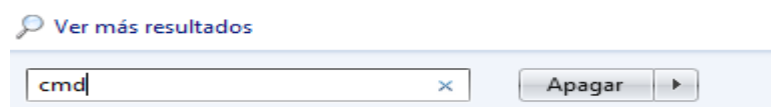
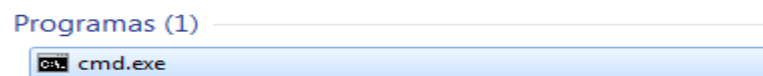


Figura 36: Ejecutar consola de Windows 7.

Tras esto se abrirá una consola desde la que podremos ejecutar diversos comandos. Para este caso, escribiremos “**mongod**” y pulsamos enter. No es necesario escribir la ubicación completa hacia este archivo, que se encuentra en el directorio “**bin**” de la ruta que elegimos durante su instalación, porque se encuentra en el path de Windows 7, que es una variable que almacena rutas específicas para una ejecución más sencilla de los servicios.

Es posible añadir la opción “**-dbpath ruta**” para especificar el lugar donde se guardaran los datos que se carguen en la base de datos.



```
Microsoft Windows [Versión 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. Reservados todos los derechos.

C:\Windows\system32>mongod -dbpath D:/datosMongo
```

Figura 37: Ejecución del servidor de MongoDB.

Posteriormente, cuando se muestre en pantalla el mensaje “waiting for connections on port 27017” ya tendremos listo el servidor para realizar las cargas y consultas necesarias.

```

2016-08-30T18:57:37.150+0200 I CONTROL [initandlisten] build environment:
2016-08-30T18:57:37.150+0200 I CONTROL [initandlisten] distmod: 2008plus
2016-08-30T18:57:37.165+0200 I CONTROL [initandlisten] distarch: x86_64
2016-08-30T18:57:37.165+0200 I CONTROL [initandlisten] target_arch: x86_64
2016-08-30T18:57:37.165+0200 I CONTROL [initandlisten] options: { storage: { db
Path: "D:\TFG\DatosMongo" } }
2016-08-30T18:57:37.165+0200 I - [initandlisten] Detected data files in D
:\TFG\DatosMongo created by the 'wiredTiger' storage engine, so setting the acti
ve storage engine to 'wiredTiger'.
2016-08-30T18:57:37.165+0200 W - [initandlisten] Detected unclean shutdown
n - D:\TFG\DatosMongo\mongod.lock is not empty.
2016-08-30T18:57:37.165+0200 W STORAGE [initandlisten] Recovering data from the
last clean checkpoint.
2016-08-30T18:57:37.165+0200 I STORAGE [initandlisten] wiredtiger_open config:
create,cache_size=4G,session_max=20000,eviction=(threads_max=4),config_base=false,statistics=(fast),log=(enabled=true,archive=true,path=journal,compressor=snappy),file_manager=(close_idle_time=100000),checkpoint=(wait=60,log_size=2GB),statistics_log=(wait=0).
2016-08-30T18:57:40.356+0200 I NETWORK [HostnameCanonicalizationWorker] Starting hostname canonicalization worker
2016-08-30T18:57:40.356+0200 I FTDC [initandlisten] Initializing full-time diagnostic data capture with directory 'D:\TFG\DatosMongo\diagnostic.data'
2016-08-30T18:57:40.366+0200 I NETWORK [initandlisten] waiting for connections on port 27017

```

Figura 38: Servidor MongoDB esperando por conexiones.

7.2.- Aplicación Python.

Para ejecutar la aplicación Python, es tan simple como ir al directorio en el que se encuentre la aplicación y ejecutar desde la consola “**python transformacion.py**”.

Tras unos segundos la aplicación se iniciará y nos preguntará que opción queremos elegir.

```

C:\Windows\system32\cmd.exe - python transformacion.py
D:\>cd Transformacion
D:\Transformacion>python transformacion.py
Listing librerias/ ...
Compiling librerias/\__init__.py ...
Compiling librerias/gzhandler.py ...
Compiling librerias/lectura_L2B.py ...
Compiling librerias/lectura_L2B12.py ...
Compiling librerias/readnc.py ...
Compiling librerias/recorrerdir.py ...
Compiling librerias/unirficheros.py ...

Bienvenido al programa de extraccion-modificacion de datos

=====
                        MENU
=====
0.- Salir
1.- Obtener datos L2B12 de Rapidscat
2.- Unir Ficheros

Introduzca su seleccion:

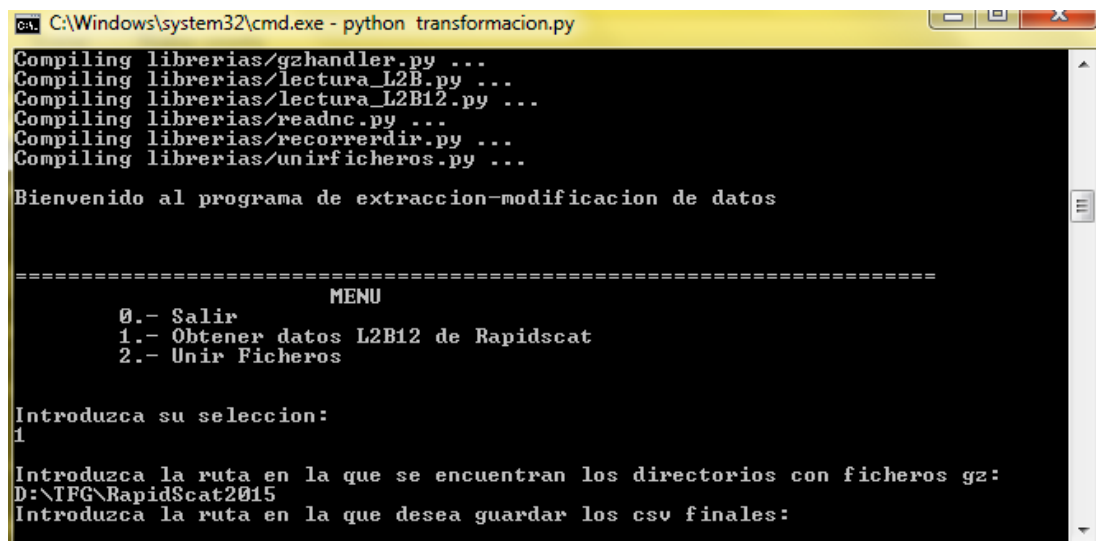
```

Figura 39: Ejecución de programa Python.

Las opciones a elegir son:

- 0. Salir: acabará la ejecución del programa.
- 1. Obtener datos L2B12 de RapidSCAT: transforma los datos del directorio proporcionado a formato CSV y posteriormente une dichos ficheros.
- 2. Unir Ficheros: une los ficheros del caso anterior por días. Esta opción solo es necesario utilizarla si se cortó este proceso durante la obtención de los datos en la selección 1.

Al seleccionar la opción 1, nos pedirá la ubicación de los ficheros “.gz” descargados desde el repositorio. Habrá que proporcionarle el directorio donde se ubican los directorio de los días y almacenan los archivos “.gz” (tal y como están colocados en el repositorio dentro de cada directorio de cada año). Después nos pedirá la ubicación donde queremos que estos ficheros se guarden una vez transformados y unidos.



```
C:\Windows\system32\cmd.exe - python transformacion.py
Compiling librerias/gzhandler.py ...
Compiling librerias/lectura_L2B.py ...
Compiling librerias/lectura_L2B12.py ...
Compiling librerias/readnc.py ...
Compiling librerias/recorrerdir.py ...
Compiling librerias/unirficheros.py ...

Bienvenido al programa de extraccion-modificacion de datos

=====
                        MENU
=====
0.- Salir
1.- Obtener datos L2B12 de Rapidscat
2.- Unir Ficheros

Introduzca su seleccion:
1

Introduzca la ruta en la que se encuentran los directorios con ficheros gz:
D:\TFG\RapidScat2015
Introduzca la ruta en la que desea guardar los csv finales:
```

Figura 40: Ejecución de programa Python, selección 1.

Si seleccionamos la opción 2, de nuevo nos pedirán las rutas de los ficheros comprimidos “.gz” así como la ruta final donde queremos que se guarden estos ficheros.

7.3.- Aplicación java.

Para la ejecución del programa java, será tan simple como hacer doble clic sobre el fichero “**Geoindex.jar**” desde el directorio en el que se encuentra la aplicación. IMPORTANTE: en el directorio donde se encuentra este ejecutable, también está la aplicación “**QSreadx64.exe**”. En caso de que se copiase la aplicación java, también sería necesario copiar QSreadx64 a la misma ubicación.

Veremos 3 pestañas diferenciadas:

1. **Transformación:** se seleccionan la ubicación de los ficheros descargados del repositorio, así como donde queremos que se guarden una vez transformados.
2. **Carga:** se proporciona la ubicación de los ficheros transformados para realizar la carga de los datos a la base de datos, así como la elección del índice que deseamos utilizar.
3. **Consulta:** se proporcionan los datos de consulta para realizar la extracción de los datos y se elige el índice en el que deseamos realizarla.

Hay una zona común para todas las pestañas, situada en la parte inferior. En este apartado podremos leer texto de información sobre el programa, como saber si la carga de datos ha terminado o la validez del dato que se está intentando introducir. También contiene un botón llamado “**Limpiar log**” para borrar el texto de este recuadro en caso de que fuese necesario.

7.3.1.- Transformación.

Una vez ejecutada la aplicación, la primera pantalla nos presenta la transformación de los datos QuikSCAT de su formato científico a CSV.

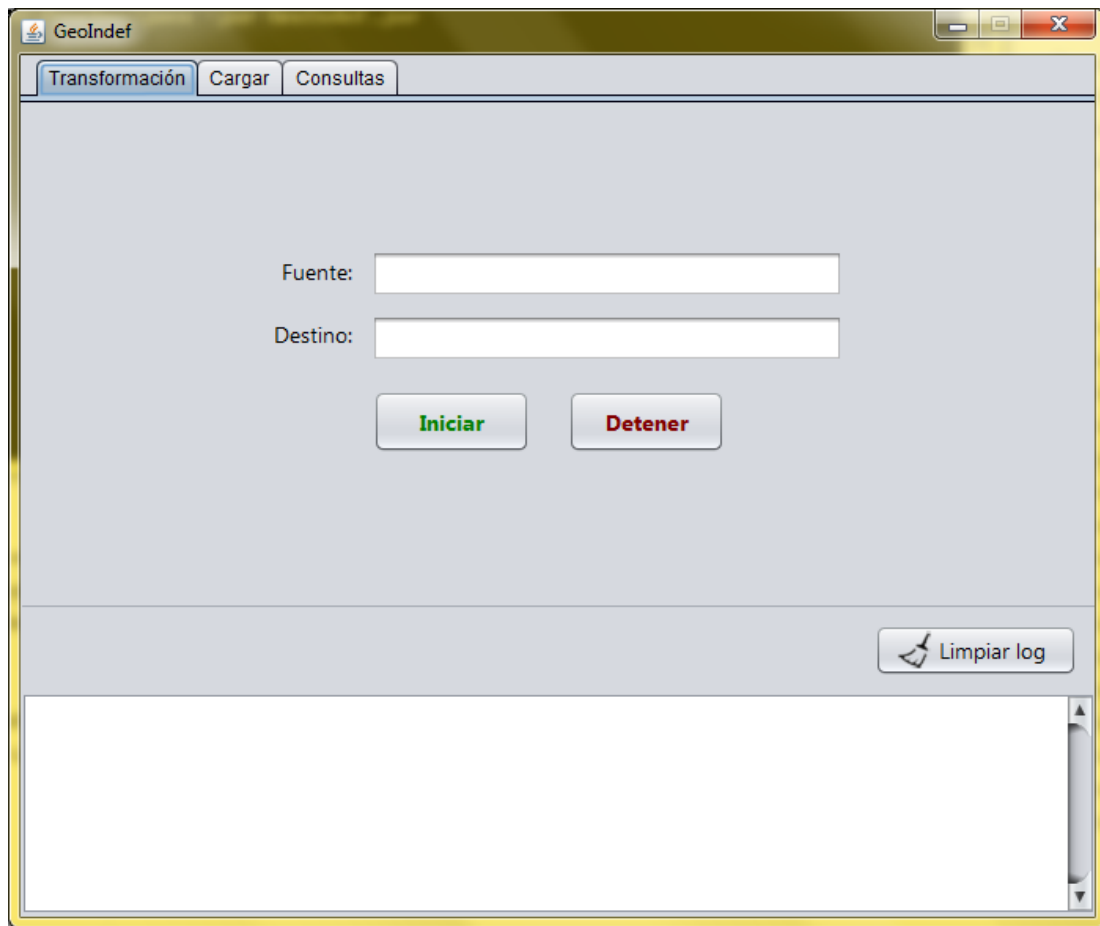


Figura 41: Pantalla de aplicación Java, Transformación.

Para transformar los datos, habrá que proporcionar como “**Fuente**”, la ubicación donde se encuentran los datos QuikSCAT tal y como los bajamos desde el repositorio. Es decir, hay que proporcionar el directorio del año que dentro contendrá un directorio para cada día y en el interior de cada uno los ficheros con los datos.

Como “**Destino**” pondremos la ubicación deseada. Hay que tener en cuenta que la aplicación creará un directorio llamado “**datos**” dentro de la ubicación destino, borrándolo si ya existía.

Tras esto, simplemente habrá que presionar el botón “**Iniciar**” y el proceso comenzará. Cuando la transformación de los datos haya terminado, en el recuadro inferior se mostrará un mensaje de finalización.

Con el botón “**Detener**” podemos detener el proceso de transformación en caso de que fuese necesario. Al volver a iniciarlo, la transformación comenzará desde el principio nuevamente.

7.3.2.- Cargar.

En la pantalla de carga se muestran 2 apartados distintos.

El primer apartado situado en la zona central, mostrará 5 botones de opción, que indican el tipo de índice que se utilizará para los datos que carguemos.

En la caja de texto con la etiqueta “**Fuente**”, debemos indicar la ubicación de los datos que hayamos elegido siendo el final de la ruta el directorio “**datos**”. Al transformar los datos QuikSCAT este directorio se genera automáticamente y es utilizado para guardar los datos, no ocurriendo lo mismo con los datos RapidSCAT, que el directorio “**datos**” no se genera automáticamente, teniendo que crear manualmente ese directorio.

En el segundo apartado situado en la parte inferior, veremos un recuadro con 2 botones de opción, donde elegiremos el tipo de datos a tratar, pudiendo ser RapidSCAT o QuikSCAT.

Posteriormente presionando el botón “**Cargar**” se iniciará la carga de los datos seleccionados a la base de datos.

En caso de que deseemos borrar una base de datos que ya hayamos cargado para liberar espacio, tendremos que seleccionar el tipo de índice correspondiente a esa base de datos y el tipo de dato y presionar el botón “**Limpiar BD**”.

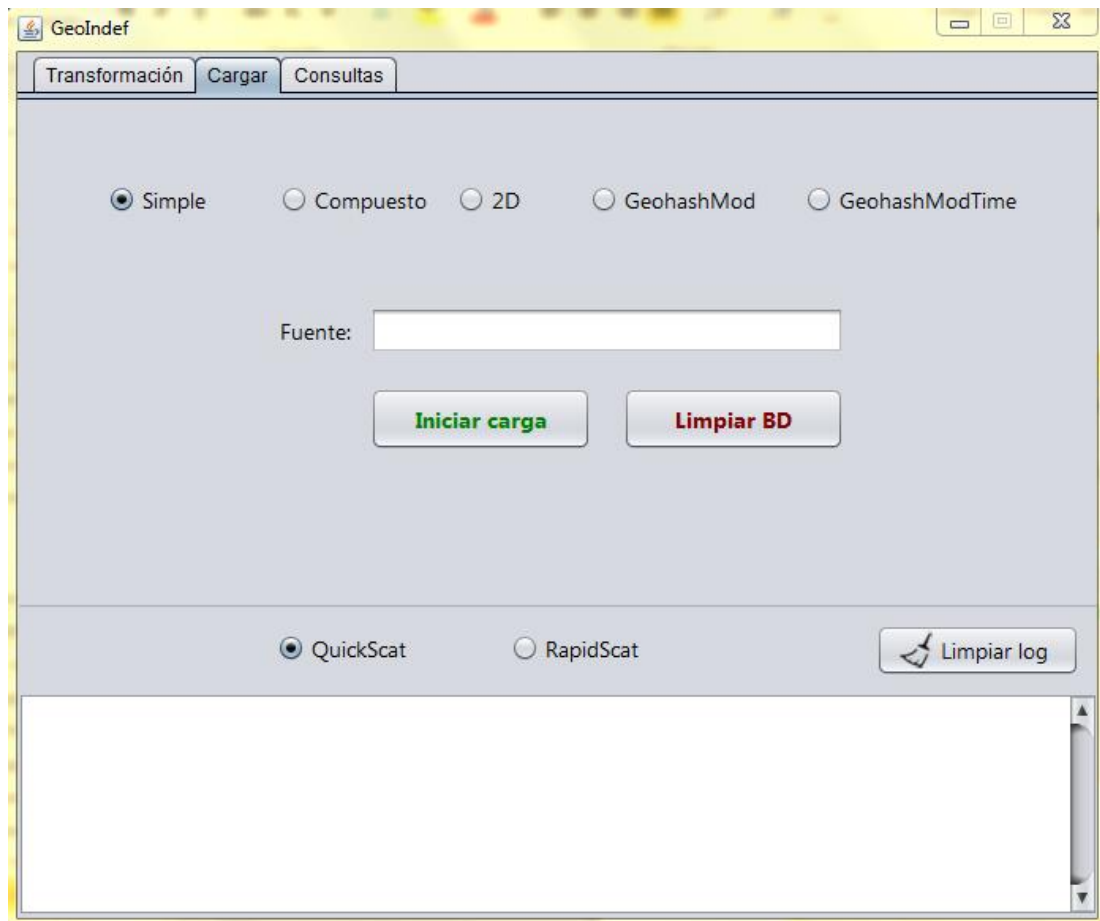


Figura 42: Pantalla de aplicación Java, Cargar.

7.3.3.- Consultas.

Desde este apartado podremos realizar las consultas sobre la base de datos. Al igual que en el apartado anterior, tenemos 2 grupos de botones de opción, el primer grupo muestra el tipo de índice que se utiliza (Simple, Compuesto, 2D, GeohashMod, y GeohashModTime) y el segundo grupo el tipo de dato (QuikSCAT y RapidSCAT).

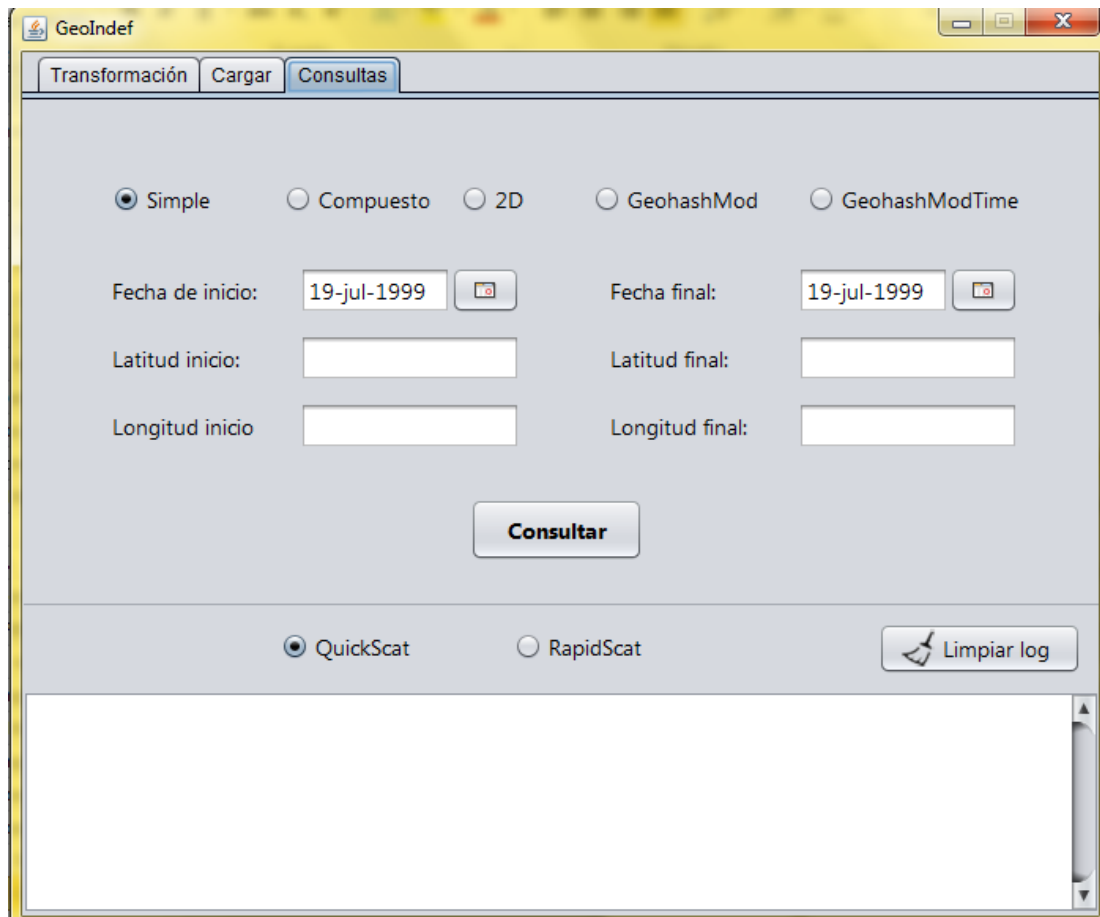


Figura 43: Pantalla de aplicación Java, Consultas.

Para realizar una consulta sobre una base de datos con un tipo de dato y un índice específicos, habrá que seleccionar las opciones correspondientes y proporcionar los datos específicos en las distintas cajas de texto.

Las cajas de texto de la izquierda, nos indican la fecha, latitud y longitud iniciales desde las que deseamos realizar la consulta, mientras que las de la derecha indican los datos finales.

Para seleccionar la fecha, habrá que tener en cuenta los datos de los que se disponen en la base de datos y seleccionarlos desde los desplegados.

Para la latitud y longitud, escribiremos las cantidades con hasta 2 cifras decimales con coordenadas negativas y positivas (latitud desde -90 hasta 90 y longitud, desde -180 a 180).

Estos valores siempre deben estar ordenados de forma que las cajas de texto iniciales contengan los valores inferiores y las cajas de texto finales contengan los valores superiores.

Una vez seleccionado los datos con los que se va a realizar la consulta, bastará con presionar el botón “**Consultar**” y esperar mientras se realiza. Tras un tiempo, en la pantalla inferior se mostrará el tiempo de consulta, filtrado y total, además de los datos que se han encontrado como válidos tras el filtrado.

En el directorio en el que se encuentre la aplicación también tendremos un archivo CSV con los datos que se han consultado bajo el nombre de “**ResultadoTipoIndice.csv**”, siendo “TipoIndice” el nombre del índice que se utilizó para realizar la consulta.

Bibliografía.

Continuum Analytics (2016). Download Anaconda Now! [en línea]. Disponible en <<https://www.continuum.io/downloads>> [12 de agosto 2016].

Edlich (2016). NoSQL databases. [en línea]. Disponible en <<http://nosql-database.org/>> [10 de agosto 2016]

Filezilla (2016). Overview [en línea]. Disponible en <<https://filezilla-project.org/index.php>> [15 de junio 2016]. JSON (2016). JSON [en línea]. Disponible en <<http://www.json.org/>> [13 de julio 2016].

Julian Browne (2009). Brewer's CAP Theorem [en línea]. Disponible en <<http://www.julianbrowne.com/article/viewer/brewers-cap-theorem>> [28 de junio 2016].

Kai Tödter (2016). JCalendar [en línea]. Disponible en <<http://toedter.com/jcalendar/>> [10 de agosto 2016]

Luis Miguel Gracia (2013). Eligiendo Base de datos NoSQL según el teorema CAP [en línea]. <<https://unpocodejava.wordpress.com/2013/05/29/eligiendo-una-base-de-datos-nosql-segun-el-teorema-cap/>> [28 de junio 2016].

Microsoft (2016). Información sobre el Service Pack 1 para Windows 7 y Windows Server 2008 R2 [en línea]. Disponible en: <<https://support.microsoft.com/es-es/kb/976932>> [23 de julio]

MongoDB (2008-2016a). Java MogoDB Driver [en línea]. Disponible en <<https://docs.mongodb.com/ecosystem/drivers/java/>> [10 de agosto 2016].

MongoDB (2008-2016b). Indexes. Index Types [en línea]. Disponible en <<https://docs.mongodb.com/manual/indexes/#index-types>> [30 de junio 2016].

MongoDB (2008-2016c). 2d Index Internals [en línea]. Disponible en <<https://docs.mongodb.com/manual/core/geospatial-indexes/>> [30 de junio 2016].

MongoDB (2016a). What is Big Data? [en línea]. Disponible en <<https://www.mongodb.com/big-data-explained>> [13 de agosto 2016].

MongoDB (2016b). What is NoSQL? [en línea]. Disponible en <<https://www.mongodb.com/nosql-explained>> [28 de junio 2016].

MongoDB (2016c) What is MongoDB? [en línea]. Disponible en <<https://www.mongodb.com/what-is-mongodb>> [28 de junio 2016].

MongoDB (2016d). MongoDB Download Center [en línea]. Disponible en <<https://www.mongodb.com/download-center#community>> [10 de agosto 2016].

NASA/JPL (2000), SeaWinds – Oceans, Land, Polar Regions [en línea]. Disponible en: <<http://photojournal.jpl.nasa.gov/catalog/PIA02458>> [13 de agosto 2016]

NASA/JPL (2016a), Missions QuikSCAT [en línea]. Disponible en: <<https://winds.jpl.nasa.gov/missions/quikscat/>> [13 de agosto de 2016].

NASA/JPL (2016b). Missions, RapidScat [en línea]. Disponible en <<http://winds.jpl.nasa.gov/missions/RapidScat/>> [13 de agosto 2016].

Numpy (2016). NumPy [en línea] disponible en <<http://www.numpy.org/>> [10 de agosto 2016].

Oracle (2016a). Conozca más sobre la tecnología Java [en línea]. <<https://www.java.com/es/about/>> [16 de junio 2016].

Oracle (2016b). Descarga gratuita de software de Java [en línea]. <<https://www.java.com/es/download/>> [10 de agosto 2016].

Phil Whelan (2011). Geohash intro [en línea]. <<http://www.bigfastblog.com/geohash-intro>> [15 de julio 2016].

Pramod Sadalage (2014) NoSQL Databases: An Overview [en línea]. <<https://www.thoughtworks.com/insights/blog/nosql-databases-overview>> [28 de junio 2016].

PODAAC(2016a). NSCAT, Mission Specification. [en línea]. <<https://podaac.jpl.nasa.gov/NSCAT>> [20 de junio 2016].

PODAAC(2016b). Repositorio FTP [en línea]. Disponible en <<ftp://podaac-ftp.jpl.nasa.gov/OceanWinds>> [10 de junio 2016].

PODAAC(2016c). Repositorio FTP datos QuikSCAT [en línea]. Disponible en <<ftp://podaac-ftp.jpl.nasa.gov/OceanWinds/quikscat>> [10 de junio 2016].

PODAAC(2016d). SeaWinds on QuikSCAT Level 2B Ocean Wind Vectors in 25 Km Swath Grid Version 2 [en línea]. Disponible en <https://podaac.jpl.nasa.gov/dataset/QSCAT_LEVEL_2B_V2> [11 de junio 2016].

PODAAC(2016e). Repositorio FTP datos RapidSCAT [en línea]. Disponible en <<ftp://podaac-ftp.jpl.nasa.gov/OceanWinds/rapidscat>> [10 de junio 2016].

PODAAC(2016f). RapidScat Level 2B Ocean Wind Vectors in 12.5km Slice Composites Version 1.2 [en línea]. Disponible en <https://podaac.jpl.nasa.gov/dataset/RSCAT_LEVEL_2B_OWV_COMP_12_V1.2> [11 de junio 2016].

Python Software Foundation (2001-2016). Python download [en línea]. Disponible en <<https://www.python.org/downloads/>> [10 de agosto 2016].

Python Software Foundation (1990-2016). NetCDF4 1.0.7: Python Package Index [en línea]. Disponible en <<https://pypi.python.org/pypi/netCDF4/1.0.7>> [10 de agosto 2016].

Ruíz Romero, Fco. (2012). "Procesamiento eficiente de datos del satélite QuikSCAT mediante tarjetas gráficas programables". Trabajo Fin de Máster del Máster Universitario en Computación Grid y Paralelismo de la Universidad de Extremadura. Directores: Antonio J. Plaza Miguel y Félix R. Rodríguez. Marzo, 2012.

SciPy developers (2016). Obtaining NumPy & SciPy libraries [en línea]. Disponible en <<http://www.scipy.org/scipylib/download.html>> [10 de agosto 2016].

Techopedia Inc (2016). Java Runtime Environment [en línea]. Disponible en <<https://www.techopedia.com/definition/5442/java-runtime-environment-jre>> [10 de agosto 2016].

Teomiro Villa, Daniel (2015). “Integración de datos de vientos marinos en Oracle NoSQL”. Trabajo Fin de Grado del Grado en Ingeniería Informática en Ingeniería del Software de la Universidad de Extremadura. Director: Félix R. Rodríguez. Septiembre, 2015.

The Eclipse Foundation (2016). Eclipse Mars 1 Packages. [en línea]. Disponible en <<https://eclipse.org/downloads/packages/release/Mars/1>> [10 de agosto 2016].

The HDF Group (2011). The HDF Levels of Interaction [en línea]. Disponible en <<https://www.hdfgroup.org/products/hdf4/whatishdf.html>> [10 de agosto 2016].

The HDF Group (2016). Obtaining the HDF Software [en línea]. Disponible en <<https://www.hdfgroup.org/release4/obtain.html>> [10 de agosto 2016].

Unidata (2016). Introduction and Overview [en línea]. Disponible en <<http://www.unidata.ucar.edu/software/netcdf/docs/index.html>> [10 de agosto 2016].

Wikipedia (2016a). NoSQL [en línea]. Disponible en: <<https://es.wikipedia.org/wiki/NoSQL>> [8 de agosto 2016].

Wikipedia (2016b) MongoDB [en línea]. Disponible en: <<https://es.wikipedia.org/wiki/MongoDB>> [8 de agosto 2016].