Universidad Carlos III de Madrid
Escuela Politécnica Superior
Grado en Ingeniería Informática

Proyecto de Fin de Grado

Extracción de información social desde Twitter
y análisis mediante Hadoop

Autor: Cristian Caballero Montiel
Tutor: Daniel Higuero Alonso-Mardones
Co-Tutor: Juan Manuel Tirado Martín

Septiembre 2012

AGRADECIMIENTOS

Esta sección ha sido reservada para mostrar todo mi agradecimiento a todas esas personas que, de un modo u otro, me han ayudado ofreciéndome su apoyo y paciencia en toda mi etapa universitaria.

En primer lugar, necesariamente tengo que dar las gracias a mis padres y a mi hermana por comprenderme y por apoyarme en todo momento y a mi novia, por la excesiva paciencia que ha tenido conmigo, además de su total apoyo, compresión y los buenos momentos que me hace pasar a diario.

A todos mis amigos y “colegas”, por permitirme desconectar algunos fines de semana y pasar buenos ratos, y a todos mis compañeros de clase y profesores, tanto los de la universidad, como los de mi antiguo instituto y colegio, porque sin ellos, no podría haber escrito este documento.

A Daniel Higuero y Juan Manuel Tirado, tutor y co-tutor de este proyecto, no solo por haberme guiado durante el desarrollo de este proyecto, sino porque gracias a ellos pude realizar una formación extra académica y profesional compaginándolo con el estudio del grado. Esto, sin duda, ha sido un gran punto de inflexión para mi carrera profesional, ya que me ha ayudado a valorar la importancia de seguir investigando y trabajando en el ámbito de la informática.

A todo el personal de la Universidad Carlos III de Madrid, destacando los docentes e investigadores del grupo de arquitectura de computadores y sistemas (ARCOS).

Por último, aunque de forma no tan directa y personal como los citados hasta ahora, también agradecer a todos los trabajadores de las tecnologías implicadas en el proyecto, ya que, sin ellos, este proyecto no habría tenido sentido. De este numeroso grupo de profesionales de las TI, destacar, entre otros, al desarrollador de la librería Twitter4J y a toda la gente que hay por detrás del desarrollo de Hadoop, por realizar un framework potente y distribuido para usar en grandes clusters de máquinas. También mencionar a Google por crear el paradigma MapReduce en clusters y, por tanto, ser motivación para el desarrollo de Hadoop.

A toda la gente que desarrolla software open source, por intentar hacer día a día una informática colaborativa y accesible a más gente y a todas las empresas que colaboran con esto.

Y en general, a toda la gente que me ha acompañado durante toda mi etapa universitaria, por todas las experiencias y buenos momentos vividos, y los que se vivirán. A toda la gente que me ha echado una mano en algún momento para alcanzar el objetivo final: presentar este TFG como previo a la obtención del título del Grado en Ingeniería Informatíca.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ACRÓNIMOS

1. **AWS**: Amazon Web Services
2. **API**: Application Programming Interface
3. **CUDA:** Compute Unified Device Architecture
4. **DM:** Direct Messages
5. **GB**: GigaByte
6. **GiB:** Giga-binary-Bytes
7. **GFS**: Google Fyle System
8. **GPU:** Graphics Processing Unit
9. **HaaS**: Hardware as a Service
10. **HP**: Hewlett-Packard
11. **HW**: Hardware
12. **IaaS**: Infrastructure as a Service
13. **IVA**: Impuesto al valor agregado
14. **Jar:** Java archive
15. **KB:** KiloBytes
16. **MB:** MegaBytes
17. **ME:** Mobile Edition
18. **PaaS**: Platform as a Service
19. **PFC:** Proyecto de fin de carrera

20. **POO**: Programación orientada a objetos
21. **RT**: Retweet
22. **SaaS**: Software as a Service
23. **SW**: Software
24. **TB**: TeraByte
25. **TFG**: Trabajo de Fin de Grado
26. **TI**: Tecnologías de la información
27. **UC3M**: Universidad Carlos III de Madrid
28. **UML**: Unified Modeling Language
29. **URL**: Uniform Resource Locator
30. **WSDL**: Web Services Description Language

1. INTRODUCCIÓN

En este apartado se pretende dar una presentación al trabajo realizado en el proyecto. Para ello, se introduce el proyecto a través de su motivación y sus principales objetivos. Acto seguido, se expone cual es el propósito de este documento y por último se detalla el alcance de este, enumerando cada uno de los capítulos y una breve descripción de cada uno de estos.

A grandes rasgos, el proyecto consiste en el desarrollo de un extractor de Twitter, su puesta en marcha para la obtención de datos masivos con respecto a las elecciones generales de 2011, y el posterior análisis utilizando Hadoop. Esto último resuelve de forma muy eficiente uno de los problemas principales del proyecto: el procesamiento de grandes cantidades de datos en un tiempo razonable. Para ello, se utilizan sistemas distribuidos y un paradigma de programación que cada día cobra más importancia dentro de las grandes empresas: MapReduce.

1.1 Motivación del proyecto

Durante estos últimos años, hemos sido conscientes del gran crecimiento que ha tenido la web, tanto es así que fue necesario dar paso a lo que hoy en día conocemos como Web 2.0 [1].

Este término está asociado a las aplicaciones web que permiten al propio usuario crear contenido y facilitan la compartición de estos. Es decir, es la representación de la evolución de las aplicaciones tradicionales a las aplicaciones enfocadas al usuario final.

Algunos de los servicios que ofrece la nueva Web 2.0 son los sistemas de etiquetado (TAGS), Blogs, Wikis, etc. Pero sin duda, los servicios que más crecimiento en cuanto a usuarios han tenido han sido los Microblogging y las redes sociales.

Las redes sociales son un medio de comunicación social centrado en encontrar gente para relacionarse on-line. Generalmente, esta gente mantiene una relación de amistad entre sí, aunque no siempre ocurre así. Algunos ejemplos de redes sociales son *Facebook* [2], *Hi5* [3] o *Google+* [4].

Los sistemas de microblogging, también conocido como Nanoblogging funcionan como un servicio que permite a sus usuarios publicar mensajes breves (alrededor de 140 caracteres como máximo) a través de internet, SMS o aplicaciones ad hoc. Algunos ejemplos de microblogging son *Plurk* [5], *Tmblr* [6].

Dentro de todo este gran conjunto de servicios, contamos con Twitter, uno de los más relevantes hoy en día. Twitter, aunque a menudo se la determina como red social, realmente pertenece al conjunto de servicios de microblogging. A pesar de todo, el gran aumento de usuarios que ha tenido en los últimos años han provocado que pase a tener un objetivo similar al de las redes sociales. Por tanto, se puede considerar a Twitter tanto como un servicio de microblogging como una red social.

Twitter contiene una gran cantidad de contenido público escrito por los usuarios que, a menudo, muestra la opinión de éstos con respecto a determinados temas. Además, este contenido puede ser analizado para obtener una visión de la opinión general de los usuarios, lo cual puede ser útil para realizar estudios sociológicos, de mercado o aplicaciones similares.

Sin embargo, esto tiene una desventaja. La cantidad de información a analizar es tanta que en ocasiones su posterior análisis se hace muy complejo. Por tanto, es interesante analizar todo este gran conjunto de datos utilizando sistemas distribuidos.

Dentro de los framework de software existentes que son utilizados junto a sistemas de altas prestaciones o sistemas distribuidos, Hadoop es uno de los más novedosos e interesantes para el análisis de datos.

1.2 Objetivos del proyecto

Partiendo del apartado anterior, la finalidad del proyecto es elaborar un extractor masivo de información para Twitter sobre las elecciones generales de España que tuvieron lugar el 20 de Noviembre del 2011 y un posterior análisis de tendencias mediante Hadoop.

Los datos extraídos pertenecen al contenido público de Twitter y por tanto, son accesibles para todo tipo de usuarios de la web. Una vez extraídos y almacenados estos datos, serán posteriormente analizados para obtener resultados estadísticos y posibles tendencias.

Partiendo del objetivo principal, podemos definir los siguientes subobjetivos:

- Estudio de la situación actual de las redes sociales y la información que en ellas se vuelca, poniendo especial énfasis en Twitter.
- Diseño e implementación de un extractor de datos sobre Twitter apoyado en el API de Streaming, de usuarios y de términos.
- Diseño e implementación de un filtro de idioma sobre Hadoop para centrar nuestro estudio únicamente en los tweets que estén en español.
- Diseño e implementación de una aplicación sobre Hadoop para análisis de datos y obtención de estadísticas.
- Análisis y evaluación de los resultados.

1.3 Propósito del documento

El presente documento se realizó con el fin de detallar el trabajo realizado durante el desarrollo del TFG, ya sea trabajo de diseño, implementación o de gestión del proyecto. Además, el documento

pretende mostrar los resultados del proyecto de una manera sencilla para el lector.

Al ser un trabajo con gran carga sociológica y basada en datos reales obtenidos, el documento muestra el proyecto como un trabajo de investigación en cuanto a la obtención y el análisis de los resultados estadísticos.

1.4 Estructura del documento

Para una mejor lectura del documento, se ha organizado la memoria en distintos apartados dependiendo de los temas a tratar:

- **Capítulo 1. Introducción**

Descripción general del proyecto. Motivación para la realización de este y definición de objetivos a seguir para el correcto desarrollo del proyecto.

- **Capítulo 2. Estado del arte**

Se estudian las distintas tecnología del contexto del proyecto actual. En concreto, se estudian por un lado tecnologías de la Web 2.0 en general, Twitter en particular y el desarrollo utilizando el API de Twitter Developers y por otro lado se estudian los principales conceptos de la computación de altas prestaciones, distribuida y el análisis masivo de datos. De esto ultimo, destacar las tecnologías de Cloud computing, MapReduce y Hadoop como implementación de este último.

- **Capítulo 3. Trabajo realizado**

Descripción del trabajo realizado con el mayor detalle posible. Este capítulo abarca desde el diseño e implementación de un extractor de datos de Twitter hasta el análisis de los resultados obtenidos, pasando por el filtrado de estos resultados.

- **Capítulo 4. Evaluación del proyecto**

Presentación de las pruebas realizadas para la verificación de que el trabajo realizado cumple con los objetivos definidos. Además, se detallan las distintas pruebas de rendimiento realizadas.

- **Capítulo 5. Gestión del proyecto**

En este apartado se muestra la metodología empleada durante el desarrollo del proyecto. Se muestra el proyecto como un conjunto de tareas y subtareas planificadas en un calendario y se describe el presupuesto final de la realización del proyecto.

- **Capítulo 6. Conclusiones**

Redacción de las principales conclusiones finales del proyecto, tanto a nivel técnico como a nivel personal.

- **Capítulo 7. Trabajo futuro**

Resumen del posible trabajo a realizar como consecuencia de los resultados de este o como complemento.

- **Capítulo 8. Referencias**

Bibliografía, documentación y referencias online que se han consultado como apoyo para la realización del proyecto.

2. ESTADO DEL ARTE

En este apartado se detallará el análisis que se ha realizado como trabajo previo al desarrollo del proyecto. Durante este análisis, se han estudiado las distintas tecnologías existentes que mantienen algún tipo de relación con el proyecto y/o cada una de sus subtareas.

A lo largo de este apartado, se mostrarán los conceptos teóricos sobre los temas a investigar y se detallarán todos aquellos que sean más relevantes o que tengan más importancia para el trabajo que nos ocupa. Finalmente, se mostrarán comparativas mostrando las características de las soluciones disponibles y la elección de esta.

2.1 Web 2.0

En este apartado se resumirá brevemente los aspectos más destacados de lo que llamamos Web 2.0. Esto ha dado mucho de que hablar puesto que es un concepto reciente y algo ambiguo. En esta sección, daremos una definición sobre el término Web 2.0, comentaremos algunos de los principales servicios y mostraremos las principales diferencias con la Web tradicional o Web 1.0.

2.1.1 Definición de Web 2.0

El término Web 2.0 está acuñado a *Tim O'Reilly* [7], fundador de *O'Reilly Media* [8] durante una sesión de brainstorming para desarrollar ideas para una conferencia. El termino Web 2.0 sugiere una nueva versión de la Web, aunque no se refiere a un cambio en las especificaciones técnicas, sino más bien a los cambios en cuanto a las especificaciones de los usuarios finales y los cambios en el desarrollo de páginas web.

La Web 2.0 propone una nueva visión de la World Wide Web, basada en la compartición de la información, el diseño centrado en el usuario y la colaboración entre los usuarios. Un sitio Web 2.0 es aquel que permite a los usuarios interactuar entre sí y crear contenido dentro de una comunidad virtual, a diferencia de la Web tradicional en la que los usuarios se limitan a observar los contenidos.

Una definición alternativa del termino vendría dada por el fenómeno social surgido a partir del desarrollo de algunas aplicaciones en internet tales como los blogs y las redes sociales entre otras. Todo lo anterior también trajo consigo un cambio en el modelo de negocio utilizado en la World Wide Web.

La siguiente figura, conocida como mapa de *Markus Angermeier*, muestra de forma visual este concepto.

Figura 1: Mapa de Markus Angermeier

2.1.2 Servicios de Web 2.0

La Web 2.0 ofrece una serie de servicios o herramientas diseñadas para la compartición o creación de contenido o la interacción de los usuarios. En lineas generales, estos servicios son los siguientes:

- **Etiquetado de recursos**: Consiste en categorizar o etiquetar los recursos de la web, de tal forma que la etiqueta representa una alusión semántica al recurso referenciado. Normalmente, esta organización o clasificación la introducen los propios usuarios. Una vez indizados los recursos, los sitios web proporcionan nubes de tags en las que el usuario puede ver los contenidos con etiquetas comunes.
- **Blog**: Un blog es un sitio web que recopila de forma cronológica textos, artículos o determinada información de uno o varios artículos. La información se muestra en el orden en el que se crea apareciendo primero el más reciente. El término Blog viene de Web + Log → Blog y se refiere a el registro de comentarios o de conversación entre usuarios en la web. Existen herramientas que permiten una fácil creación y alojamiento. Las más populares son *Blogger*[9] y *WordPress*[10].
- **MicroBlogging**: También conocido como NanoBlogging, es un servicio que permite a sus

usuarios publicar mensajes breves (generalmente, solo texto) con un tamaño alrededor de 140 caracteres. Las actualizaciones se muestran vía web, sms, aplicaciones ad hoc etc. Estas actualizaciones se muestran en la página de perfil del usuario y también son enviadas a otros usuarios que tengan la opción de recibirlas. El mejor ejemplo de este tipo de servicios es el que usaremos a lo largo del proyecto por su gran popularidad: *Twitter*.

- **Wikis**: Término proveniente del hawaiano wiki – rápido, de forma sencilla – Una wiki es una web cuyo contenido puede ser editado por múltiples usuarios a través del navegador web de forma sencilla y rápida. Los usuarios pueden crear, modificar o eliminar texto de la web de forma colaborativa. A menudo, las wikis funcionan como enciclopedias colectivas. La wiki más famosa es *Wikipedia* [11].
- **Compartición de recursos**: La web 2.0 llegó para adecuarse a la demanda de los usuarios en el uso de la web. Uno de los factores que más influyeron en esto fue la necesidad de compartir recursos a través de la web. Gracias a los servicios de compartición de recursos un usuario puede subir a la nube de internet cierto contenido al que cualquier otro usuario tendrá acceso. Existen infinidad de servicios de compartición de recursos, que podemos clasificar según el tipo de recurso compartido:
  - Documentos: *Google Docs* [12], *Issuu* [13], *Calameo* [14].
  - Fotos: *Flickr* [15], *MetroFlog* [16].
  - Presentaciones: *Slideshare* [17].
  - Vídeos: *Youtube* [18], *Vimeo* [19].
  - Generalistas: *Megaupload*[1], *MediaFire* [21], *Rapidshare* [22].
- **Redes Sociales**: Las redes sociales son un medio de comunicación utilizado para relacionarse on-line por personas que comparten algún tipo de relación. En los últimos años han cobrado mucha popularidad y cada día el número de usuarios de estas aumenta considerablemente. Existen multitud de redes sociales, que podemos clasificar entre generalistas y específicas. El ejemplo más popular entre las generalistas es sin duda *Facebook* y en cuanto a las específicas podemos encontrar redes como *Linkedin* (ámbito profesional), *MySpace* [23] (música) o *Tuenti* [24] (para edades específicas) entre otras.
- **Sistemas de recomendación**: Se encargan de recolectar la información sobre la que un determinado usuario está interesado para poder mostrarle después información similar (música, libros, páginas webs...). Como ejemplos contamos con *Genius* [25] o *Amazon* [26].

2.1.3 Tabla comparativa de Web 1.0 y Web 2.0

La siguiente tabla muestra una comparación entre la web 1.0 y la web 2.0

1 Desde el 19 de enero de 2012, *Megaupload* se encuentra inhabilitado por el FBI por supuesta infracción de derechos humanos. Se estima que *Megaupload* sumaba 150 millones de usuarios registrados y cerca del 4% del total del contenido de la web [20].

Web 1.0	***Web 2.0***
Web como diseño e información	Web como comunicación
Contenido estático	Contenido dinámico
El usuario solo consulta el contenido	El usuario crea y edita contenido.
Publicación de los contenidos sin participación de los usuarios	Participación en la creación de los contenido.
Existen versiones finales	No existen versiones finales. Se está constantemente en versión beta.
Tecnología asociada con HTML, JavaScript 1.0, etc	Tecnología asociada con XHTML, JavaScript, Google, etc
La actualización de los sitios web no es periódica	La actualización de los sitios web se hace de forma periódica y por parte de los usuarios.

Tabla 1. Tabla comparativa Web 1.0 vs Web 2.0

Como se puede ver en la tabla, en la Web 2.0 predomina el desarrollo constante y el contenido dinámico, a diferencia de la Web 1.0 en la que predomina el contenido estático. Además, la Web 2.0 utiliza tecnologías nuevas de desarrollo Web para dar soporte a la participación de los usuarios. Tecnologías de Streaming, las redes sociales, etc son algunos ejemplos que muestran el gran avance que está teniendo en los últimos años la Web desde el punto de vista del usuario. Por último, también hay que destacar que este desarrollo constante por la innovación complica el desarrollo Web al ofrecer funcionalidades complejas, algo que en la Web 1.0 era impensable, puesto que cada sitio Web era simplemente un contenedor de contenido que el usuario solo consulta, mientras que en la Web 2.0, predomina la creación y edición de contenido por parte de los usuarios.

2.2 Twitter

En este apartado daremos una breve visión al lector del *Twitter*, que es, cual es su estado actual, como funciona y en que consiste el API de Streaming de *Twitter*. Este último lo usaremos para la extracción de datos, parte esencial para el completo desarrollo del proyecto.

2.2.1 ¿Qué es Twitter?

Twitter es una de las aplicaciones web que más crecimiento ha tenido en los últimos años. Aunque la mayoría de la gente cree que Twitter es una red social, su filosofía es la de un sistema de Microblogging.

Twitter fue creado por *Jack Dorsey* [27] en marzo de 2006 y lanzado en Julio del mismo año, sin embargo, la versión española apareció el 4 de Noviembre de 2009. A día de hoy, se estima que cerca del

10% de usuarios de internet utilizan diariamente *Twitter*, como podemos ver en el siguiente gráfico tomado del sitio web de métricas www.Alexa.com , que muestra el uso de *Twitter* en los últimos meses[28].

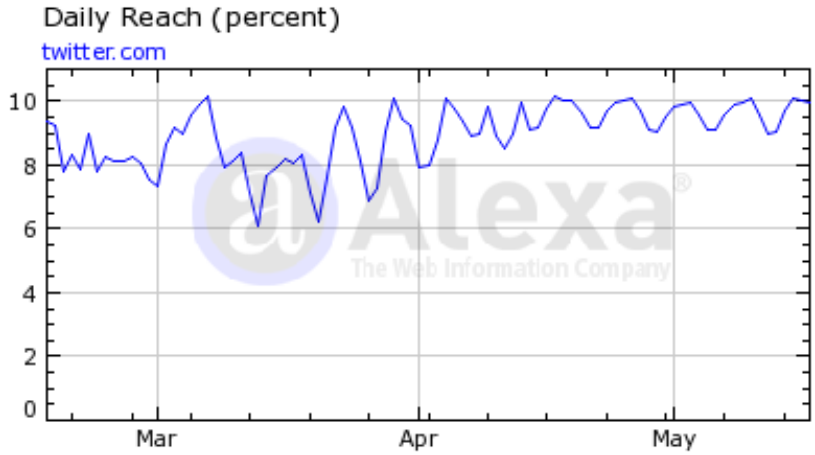

Figura 2: Tráfico de usuarios de Twitter

En este mismo portal, podemos ver que *Twitter* es el noveno sitio web más visitado del mundo, justo por debajo de el sitio de QQ (8º). El top site lo forman los sitios de *Youtube*(3º), *Facebook*(2º) y *Google*(1º). La siguiente tabla muestra las 10 primeras posiciones del top site, junto a una breve descripción de cada site.

Posición	Sitio web	Descripción
1º	Google	Permite a los usuarios buscar la información del mundo, incluyendo páginas web, imágenes y vídeos.
2º	Facebook	Red social que conecta a las personas, permite compartir enlaces y fotos, entre otras funciones.
3º	Youtube	Es una herramienta por la cual los usuarios pueden subir y compartir videos
4º	Yahoo	Uno de los portales de Internet más importantes. Ofrecen servicios de búsqueda, de chat y de correo electrónico, entre otros.
5º	Baidu	El principal motor de búsqueda chino.
6º	Wikipedia	Es una enciclopedia gratis, libre, de entorno colaborativo y accesible por todos.
7º	Windows live	Motor de búsqueda de Microsoft.
8º	QQ	Es un cliente de mensajería instantánea chino.
9º	Twitter	Servicio de microblogging que utiliza mensajería instantánea, interfaz web o SMS.
10º	Amazon	Servicio de comercio electrónico

Tabla 2: Top site de los 10 sitios web más visitados

2.2.2 Algunas definiciones importantes

Twitter cuenta con un cierto lenguaje para algunos términos, en este apartado, daremos el significado a los términos más utilizados dentro del mundo de *Twitter.*

- **Tweet**: Mensaje que publica (twittea) el usuario.
- **Followers**: Seguidores de una cuenta concreta.
- **Following**: Cuentas a las que sigues desde una cuenta concreta.
- **Mencionar:** Enviar un tweet a una persona específica. Para ello, es necesario indicarlo mediante *@nombreDestinatario*
- **Mensajes Directos:** Enviar un mensaje privado a otro usuario, de forma que solo pueda leerlo el usuario destinatario.
- **Retweet:** Reenviar(retwittear) un tweet.
- **Hashtag:** Tema de conversación utilizado para categorizar tweets. Para ello, se indica mediante *#nombreHashtag*
- **Trending Topics:** Son los temas de los que más se habla en *Twitter*.

2.2.3 Breve descripción del funcionamiento de Twitter

El funcionamiento de *Twitter* es muy simple: el usuario escribe en su página de perfil mensajes(tweets) con un máximo de 140 caracteres, que pueden ser leídos por cualquiera que tenga acceso a su página (la mayoría de los perfiles de Twitter son abiertos). Cada usuario tiene, además, una lista de seguidores (followers) y seguidos (following). Una vez que un usuario twittea un mensaje, este aparecerá en la página de perfil de todos sus seguidores.

Un usuario puede mencionar a otro. En este caso, se envía el tweet referido a un usuario específico, sin embargo, se diferencia de los mensajes directos (DM) en que al mencionar el contenido del tweet es visible por otros usuarios mientras que en los DM el contenido es solamente visible por el usuario destino.

Además de todo esto, *Twitter* permite a los usuarios retwittear mensajes de otro usuarios, en ese caso, el tweet saldrá en el timeline del usuario con el usuario original del tweet, e indicando el usuario que retwitteó el mensaje original.

El siguiente modelo de casos de uso resume la principal funcionalidad con la que cuenta un usuario de *Twitter*:

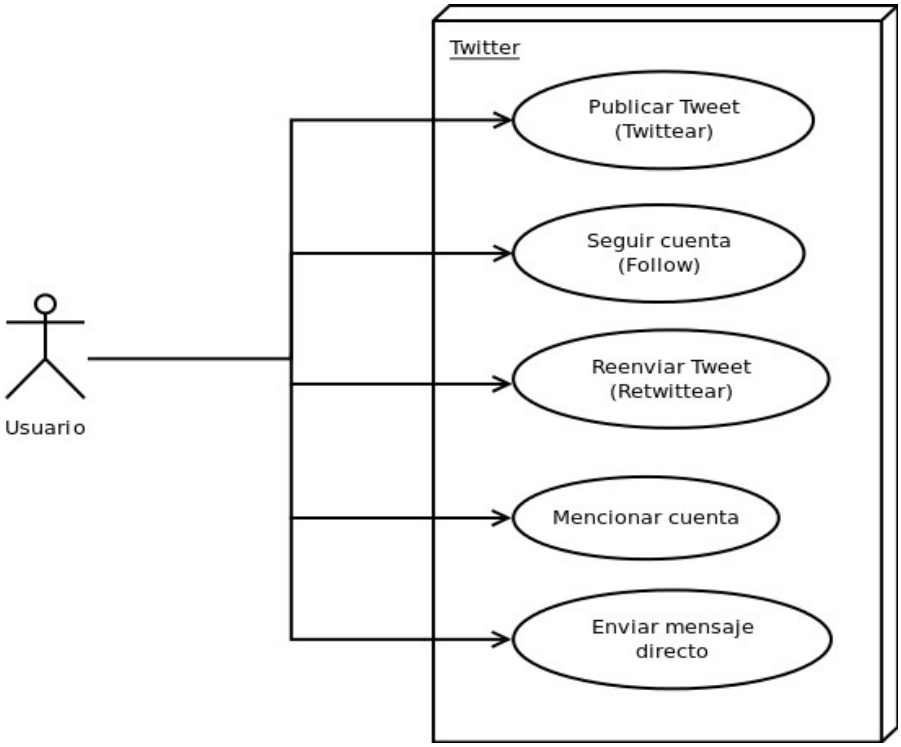

Figura 3: Diagrama de casos de uso Twitter

2.3 Twitter API

Twitter tiene publicada a disposición de cualquiera de sus usuarios su propia API, de forma que cualquiera puede crear aplicaciones que comuniquen con esta, teniendo en cuenta ciertas restricciones. En el presente apartado, haremos un análisis exhaustivo acerca del API de *Twitter* y como este nos ayuda para el desarrollo del proyecto.

2.3.1 Partes del API

El API de *Twitter* tiene tres partes claramente diferenciadas, dos pertenecen al API REST y la otra corresponde con el API de Streaming. Si bien, podemos dividir el API REST en dos APIs, quedando el API estructurado de la siguiente forma:

- **API Rest** [29]: Permite acceder a las funciones primitivas o básicas de *Twitter*, como actualizaciones de estado, obtener información del cierto usuario (nombre, descripción...) o responder ciertos tweets, entre otras muchas cosas.

- **API Search** [30]: La API de búsqueda está diseñada para consultar ciertos contenidos de *Twitter*. Por ejemplo, buscar tweets que mencionan a un cierto usuario, o tweets con ciertas palabras clave, entre otras.
- **API Streaming** [31]: Esta API está pensada para desarrollos con necesidades intensivas y de seguimiento de datos. Con ella podemos realizar una conexión HTTP durante más tiempo y mantener esa conexión abierta mientras realizamos las llamadas a las funciones del API. El API de Streaming permite consultas en tiempo real y con menos restricciones que en el API Search.

Las funciones de cada una de las partes del API están bien definidas en la documentación oficial de *Twitter developers* [32].

2.3.2 Diferencias entre REST y Streaming

A diferencia del API Rest y el API Search, el API Streaming necesita mantener una conexión HTTP abierta. Mediante esta conexión, se reciben los tweets que satisfacen las condiciones o filtros, se llevan a cabo los procesos necesarios y se almacenan los resultados.

En los APIs Rest y Search, el proceso funciona de una manera más sencilla. Consideremos, por ejemplo, una web que permite a los usuarios realizar consultas interactuando con el API Rest, esta consulta (o consultas) se realiza a través del servidor HTTP, que manda la petición al API de *Twitter*. Una vez procesada y obtenido el resultado de la consulta, se envía al usuario. El siguiente diagrama de secuencia muestra la interacción entre cada uno de los elementos:

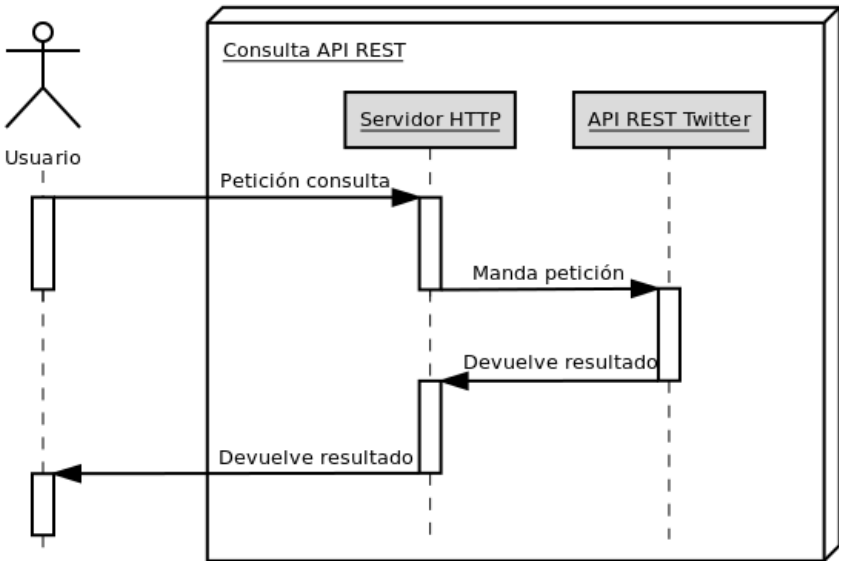

Figura 4: Diagrama de secuencia API REST Twitter

Las aplicaciones conectadas con el API de Streaming, al contrario, no establecen una conexión en respuesta a una petición del usuario. En su lugar, el mantenimiento de la conexión de Streaming lo realiza un proceso diferente al proceso que recibe las peticiones HTTP. El siguiente diagrama muestra la interacción de los distintos elementos al utilizar el API de Streaming:

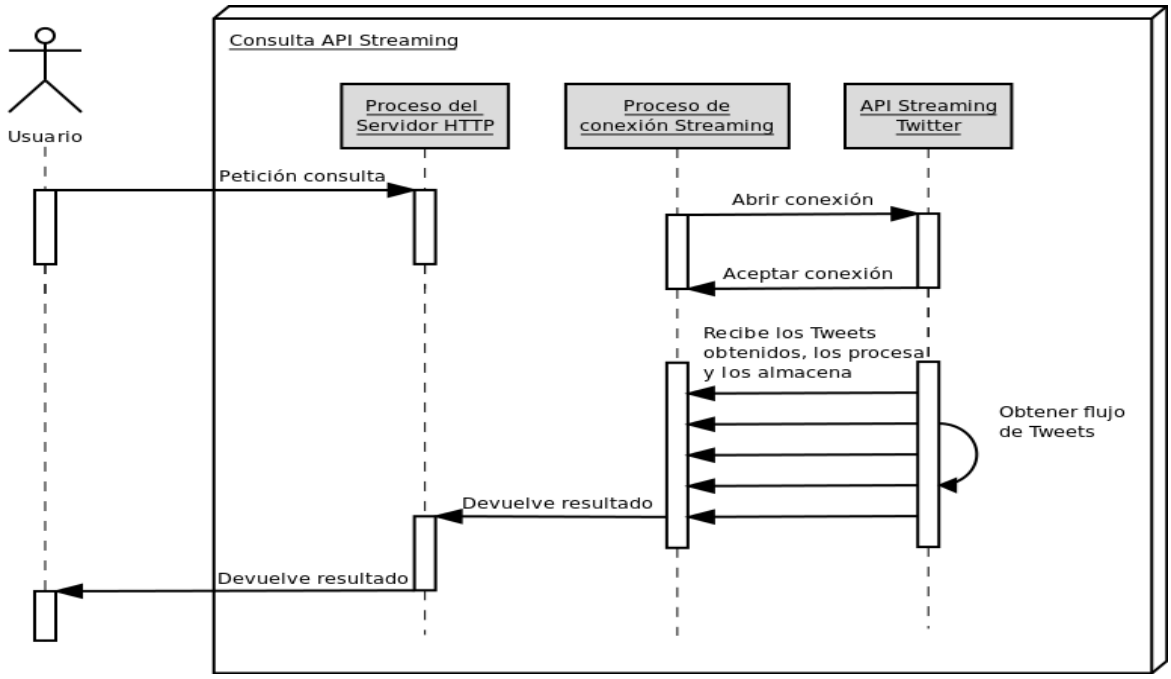

Figura 5: Diagrama de secuencia API Streaming Twitter

2.3.3 Librerías de Twitter

Desde el punto de vista del desarrollo, *Twitter* ofrece una serie de librerías para distintos lenguajes de programación con soporte al API de Twitter. En concreto, ofrece librerías para los siguientes lenguajes: *ActionScript/Flash, C++, Clojure, ColdFusion, Erlang, Java, JavaScript, .NET, Objective-C/Cocoa, Perl, PHP, Python, Ruby y Scala*.

Para el desarrollo del proyecto, nos centraremos únicamente en las librerías de Java [33], que es el lenguaje en el que está escrito la mayor parte del proyecto.

Las librerías de Java para el uso del API de Twitter son las siguientes:

- **Scribe** [34]: Librería con soporte para autenticaciones con el protocolo *Oauth*.
- **Twitter API ME** [35]: Librería con soporte para autenticaciones básicas con el protocolo *Xauth*
- **JTwitter** [36]: Pequeña librería con fácil acceso al API de Twitter, permite realizar operaciones primitivas con el API REST y API Search, pero no dispone de un API de Streaming robusto.

- **Twitter4J** [37]: Librería no oficial bastante completa. Permite distintos protocolos de autenticación y ofrece una interfaz bastante sencilla e intuitiva.

La siguiente tabla muestra una rápida comparativa entre las ventajas y desventajas principales de cada una de ellas de cara al desarrollo del proyecto.

Librería	Ventajas	Desventajas
Scribe	Librería que implementa el protocolo *Oauth*	No implementa el API de *Twitter* propiamente dicho
Twitter API ME	Bien definida Soporte para java ME (Mobile Edition)	Solo implementa el protocolo *XAuth* (en las versiones más actuales, existe beta para el protocolo *OAuth*)
JTwitter	Simple y fácil de utilizar para operaciones primitivas	Difícil de implantar Streaming
Twitter4j	Librería completa y sencilla No requiere ninguna dependencia ni configuración adicional Funciona en cualquier plataforma Java	Los ejemplos de la distribución no compilan en una versión de Java inferior a 5.0

Tabla 3: Comparativa librerías java para desarrollo con API Twitter

Teniendo en cuenta todo lo anterior, la librería que se ha elegido para el desarrollo del Crawler de *Twitter* ha sido *Twitter4J* por su facilidad de uso y robustez del código. Además, su configuración es simple y su aprendizaje mediante ejemplos es trivial, de forma que permite ponerse a trabajar con ella rápidamente con pocas lineas de código.

2.4 Cloud Computing

En este apartado se dará una visión general sobre Cloud Computing, uno de los conceptos más novedosos de la actualidad y se analizaran algunas de las soluciones de implementación existentes en la actualidad.

2.4.1 Concepto de Cloud Computing

En los últimos años, se ha puesto en alza una nueva tecnología llamada “Cloud Computing” o “computación en la nube”. Cloud Computing es un paradigma que nos ofrece servicios de computación

a través de internet (de ahí el término “nube”), gracias al cual logramos una capacidad de cómputo mayor además de otras muchas ventajas, entre ellas las siguientes:

- Reducción de los tiempos de inactividad
- Reducción de recursos, ya que utilizamos única y exclusivamente los necesarios
- Simplicidad y facilidad de uso
- Ahorro de hardware, ya que esta externalizado
- Abstracción de implementación e infraestructura

El termino Cloud se utiliza para definir una arquitectura por la cual usuarios o desarrolladores tienen acceso a aplicaciones o servicios a través de internet, donde la información fluye de forma aparentemente desconocida “en la nube”. Gracias a esto, conseguimos abstraernos de la infraestructura física que hay detrás, ya que las aplicaciones se ejecutan en una máquina física que no está especificada, los datos se almacenan en ubicaciones externas, cuya administración es responsabilidad de un tercero.

La clave del Cloud Computing es la abstracción de la compartición de recursos físicos a través de la virtualización. En una red distribuida, el Cloud Computing hace referencia a los servicios que se ejecutan en esta a partir de la virtualización, utilizando protocolos de comunicación en redes.

El Cloud Computing empezó en grandes proveedores de servicios de Internet, como Google y *Amazon*, que ofrecían servicios de pago según el consumo realizado. Además, permite acceder a un catálogo de servicios estandarizados a través de internet para responder las necesidades del negocio. En el caso de picos temporales de trabajo, en los que se necesita un computo mayor, se puede pagar únicamente por el consumo realizado, lo que nos permite adaptar el negocio a necesidades temporales y/o extremas en todo momento, pagando únicamente por lo que utilizamos.

2.4.2 Modelos de Cloud Computing

La computación en la nube provee sus servicios acorde a tres tipos de modelos o capas diferenciadas: Infraestructura como servicio (*IaaS*: infrastructure as a service), plataforma como servicio (*PaaS*: platform as a service) y software como servicio (*SaaS*: software as a service). La figura 6 muestra la jerarquía que guardan estas capas [38].

A modo de introducción y sin entrar al máximo en detalle, a continuación explicaremos cada uno de los modelos diferenciados que se utilizan cuando se adopta Cloud Computing.

- **Software as a Service (SaaS)**: Se encuentra en la capa más alta y consiste en ofrecer una aplicación completa como un servicio demandado, es decir,una sola instancia de un software de cierta empresa que sirve a cada cliente utilizando su propia infraestructura. Como ejemplo de este modelo, encontramos los servicios de *GoogleApps* [39], *ExchangeOnline* [40] y *SalesForce.com* [41].
- **Platform as a Service (PaaS)**: Se encuentra en la capa intermedia. Es la abstracción de un

ambiente de desarrollo, de tal forma que el desarrollador se olvida del hardware sobre el que trabaja, teniéndolo a su alcance en cualquier momento. Además, trabajando mediante este paradigma, la figura del administrador de sistemas “desaparece” puesto que estos sistemas son administrados por el personal encargado de ofrecer la plataforma como servicio. El ejemplo más claro de esta tecnología es *GoogleAppEngine* [42], *Windows Azure* [43], *Force.com* [44].

- **Infrastructure as a Service (IaaS):** También conocido como HaaS(Hardware as a Service). Se encuentra en la capa inferior y, actualmente,es el servicio de Cloud más utilizado actualmente. Los proveedores ofrecen hardware como almacenamiento básico de información o como servicios de cómputo basados en red. Estos servicios pueden ser, por ejemplo, redes de computo en las que el cliente lanza un trabajo de forma distribuida. El proveedor fijará el coste en función de los recursos utilizados o el coste, entre otros. Dos de los más conocidos son *Amazon EC2* e *IBM Blue Cloud* [45]. En el siguiente apartado, analizaremos varias soluciones de este modelo de Cloud Computing.

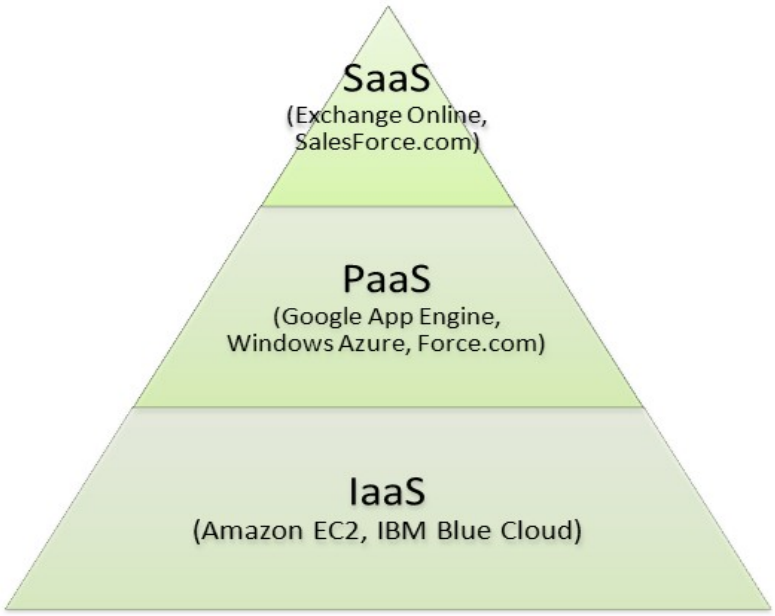

Figura 6: Capas de Cloud Computing

2.4.3 Tecnologías de Cloud Computing

Debido a que *IaaS* es el servicio más utilizado de la computación en la nube y a su importancia para el desarrollo del proyecto, en este apartado, analizamos y comparamos varias de las soluciones existentes.

OpenStack

Es un proyecto de código abierto liderado por la empresa *RackSpace* y la *NASA*. Permite implementar nubes privadas en clusters gracias a sus tres componentes: *Compute (NOVA), Object Storage (Swift) y Image Service (Glance)*. [46]

- **Compute(NOVA)**: Es el componente software encargado de instanciar y administrar redes de máquinas virtuales. Entre sus funciones, se encuentran las de administrar los recursos virtualizados y las redes locales. Está escrito en *Python* y utiliza muchas bibliotecas externas, como *Eventlet* (para la programación concurrente), *kombu* (para la comunicación *AMQP* y *SQLAlchemy* (para el acceso a la base de datos)
- **Object Storage(Swift)**: Se encarga de crear objetos de almacenamiento masivo, escalable y redundante. Permite un almacenamiento ilimitado ya que administra el espacio de forma elástica, aumentando y reduciendo este cuando sea necesario.
- **Image Service (Glance)**: Proporciona el descubrimiento, registro y entrega de servicios para las imágenes de discos virtuales. Soporta varios formatos, entre ellos, *VDI(VirtualBox), QCOW2(Quemu, KVM)* y *VMDK(VMWare)*

Eucalyptus

Es una plataforma open source para la implementación de computación en la nube de forma privada en clusters. Viene integrado con la distribución GNU/Linux Ubuntu desde la versión 9.04 como un útil de computación en la nube [47]. Las funciones más importantes de *Eucalyptus* se resumen a continuación:

- Implementa el API de *Amazon Web Services (AWS)*, lo que le permite la interoperabilidad de *Eucalyptus* con servicios *AWS*.
- Instalación sencilla utilizando paquetes *DEB* y *RPM*.
- Comunicación segura ente procesos utilizando protocolos *SOAP* o *WS-Security*
- Posee herramientas de administración básica, así como gestión de usuarios y grupos.
- Posee soporte para máquinas virtuales Linux y Windows.

La siguiente figura, tomada de la web de *Eucalyptus*, muestra su arquitectura de Software :

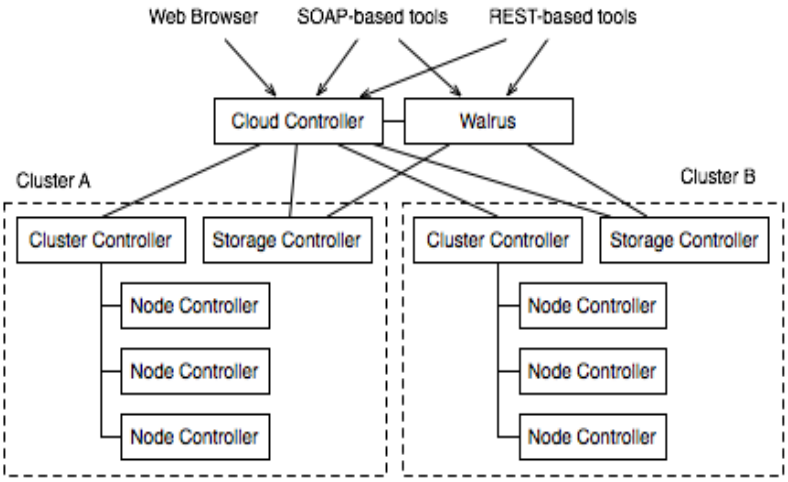

Figura 7: Arquitectura de Eucalyptus

Como vemos, existen, al menos cinco componentes o módulos claramente diferenciados: *Cloud Controller, Cluster Controller, Walrus, Storage Controller y Node Controller*. Cada uno de estos componentes esta implementado como un servicio web independiente y tienen su propia interfaz web. Este diseño tiene grandes ventajas, la más importante es que cada servicio Web expone una API bien definida y completamente independiente del lenguaje de programación utilizando el formato *WSDL (Web Services Description Language)*.

OpenNebula

OpenNebula es una solución software open source que permite construir clouds privados, públicos e híbridos. Ha sido diseñado para ser integrado en cualquier tipo de red y centro de datos.[48]

OpenNebula administra y gestiona el almacenamiento, la red y el método de virtualización, además, cuenta con tecnología para monitorizar estas tareas. También proporciona seguridad mediante su sistema de gestión de usuarios. Para ello, el usuario administrador será el único que indica que usuarios (no privilegiados) utilizan ciertas máquinas y/o redes virtuales.

La arquitectura interna de *OpenNebula* se divide en tres capas claramente diferenciadas:

- ***Tools***: Contiene herramientas de gestión empleando las interfaces proporcionadas por el núcleo del proyecto. Entre estas, contiene una herramienta que permite a los usuarios gestionar de forma manual la plataforma virtual.
- ***Core***: Contiene un conjunto de componentes para gestionar y monitorizar el núcleo de *OpenNebula*, es decir, herramientas para gestionar las máquinas virtuales, las redes virtuales, el almacenamiento y la infraestructura física (nodos).
- **Driver Modules**: Contiene algunos módulos específicos para interactuar con el middleware. El más destacado es el hipervisor.

Amazon EC2

Amazon Elastic Compute Cloud (*Amazon EC2*) es un servicio web que proporciona herramientas de computación en la nube de manera flexible. Posee una interfaz web que permite configurar y monitorizar los servicios de manera intuitiva. Además, proporciona capacidad de manera "elástica" y ágil, de tal forma que solo se utilizan los recursos estrictamente necesarios dependiendo de las necesidades.

A diferencia de las soluciones estudiadas hasta ahora, no se conocen los detalles internos del funcionamiento del Cloud de *Amazon*, sin embargo, podemos ver en la web de *Amazon* [49] las distintas instancias que nos ofrecen según nuestras necesidades.

Las instancias se pagan por horas de uso, es decir, si un cliente desea una instancia durante menos de una hora, se facturará como una hora completa.

Amazon EC2 está cambiando el modelo económico de la informática, ya que permite pagar sólo por la capacidad y los recursos utilizados realmente.

Google Engine

Esta tecnología, desarrollada por *Google*, nace con la intención de competir con *Amazon EC2*. *Google Engine* tiene la misma filosofía que su competidor: obtener sistemas virtualizados escalables según las necesidades del usuario [50].

Las instancias de máquinas virtuales que ofrece *Google Engine* son imágenes basadas en *Ubuntu 12.04 (Precise Pangolin)* o *CentOs 6.2*, aunque también nos permite crear nuestro propia imagen del sistema de ficheros.

Una vez obtenidas las máquinas virtuales, *Google Engine* nos permite conectarlas para utilizarlas en forma de cluster. Además, se ofrece una sencilla interfaz para configurar cualquier detalle de la máquina y/o sistema operativo. Por ejemplo, se nos permite de forma sencilla configurar el firewall o otras características del cluster virtual creado.

2.4.4 Resumen de las tecnologías de Cloud Computing

Una vez estudiadas algunas de las distintas tecnologías que implementan Cloud Computing, podemos elaborar la siguiente tabla resumen con algunas de sus principales características.

Implementación	Tipo de implementación de Cloud	Lenguajes de programación	Sistemas de virtualización	Licencia del código
OpenStack	Privado	Python	VirtualBox, QEMU, VMware	Open Source
Eucalyptus	Privado	Java, C	Xen, KVM, VMware	Open Source
OpenNebula	Público, privado e híbrido.	C++, C, Ruby, Java, Shell script	Xen, KVM, VMWare	Open Source
Amazon EC2	Público	N/A	Xen	Propietaria
Google Engine	Público	N/A	KVM	Propietaria

Tabla 4: Resumen de las características de las soluciones de Cloud Computing

Además de lo mostrado en la tabla anterior, hay que comentar que, mientras que todas estas tecnologías se presentan como *Infraestructure as a Service*, hay bastante diferencia en cuanto a *OpenStack*, *Eucalyptus* y *OpenNebula* por un lado y *Amazon EC2* y *Google Engine* por el otro.

La primera diferencia clara es que las tres primeras son herramientas de código abierto, y su uso suele ser el de implementar computación en la nube para un cluster privado, mediante herramientas de administración y gestión del cluster. En cambio, *Amazon* y *Google* ofrecen un modelo distinto, proporcionando infraestructura y recursos a través de la nube, pero no permitiendo crear una nube privada.

2.5 Procesamiento de gran cantidad de datos

En los últimos años, el volumen de datos que utilizamos a diario ha aumentado considerablemente. Tanto es así, que a la hora de administrar conjuntos masivos de datos, su procesamiento puede tener tiempos prohibitivos.

A menudo, aplicar procesamiento paralelo entre conjuntos de máquinas puede ser la solución, aunque suele ser difícil aprovechar al máximo los recursos.

Por todo esto, en los últimos años han aparecido distintas soluciones que permiten manejar grandes volúmenes de datos en sistemas distribuidos, entre las que destacan las que analizaremos en este apartado: el motor de base de datos distribuido *BigTable*, el sistema de ficheros distribuido *Google File System* (*GFS*) y el framework de software *MapReduce*. Todas creadas por *Google*.

2.5.1 BigTable

BigTable es un motor de base de datos creado por *Google* para satisfacer las necesidades de la compañía. Comenzó a ser desarrollado a principios de 2004 por la necesidad de crear un sistema

suficientemente grande y distribuido, algo que nunca se había pensado con los sistema se bases de datos relacionales. [51]

El proyecto fue presentado en Noviembre de 2006 en un paper de *Google* [52] . En el artículo, se muestra el funcionamiento y rendimiento de *BigTable*, además de algunas de sus aplicaciones reales dentro de la compañía (*Google Analytics*, *Google Earth*, etc).

BigTable almacena la información en tablas multidimensionales en forma de mapa, por tanto, puede ser definido como un mapa multidimensional, distribuido y ordenado, en lugar de el concepto de base de datos relacional.

El mapa es indexado por la clave de la fila, la clave de la columna y un timestamp y el valor obtenido es un string, por tanto, el mapeo clave-valor vendría dado como sigue:

```
(fila:string, column:string, time:int64) → string
```

La siguiente figura, tomada del paper de *Google* muestra un ejemplo:

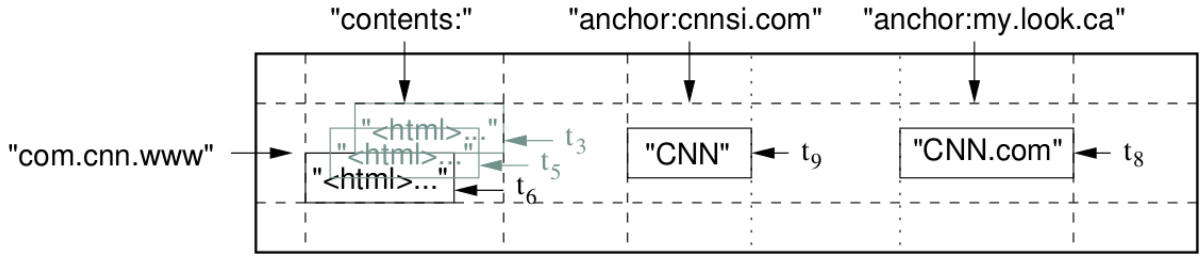

Figura 8: Ejemplo de tabla en BigTable que guarda datos sobre páginas web

- **Filas**: La clave de la fila es un String ordenado lexicográficamente, normalmente de unos 10-100 bytes de tamaño, aunque algunos de sus campos superan los 64KB. En la figura 8, se corresponde con la URL inversa. *BigTable* posee técnicas para particionar dinámicamente ciertos rangos de filas, llamados tabletas. Estas tabletas tienen un tamaño de entre 100 y 200 MB y son particionadas cuando aumentan de este tamaño, consiguiendo así balanceo de carga entre los nodos.
- **Columnas**: Las claves de las columnas son agrupadas en familias de columnas. Por tanto, cada fila tiene una familia de columnas y cada familia de columnas tiene una serie de columnas. En la figura 8, la familia de columnas contendría “contents” y “anchor” y cada uno de estos podría tener un conjunto de valores clave-valor indexados. La forma de acceder a estos últimos sería mediante la notación *familia:columna.* En nuestro ejemplo, se puede ver en *“anchor:cnnsi.com” y “anchor:my.look.ca”*
- **TimeStamps**: Cada celda puede contener varias versiones del mismo valor, por tanto, es

necesaria una marca de tiempo o TimeStamp. En *BigTable* esto es representado mediante un campo de tipo Integer de 64 bits. En la figura 8, podemos ver como "*contents*" tiene tres versiones, marcadas con t3, t5 y t6 mientras que *"anchor:cnnsi.com"* y *"anchor:my.look.ca"* tienen únicamente una versión, marcadas con t8 y t9.

Las ventajas que tiene *BigTable* con respecto a los sistemas de base de datos tradicionales son, entre otras, escalabilidad, alta disponibilidad y tolerancia a fallos.

2.5.2 Google File System

El *sistema de ficheros de Google (Google File System)*, *GFS*, es un sistema de ficheros distribuido propietario desarrollado por *Google*. *GFS* esta optimizado para el almacenamiento y procesamiento de datos según las necesidades de *Google*. Fue presentado en 2003 en el artículo de *Google* con el mismo nombre [53].

Para el diseño de *GFS*, se tuvieron en cuenta varias premisas:

- El fallo de un componente (tanto SW como HW) es la norma, no la excepción. El sistema está construido para que el fallo no le afecte.
- El sistema almacena archivos de gran tamaño. Los archivos de varios GB son comunes.
- La mayoría de las modificaciones son de adicción de grandes secuencias de datos a ficheros, las escrituras aleatorias casi inexistentes.
- Existen dos tipos de lecturas: Grandes lecturas de datos consecutivos y pequeñas lecturas aleatorias.
- Un gran ancho de banda prolongadamente es más importante que una baja latencia.

En *GFS*, los ficheros están divididos en bloques o *chunks* y típicamente estos bloques están en distintas máquinas. Es decir, un solo fichero está en varias máquinas en lugar de en una sola.

La arquitectura de *GFS* consiste en una gran cantidad de equipos (miles de componentes) juntos como parte de un sistema general (distribuido). Cada una de estas máquinas descansan sobre Linux y tienen la característica de ser ordenadores con características muy similares a los domésticos.

Existen dos tipos de servidores, los *Masters* y los *ChunkServers*. Los primeros almacenan la dirección física de los bloques o *chunks* que componen los ficheros y la jerarquía de ficheros y directorios, es decir, almacenan meta-datos. Los *ChunkServers* almacenan estos datos en bloques de 64MB.

Cada cluster que utilice GFS se compone de un servidor *Master* y múltiples *ChunkServers*. La arquitectura cliente-servidor en *GFS* se puede resumir en la siguiente figura, tomada de la presentación *GFS* realizada por los autores del paper [54] .

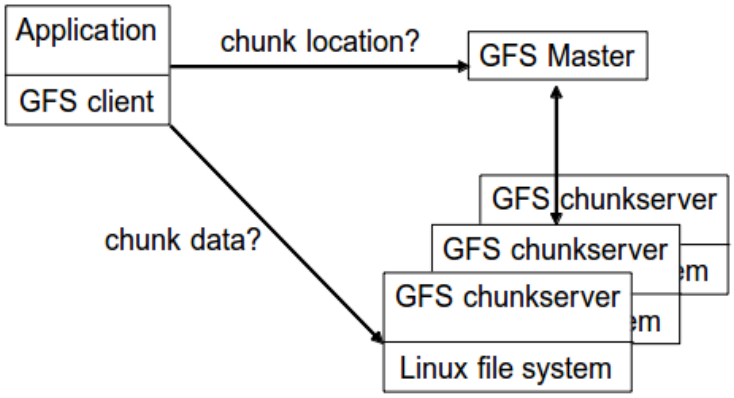

Figura 9: Arquitectura de Google File System

Como vemos, cuando se va a escribir en un fichero, el cliente "pregunta" en que *ChunkServer* puede escribir, consultando los meta-datos del *Master*. Estos meta-datos se mantienen en memoria y se suelen realizar tareas periódicas con el fin de reorganizar estos datos para optimizar las consultas. En general, la correspondencia de tamaño de meta-datos por bloque suele ser de 64 bytes, es decir, se tienen 64 bytes de meta-datos por cada 64MB de datos.

Muchos de estos meta-datos no se guardan de forma persistente, sino que el *Master* pide esta información periódicamente a los *ChunkServers*.

Existen *ChunkServers* primarios y secundarios. Los secundarios almacenan copias de los primarios. Cuando el cliente escribe en un fichero, se solicita almacenamiento en el *ChunkServer* primario. Este asigna a los datos unos números de serie y realiza la escritura. Acto seguido, el *ChunkServer* primario solicita al secundario que ejecuten las escrituras utilizando estos números de serie.

2.5.3 MapReduce

MapReduce es un *framework* de software introducido por *Google* en 2004 en el paper *MapReduce: Simplified Data Processing on Large Clusters* [55]. La motivación principal de este proyecto fue dar soporte a la computación paralela sobre grandes colecciones de datos en grupos de ordenadores.

El nombre *MapReduce* viene dado por la combinación de dos métodos de la programación funcional: *Map* y *Reduce*. Actualmente, existen implementaciones de *MapReduce* en C++, Java y Python.

Las funciones *Map* y *Reduce* se aplican sobre pares de datos (clave, valor). La función *Map* toma como entrada un par de datos de este tipo y devuelve una lista de pares en un dominio diferente.

```
Map(clave1 , valor 1 ) → Lista (clave 2 , valor 2)
```

Esta operación se realiza en paralelo para cada par de datos de entrada, por tanto se obtiene una lista de pares (clave, valor) por cada llamada a la función. Después de eso, el framework de *MapReduce* agrupa todos los pares generados con la misma clave de todas las listas, creando una lista por cada una de las claves generadas.

La función *Reduce* se realiza en paralelo tomando como entrada cada lista de las obtenidas anteriormente. Produciendo una colección de valores.

```
Reduce (clave 2 , lista(valor 2)) → Lista (valor 3 )
```

Desde la perspectiva del flujo de datos, la ejecución de *MapReduce* consiste en ejecutar n tareas *Map* y m tareas *Reduce*. La figura 10 muestra el flujo de ejecución de un trabajo *MapReduce*.

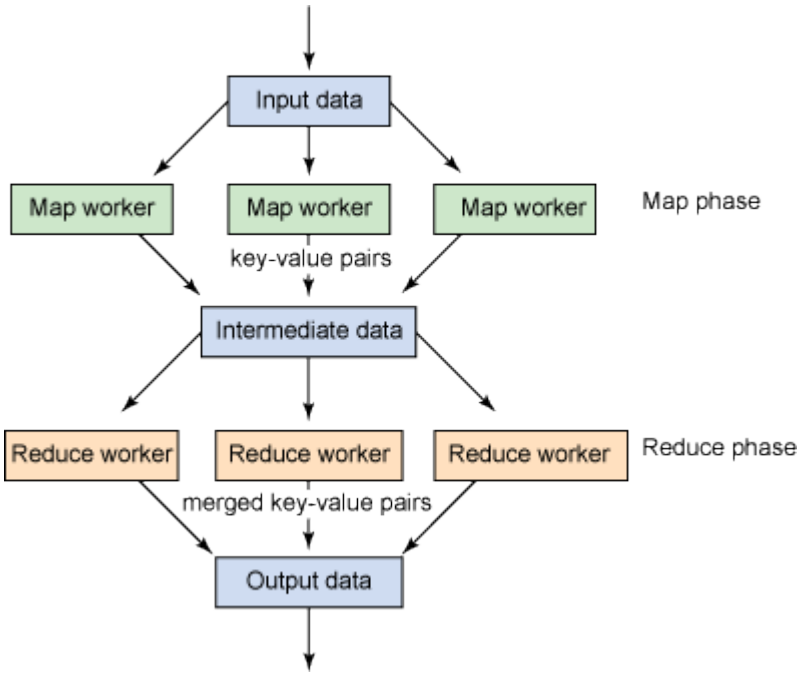

Figura 10: Flujo de ejecución de MapReduce

Para entender el funcionamiento, consideremos uno de los ejemplos básicos de *MapReduce*. Un trabajo *MapReduce* que cuenta las apariciones de cada palabra en una serie de ficheros: *WordCount*. En primer lugar, se dividen los datos de entrada en bloques de 16 a 128 MB (esta cantidad es configurable por el usuario) y cada uno de estos bloques se asigna a un proceso que realiza la función de *Map*, al que llamamos *Map worker* o *Mapper*. Cada *Mapper* lee cada palabra y emite como dato

intermedio el par (palabra, “1”). Acto seguido, la función intermedia o *Merge* agrupa los datos con la misma clave, de tal forma que se obtienen tantos pares (clave, listaValores) como tareas *Reduce* se ejecutarán. Cada una de estas tareas *Reduce* simplemente suma los valores de entrada y genera una única salida con la palabra y el número de estas. El siguiente pseudo-código muestra este ejemplo.

```
map(String key, String value):
      // key: Nombre del documento
      // value: Contenido del documento
      for each word w in value:
            EmitIntermediate(w, "1");

reduce(String key, Iterator values):
      // key: Una palabra
      // values: lista de valores
      int result = 0;
      for each v in values:
            result += ParseInt(v);
            Emit(AsString(result));
```

2.6 Hadoop

En este capítulo analizamos *Hadoop*, una de las herramientas libres que implementan *MapReduce*.

Hadoop es una implementación de código abierto de *MapReduce*. Fue desarrollado por la *Apache Software Foundation* con el fin de soportar aplicaciones distribuidas que utilicen el paradigma *MapReduce*. Permite a las aplicaciones trabajar con miles de nodos y grandes cantidades de datos (del orden de petabytes).

Hadoop cuenta con una gran cantidad de colaboradores, entre los que se encuentran desarrolladores de *Facebook* y *Yahoo*. Fue diseñado para analizar de forma rápida grandes cantidades de datos estructurado.

Una de las ventajas más importantes de *Hadoop* es la escalabilidad que ofrece. Al tener los datos distribuidos por todo el cluster, las lecturas se realizan de forma paralela.

2.6.1 Arquitectura y modos de ejecución

La arquitectura de *Hadoop* se fundamenta en tres subproyectos [56]:

- **Hadoop Common**: Contiene distintas utilidades utilizadas en los demás subproyectos de *Hadoop*.

- **Hadoop Distributed File System (HDFS)**: Un sistema de ficheros distribuido optimizado para el uso de *Hadoop*. Se explica en el siguiente apartado.
- **Hadoop MapReduce**: Un framework de software que implementa *MapReduce* en clusters para el procesamiento de grandes datos.

Dentro del motor *MapReduce*, contamos con distintos elementos que se utilizan utilizando como base el sistema de ficheros *HDFS*. La siguiente figura muestra la distribución de la arquitectura de *Hadoop MapReduce*. Esta tomada de uno de los libros más destacados para el aprendizaje de esta tecnología: *Hadoop: The Definitive Guide,* por Tom White [57].

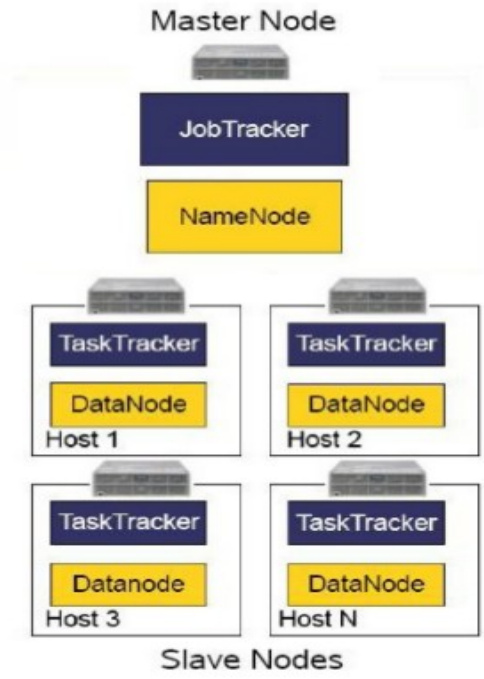

Figura 11: Arquitectura Hadoop sobre HDFS

Como vemos, La arquitectura cuenta con una serie de componentes claramente diferenciados:

- **JobTracker**: Es el planificador de trabajos que se encuentra en el nodo maestro. Es usado por las aplicaciones cliente para enviar trabajos *MapReduce* a los nodos esclavos.
- **NameNode**: Es el proceso que se encarga de almacenar las modificaciones del sistema de ficheros *HDFS* en un log.
- **TaskTracker**: Es el proceso que se encarga de procesar los trabajos *MapReduce* que se reciben del planificador de trabajos o *JobTracker*.
- **DataNode**: Es el proceso que se encarga del almacenamiento local de cada máquina. Es usado por el *NameNode* para ver los cambios en el sistema de ficheros.

Hadoop se puede ejecutar de tres formas distintas según nuestras necesidades:

- **Modo local o standalone**: Es la configuración por defecto de *Hadoop* instalándolo en un solo nodo como un proceso Java aislado. Se utiliza para depurar.
- **Modo pseudo-distribuido**: Es una configuración utilizada para pruebas en clusters. En este modo, se ejecutan los demonios de *Hadoop* en procesos diferentes de Java.
- **Modo distribuido**: Es la forma de aprovechar el potencial de *Hadoop* en grandes cluster de máquinas. Con este modo, se aprovecha al máximo el paralelismo entre los procesos del cluster.

2.6.2 HDFS: Hadoop Distributed File System

Hadoop Distributed File System (abreviado como *HDFS*) es el sistema de ficheros utilizado por *Hadoop* por defecto inspirado en el *GFS* de *Google*. Es un sistema de ficheros escrito en Java, distribuido y estructurado en bloques. Cada fichero se almacena en bloques de un tamaño fijo (normalmente 64MB, aunque es posible configurar cualquier otro valor) en múltiples máquinas.

HDFS replica la información que se va a procesar entre varias máquinas. Por defecto, el valor de replicación es 3, por lo que los datos se almacenan en tres nodos distintos: dos en el mismo rack y otro en un rack distinto. De esta forma, conseguimos fiabilidad y tolerancia a fallos.

En la figura 11 veíamos un ejemplo de la arquitectura de *HDFS*. Un cluster típico de *HDFS* se compone de un nodo maestro y N nodos esclavos. El nodo maestro contiene ejecuta los procesos para gestionar los trabajos *MapReduce* de los otros nodos, si un nodo esclavo falla, el nodo maestro re-planifica el trabajo a otro nodo.

HDFS ha añadido recientemente funciones de alta disponibilidad. Mediante un *namenode* secundario, se van realizando copias del estado del *namenode* primario, que pueden guardarse de forma remota. Estos “checkpoints” servirían como punto de recuperación en caso de fallo en el *namenode* primario.

HDFS posee herramientas propias de administración y monitorización, de forma que podemos verificar en cualquier momento en que estado se encuentra el sistema de ficheros y los trabajos ejecutados. Además de esto, cuenta con técnicas de balanceo automático de carga entre nodos y herramientas para copiar grandes conjuntos de datos entre *HDFS* y cualquier otro sistema de almacenamiento.

2.6.3 Ejemplos de uso

En los últimos años, un gran número de compañías han sido conscientes del potencial de *Hadoop* y han decidido implantarlo en sus sistemas. Algunos de los más relevantes son los siguientes [58]:

- **New York Times** utiliza *Hadoop* junto con *Amazon EC2* para generar artículos en formato PDF tomando como entradas imágenes escaneadas.
- **Yahoo** fue una de las empresas instigadoras de *Hadoop*. Utilizan más de 100,000 CPUs en más de 40,000 máquinas divididas en varios clusters para temas de búsqueda en web y tests de escalabilidad.
- **Facebook** utiliza *Hadoop* para almacenar copias de logs internos y grandes ficheros de análisis. Poseen dos clusters, uno de 1100 máquinas con 8800 cores y una traza de datos de, aproximadamente, 12 PB.
- **Twitter** utiliza *Hadoop* para procesar tweets, ficheros de logs y otros tipos de datos.

3. TRABAJO REALIZADO

En el capítulo 2 analizábamos las distintas tecnologías que guardan algún tipo de relación con el proyecto. Una vez realizado este estudio, en este capítulo hablaremos sobre el trabajo realizado durante el desarrollo del TFG.

Como ya se ha comentado, el trabajo realizado consiste en el diseño e implementación de un extractor de *Twitter* y su posterior análisis mediante *Hadoop*. El objetivo de este trabajo es obtener un conjunto de datos de gran tamaño y utilizar *Hadoop* para analizar estos datos de forma distribuida. Dado que los datos contienen un fuerte carácter social (elecciones generales de 2011), el análisis posterior vendrá dado en forma de gráficas y estadísticas. Con todo esto, se muestra al lector como *Hadoop* y el paradigma de *MapReduce* pueden tener aplicaciones de interés para las empresas.

3.1 Desarrollo de un extractor para Twitter

El primer paso para el correcto desarrollo del proyecto es la realización de un extractor o recolector de información para *Twitter*. Este extractor debe ser capaz de obtener y almacenar información sobre Twitter de manera estructurada, continua y, además, debe ser tolerante a fallos. Para ello, se apoya en el API de Streaming de *Twitter*.

El extractor de *Twitter* está íntegramente escrito en Java y se apoya en la librería *Twitter4J* para realizar las operaciones de escucha (Streaming) de tweets.

En este apartado se mostrará el diseño y la implementación del extractor o Crawler de *Twitter*. Primero se mostrarán los requisitos que debe cumplir el extractor en cuanto a la funcionalidad, se mostrará y analizará la información concreta que se va a extraer además de la arquitectura y funcionamiento del extractor y por último, se detallará el proceso de despliegue y ejecución del extractor.

3.1.1 Requisitos del extractor

Para el correcto desarrollo del proyecto, se ha decidido que el extractor debe cumplir ciertos requisitos para verificar su correcto funcionamiento.

La siguiente tabla muestra los requisitos mínimos que se han acordado para que el extractor se adapte a las necesidades del proyecto.

ID	Descripción del Requisito
R1	El extractor debe ser capaz de obtener los tweets filtrados por términos utilizando el API de Streaming
R2	El extractor debe ser capaz de obtener los tweets por usuario utilizando el API de Streaming
R3	El número de usuarios y términos que se pueden seguir al mismo tiempo estará limitado únicamente por el API de Twitter
R4	El extractor debe ser capaz de recuperarse de un error de red
R5	El extractor se apoyará en la librería Twitter4J
R6	Debe ser posible lanzar varias instancias del extractor en máquinas distintas con distintos usuarios mediante autenticación OAuth.
R7	El extractor almacenará los tweets en ficheros de texto

Tabla 5: Requisitos del extractor de Twitter

Los requisitos R1 y R2 son requisitos funcionales. Para conseguir los datos que queremos, necesitaremos extraer términos concretos y usuarios concretos. Estos deben guardar alguna relación con las elecciones generales de 2011. Esto se detallará más en el próximo apartado.

La idea del extractor es intentar obtener una traza lo mayor posible. Para ello, necesitamos extraer el mayor número de tweets de distintos términos y usuarios. En el requisito R3 se daba a entender que, efectivamente, *Twitter* tiene unas restricciones en cuanto al uso del API de Streaming para evitar sobrecarga. Por cada ejecución del extractor existe una limitación de seguimiento de 5000 usuarios y 400 términos. [59]

El requisito R4 es bastante trivial. Surge de la necesidad de mantener ejecutando el extractor durante varios días o meses. El objetivo es que, ante una posible caída de red, el extractor siga ejecutando. Durante ese intervalo de tiempo, no se recuperarán tweets, pero es necesario que el extractor vuelva a recuperar datos una vez se restablezca la conexión, sin ningún tipo de intervención manual.

El requisito R5 es simplemente una cuestión de diseño. Una vez probadas las demás librerías de *Twitter*, se llego a la conclusión de que *Twitter4J* era la librería más apropiada para cubrir nuestras necesidades de forma sencilla y eficiente.

Ya que el objetivo es lanzar varias instancias del extractor con distintos términos y usuarios, el R6 impone que puedan lanzarse varios extractores, siempre que el usuario configurado en cada extractor sea diferente. Al hacer esto, podemos “saltarnos” la limitación de términos y usuarios del API de *Twitter* y ejecutar los extractores en paralelo en una nube privada, como explicaremos más adelante.

Por último, el R7 impone que el almacenamiento se realice mediante ficheros de texto plano, en lugar de en base de datos. Aunque una base de datos permite almacenar un número amplio de filas en una única tabla, las tablas suelen tener un número máximo de filas (aproximadamente, 500,000) a partir del cual se debería realizar la partición de la tabla para optar por un despliegue distribuido. Además, dado que la intención es utilizar posteriormente *Hadoop* para filtrar y analizar los datos, el enfoque claro es el de el almacenamiento en texto plano para después gestionarlos por *HDFS*.

3.1.2 Análisis de los datos a extraer

El objetivo del extractor es obtener información sobre las elecciones generales de 2011. Para ello, se ha puesto en común una serie de usuarios y términos que se deben seguir para obtener la mayor información posible con respecto a este contexto social.

De esta forma, los parámetros de los que se alimentará el extractor serán los siguientes:

- **Una lista de usuarios**, que se almacenan en el fichero llamado '*users.dat*'. Esta lista de usuarios contiene el alias o "*screenName*" de los usuarios registrados. Estas cuentas de *Twitter* tienen relación completa con las elecciones de 2011 y pueden ser divididos en tres grandes grupos: usuarios con clara relevancia en alguno de los partidos políticos, periodistas y otros usuarios de interés. El número de usuarios analizados son 1695, correspondiente con el número de lineas del fichero '*users.dat*'.
- **Una lista de términos**, que se almacenan en el fichero '*terms.dat*'. Esta lista de términos contiene 61 palabras relacionadas con las elecciones de 2011 y el contexto social y político de ese año. Podemos dividirlas en dos grupos diferenciados:
  - Lista de hashtaghs: Una lista de topics generalistas a seguir. Algunos ejemplos son: *#20n, #15m, #eta, #conRubalcaba, #nolesvotes, #undebatedecisivo, #crisis, #spanishrevolution, etc.*
  - Lista de términos: Términos que nos dan un poco de visión de la opinión general sobre la situación. Algunos ejemplos son: *"elecciones", "crisis", "pp", "psoe", "iu", "zapatero", "rajoy", "rubalcaba", etc.*

3.1.3 Arquitectura, diseño e implementación del extractor

En este apartado trataremos los principales detalles del extractor: En primer lugar, detallaremos la arquitectura del extractor con respecto al API de *Twitter* y el almacenamiento en disco. Acto seguido, hablaremos de su implementación, mostrando cada una de las clases utilizadas y sus operaciones más importantes mediante un diagrama UML.

El extractor de *Twitter* está escrito íntegramente en Java y se apoya en el API de *Twitter* utilizando la librería *Twitter4J*. La siguiente figura resume la arquitectura del extractor:

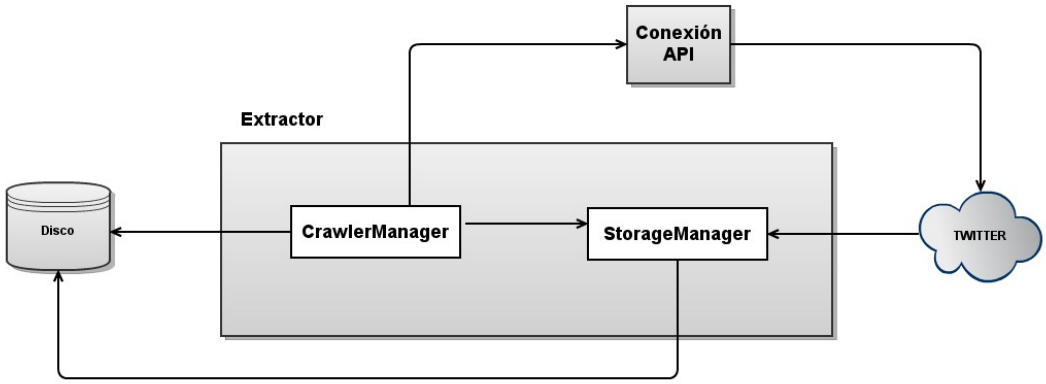

Figura 12: Arquitectura del extractor

Como vemos en la figura, hay dos componentes importantes dentro del extractor: CrawlerManager y StorageManager.

El CrawlerManager es el componente principal del extractor. Este se encarga de las funciones principales, como son establecer la conexión con el API, recuperar la conexión en caso de error y filtrar los términos y usuarios a seguir. De esta manera, el extractor queda a la escucha de los tweets que siguen estos criterios, es decir, queda a la escucha de los tweets que pertenecen a los usuarios definidos en el filtro de usuarios y a los tweets que contienen alguno de los términos definidos en el filtro de términos. Además, también responde ante otros eventos, como pueden ser el borrado o la modificación de uno de estos tweets, entre otros.

El StorageManager es el componente encargado del almacenamiento en disco. Entre sus funciones, se encuentran las de crear los ficheros donde se escribirán los tweets, escribir en estos ficheros y cerrarlos. Se encarga de almacenar un total de 10,000 tweets por fichero. Una vez almacenado este número, se cierra el fichero y se vuelve a crear uno similar, en el que se seguirán escribiendo los tweets.

Como ya hemos comentado, todos los mensajes recibidos son guardados en ficheros de texto plano, por lo que al iniciar el extractor, se crean varios ficheros:

- 1 fichero al que se escribirá toda la salida del programa, llamado: ***twitter_streaming_YYYY_MM_DD_hh_mm_log.txt***
- 1 fichero en el que se escribirán un mensaje indicando los Tweets que, por diversas causas, no se han podido obtener, llamado: ***twitter_streaming_YYYY_MM_DD_hh_mm_limitationNotice***
- 1 fichero para escribir todos los mensajes obtenidos por seguimiento de usuarios, llamado: ***twitter_streaming_YYYY_MM_DD_hh_mm_UserFollows***
- N ficheros, 1 por cada término introducido, para escribir todos los mensajes obtenidos por

seguimiento de términos, llamados: ***twitter_streaming_YYYY_MM_DD_hh_mm_término***

donde:

- **YYYY:** Es el año de creación del fichero {2011, 2012}
- **MM**: Es el mes de creación del fichero {01..12}
- **DD**: Es el día de creación del fichero {01..31}
- **hh:** Es la hora de creación del fichero {00..23}
- **mm:** Es la hora de creación del fichero {00-23}
- **término:** Es el término que identifica el fichero donde se guardarán los mensajes obtenidos por búsqueda de ese mismo término.

Cada Tweet obtenido junto a su información adicional se almacenan en una única linea. Dentro de cada linea, cada campo descriptivo está separado por un ";". Es decir, cada línea sigue la siguiente estructura:

```
nombreCampo1: valorCampo1; nombreCampo2:valorCampo2; [...]
```

La siguiente tabla muestra cada campo que se escribe por cada uno de los tweets.

Nombre del campo	Descripción	Tipo del campo Java
createdAt	Contiene la fecha de creación del tweet	Date
contributors	Contiene una lista de "contributors", o null si no hay ninguno	long[]
GeoLocation	Contiene la latitud y longitud del lugar desde donde se escribió el Tweet, o null en caso de que no haya sido posible capturar esta información	GeoLocation
hashtagEntities	Contiene un HashtagEntityJSON por cada hashtag, o null en cado de que no haya ninguno	HashtagEntity[]
id	Id del Tweet	long
inReplyToScreenName	Screen name del usuario al que se responde o null si el Tweet no es de respuesta	String
inReplyToStatusId	Id del Tweet al que se responde o -1 si el Tweet no es de resuesta	long
inReplyToUserId	Id del usuario al que se responde o -1 si el Tweet no es de respuesta	long

Nombre del campo	Descripción	Tipo del campo Java
place	Contiene un PlaceJSON con la información asociada al lugar (país, calle...)	Place
retweetCount	Número de retweet	long
retweetStatus	Texto del retweet o null si no es un RT	String
source	Origen del tweet (web, Iphone, Android...)	String
urlEntities	Contiene un URLEntity por cada url que haya en el tweet	URLEntity[]
user	Contiene un UserJSON con la información del usuario del tweet (nombre, descripción, imagen de perfil...)	User
userMentions	Contiene un UserMentionJSON por cada usuario mencionado con la información del usuario (screenName, nombre, id...) o null si no hay menciones	UserMentionEntity
isFavorited	Devuelve true si el tweet es favorito o false en caso contrario	boolean
IsRetweet	Devuelve true si el tweet es un Rto false en caso contrario	boolean
IsTruncated	Devuelve true si el tweet es “truncated” o false en caso contrario	boolean
Text	Contiene el texto del tweet	String

Tabla 6: Campos almacenados por cada tweet

Para entender esto mejor, el siguiente texto muestra una única linea almacenada en un fichero de *twitter_streaming_#20N*, resaltando en negrita el nombre de los campos para mayor claridad:

```
Created at: Mon Jan 02 14:04:39 CET 2012; contributors: null; geoLocation:
null; hashtagEntities: HashtagEntityJSONImpl{start=72, end=81,
text='recortes'} HashtagEntityJSONImpl{start=118, end=131,
text='tomadostazas'} HashtagEntityJSONImpl{start=132, end=136, text='20N'} ;
ID: 153824020603748352; inReplyToScreenName: null; inReplyToStatusId: -1;
inReplyToUserId: -1; place: null; retweetCount: 0; retweetedStatus: RT
@JulioJuanDB RT @CritiConPetrer Preparados para los #recortes en... 3, 2,
1... No quer?as cambio? #tomadostazas #20N #Rajoypresidente; source: web;
urlEntities: ; user: UserJSONImpl{id=271966992, name='Juan Fran',
screenName='kabash85', location='Elda', description='Aprendiendo a vivir,
que es como mas se disfruta ^_^ PAZ Y AMOR PARA TODOS!!',
isContributorsEnabled=false,
profileImageUrl='http://a0.twimg.com/profile_images/1720417869/4Y5ab1TR_norm
al',
profileImageUrlHttps='https://si0.twimg.com/profile_images/1720417869/4Y5ab1
TR_normal', url='null', isProtected=false, followersCount=31, status=null,
profileBackgroundColor='EDECE9', profileTextColor='634047',
```

```
profileLinkColor='088253', profileSidebarFillColor='E3E2DE',
profileSidebarBorderColor='D3D2CF', profileUseBackgroundImage=true,
showAllInlineMedia=true, friendsCount=62, createdAt=Fri Mar 25 15:58:41 CET
2011, favouritesCount=1, utcOffset=3600, timeZone='Madrid',
profileBackgroundImageUrl='http://a0.twimg.com/profile_background_images/376
262610/manita_pique.jpg',
profileBackgroundImageUrlHttps='https://si0.twimg.com/profile_background_ima
ges/376262610/manita_pique.jpg', profileBackgroundTiled=true, lang='es',
statusesCount=847, isGeoEnabled=true, isVerified=false, translator=false,
listedCount=0, isFollowRequestSent=false}; userMentions:
UserMentionEntityJSONImpl{start=3, end=15, name='Julio Juan DB',
screenName='JulioJuanDB', id=302657693}UserMentionEntityJSONImpl{start=20,
end=32, name='Julio Juan DB', screenName='JulioJuanDB',
id=302657693}UserMentionEntityJSONImpl{start=36, end=51,
name='CritiConPetrer', screenName='CritiConPetrer', id=325522024};
isFavorited: false; isRetweet: true; isTruncated: true; Text: RT
@JulioJuanDB: RT @JulioJuanDB RT @CritiConPetrer Preparados para los
#recortes en... 3, 2, 1... No quer?as cambio? #tomadostazas #20N ...
```

Como vemos, las líneas de los ficheros son relativamente largas (del orden de 1000 a 2000 caracteres) ya que almacenamos gran cantidad de metadatos. Toda esta información será la que utilicemos para analizar los datos y obtener resultados estadísticos.

La aplicación se compone de cuatro clases Java organizados en cuatro ficheros diferentes. En la figura 14 mostramos el modelado de clases en lenguaje UML [60]. Para mayor simplicidad, solo se muestran los métodos y atributos de las clases más significativos.

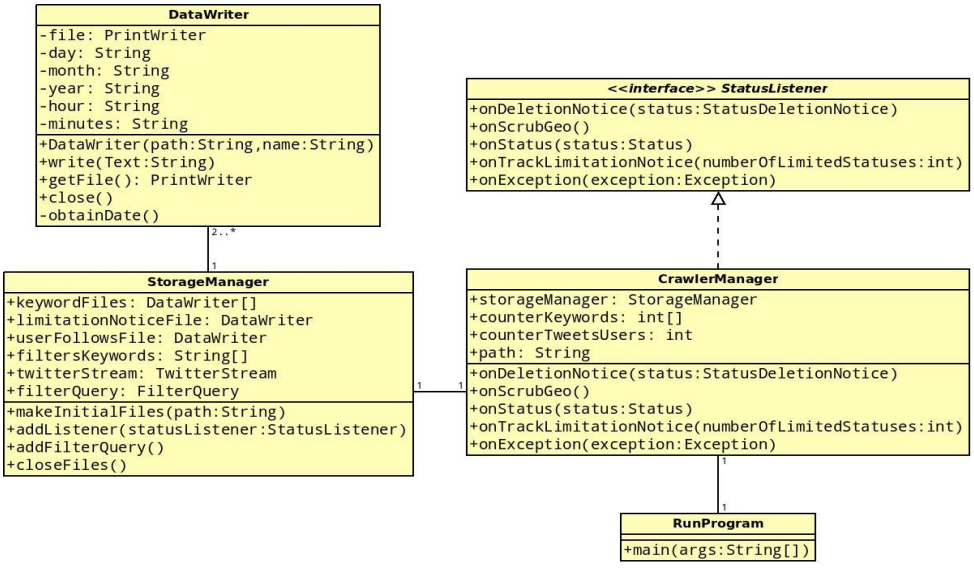

Figura 13: Modelo de clases UML del extractor

Como vemos, el extractor se compone de cuatro clases implementadas y una interface, StatusListener, propia de *Twitter4J*. A continuación se explican en detalle cada una de las clases y sus métodos implementados:

Clase DataWriter

Se encarga de escribir los ficheros en disco. Contiene los métodos necesarios para crear, escribir y cerrar un fichero de acuerdo al formato de nombrado decidido. Entre sus atributos, hay un String para indicar el año, mes, día, hora y minuto de creación del fichero y un objeto PrintWriter 'file' que contiene el puntero al fichero. Sus métodos más importantes son:

- DataWriter(String path, String name): Constructor de la clase. Escribe un fichero en la ruta indicada en path y cogiendo el nombre. De manera que el fichero tendrá el siguiente nombre: {path}/twitter_streaming_YYYY_MM_DD_hh_mm_{name}
- write(String text): Escribe el texto pasado por parámetro 'text' en el fichero 'file'.
- getFile(): Devuelve el objeto 'file'.
- close(): Cierra el fichero 'file'.
- obtainDate(): Obtiene la fecha y hora en la que se ejecuta el método y asigna ese valor a los atributos correspondientes.

Clase StorageManager

Clase que se encarga del almacenamiento y la inicialización de todos los ficheros. Posee un atributo filterQuery de tipo FilterQuery, en el que se indican los términos por los que filtrar la escucha, mediante el atributo keywords, que contiene una lista de palabras clave. Además, también contiene un objeto DataWriter por cada fichero de términos, otro para el seguimiento de usuarios y otro para escribir las limitaciones en el fichero de Limitation Notice. Además, cuenta con un objeto de tipo TwitterStream, que se encarga de gestionar el listener que queda a la espera de obtener nuevos eventos (actualizaciones de estado, etc).

Los métodos más destacables de la clase son los siguientes:

- makeInitialFiles(String path): Crea todos los ficheros de cada término, el fichero de seguimiento de usuarios y el fichero de LimitationNotice en la ruta pasada por parámetro en 'path' utilizando la clase DataWriter.
- addListener(StatusListener statusListener): Este método se encarga de inicializar el listener. El parámetro pasado, statusListener, se corresponde a nuestra clase CrawlerManager, ya que es la que implementa esa interface.
- addFilterQuery(): Se encarga de añadir los filtros al extractor y ejecutar el listener.

- closeFiles(): Cierra todos los ficheros de la instancia del extractor.

Clase CrawlerManager

Clase que implementa la interfaz StatusListener para responder a ciertos eventos. Además, inicializa una instancia del StorageManager ejecutando los métodos necesarios para ejecutar el extractor.

La clase CrawlerManager debe implementar los siguientes métodos heredados de StatusListener:

- onDeletionNotice(StatusDeletionNotice status): Se lanza cuando se borra un tweet que tiene alguno de nuestros filtros. Para el caso del extractor, este método no realiza ninguna acción.
- onScrubGeo(): Se lanza cuando se le obliga a un usuario a eliminar sus datos geográficos. Tampoco tiene ninguna acción asignada.
- onStatus(Status status): Se lanza cuando un usuario escribe un tweet que entra dentro de algunos de nuestros filtros. Para este caso, este método se encarga de comprobar si el tweet pertenece a alguno de los términos (en cuyo caso, se escribirá en el fichero de términos correspondiente) o al seguimiento de usuarios (en cuyo caso, se escribirá en el fichero de seguimiento de usuarios). Además, se encarga de comprobar cuándo un fichero ha llegado a su límite de tweets, establecido en 10,000. En ese caso, cerrará el fichero y creará uno nuevo vacío, con un nuevo timestamp diferente, en el que seguirá escribiendo.
- onTrackLimitationNotice(int numberOfLimitedStatuses): Se lanza cuando algunas actualizaciones de estado no han podido llegar por distintas limitaciones. Para este caso, se almacena un mensaje en el fichero twitter_streaming_LimitationNotice con el siguiente formato:

  “Got track limitation notice:" + numberOfLimitedStatuses

- onException(Exception e): Se lanza en caso de excepción del extractor, por ejemplo, un error de red. En este caso, el extractor escribe en un fichero de log, utilizando la librería *Log4J* [61].

Clase RunProgram

Es la clase principal del extractor. Se encarga de obtener los parámetros de entrada y ejecutar el extractor con estos. Los argumentos que recibe el parámetro es, en primer lugar, la ruta en la que se escribirán los ficheros, acto seguido, el fichero que contiene los ids de usuarios a los que segur y por último el fichero con los términos a seguir. Es decir, args[0] será la ruta en la que se escribirán los ficheros, args[1] será la ruta del fichero de texto donde leer los ids de los usuarios a seguir y args[2] será la ruta del fichero de texto donde leer los topics que seguirá el extractor.

3.1.4 Despliegue del extractor

Una vez desarrollado y probado el extractor, es hora de ponerlo en marcha. El objetivo es conseguir extraer la mayor cantidad de datos, y para ello, se ha decidido ejecutar el extractor en varias instancias, utilizando Cloud Computing y virtualización.

La herramienta de Cloud Computing utilizada es *Eucalyptus* [62]. Mediante esta herramienta, nos conectaremos a una máquina que funciona como front-end e instanciaremos las máquinas virtuales. En cada una de estas máquinas virtuales, ejecutaremos el extractor con distintos parámetros.

Una vez conectados al front-end, primero debemos ver que máquinas hay disponibles, para ello utilizamos:

```
euca-describe-images
```

En la salida, obtenemos la lista de imágenes disponibles, como vemos a continuación:

```
ccaballe@correcaminos:~$ euca-describe-images
IMAGE ami-00000011     natty-java/disk.img.manifest.xml        available
      private          x86_64     machine    aki-0000000f     ari-00000010
      instance-store
IMAGE ari-00000010     natty-java/initrd.img.manifest.xml
      available  private          x86_64     ramdisk
      instance-store
IMAGE aki-0000000f     natty-java/vmlinuz.manifest.xml         available
      private          x86_64     kernel                instance-store
```

La imagen que utilizaremos es la que tiene el identificador “ami-00000011”. Para lanzar las máquinas virtuales, utilizamos:

```
 euca-run-instances -k mykey -t m1.custom -n 5 ami-00000011
```

Con esto, instanciamos 5 máquinas virtuales. Cada una de estas máquinas ejecutará un extractor. Podemos ver el estado de las máquinas mediante el comando “euca-describe-instances”

```
euca-describe-instances
```

La salida que obtenemos es la siguiente:

```
ccaballe@correcaminos:~$ euca-describe-instances
```

```
RESERVATIONr-vu77y2l8 twitter    default
INSTANCE   i-000000c5 ami-00000011     163.117.148.183
     163.117.148.183pending mykey 1          m1.custom  2012-08-
28T17:59:42Z     nova  aki-0000000f     ari-00000010
INSTANCE   i-000000c6 ami-00000011     163.117.148.182
     163.117.148.182pending mykey 2          m1.custom  2012-08-
28T17:59:42Z     nova  aki-0000000f     ari-00000010
INSTANCE   i-000000c7 ami-00000011     163.117.148.185
     163.117.148.185pending mykey 3          m1.custom  2012-08-
28T17:59:42Z     nova  aki-0000000f     ari-00000010
INSTANCE   i-000000c8 ami-00000011     163.117.148.184
     163.117.148.184pending mykey 4          m1.custom  2012-08-
28T17:59:42Z     nova  aki-0000000f     ari-00000010
INSTANCE   i-000000c4 ami-00000011     163.117.148.181
     163.117.148.181pending mykey 0          m1.custom  2012-08-
28T17:59:41Z     nova  aki-0000000f     ari-00000010
```

Como vemos, el estado inicial de las máquinas es “pending” (excepto 163.117.148). Hay que esperar unos minutos a que el estado de la máquina sea “running” para poder utilizar las máquinas.

Una ver lanzadas las máquinas, se accedería a ellas mediante *ssh*.

```
ssh -i mykey.pem root@163.117.148.184
```

Con esto tendríamos las cinco máquinas que utilizaremos para lanzar los extractores en Cloud.

Para terminar las instancias, utilizaremos el identificador de la instancia y el comando “euca-terminate-instances”. Por ejemplo:

```
euca-terminate-instances i-000000b4
```

Ya que Twitter no permite varias conexiones de Streaming con las mismas credenciales. Es necesario crear varios usuarios para lanzar varias instancias, una en cada máquina virtual. Por tanto, nos registramos en https://dev.twitter.com/. Una vez registrados, podemos ver nuestras credenciales en dicha página, como se muestra en la figura 14[2].

2 Por motivos de seguridad, se han ocultado las credenciales utilizadas para el desarrollo del proyecto

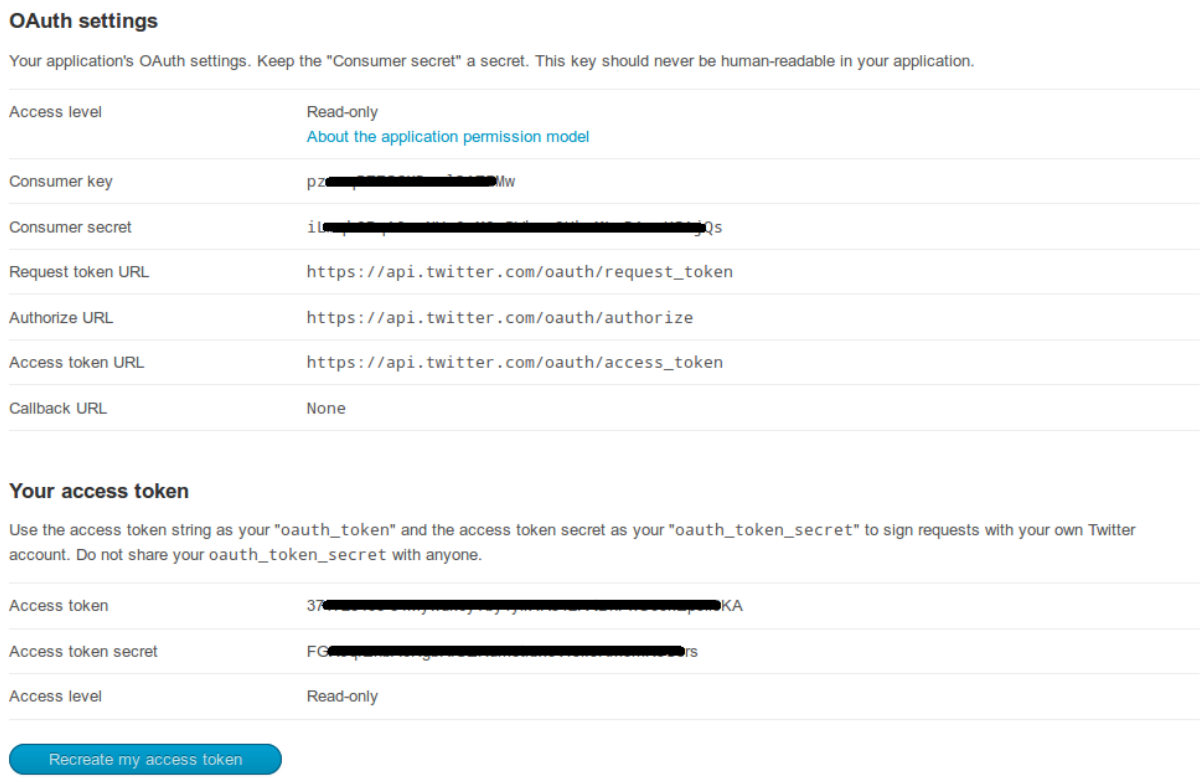

Figura 14: Credenciales de Twitter Developers

Con estos datos, podemos editar el fichero *twitter4j.properties*, que quedaría como sigue:

```
debug=true
oauth.consumerKey=pz****************Mw
oauth.consumerSecret=iL************************************Qs
oauth.accessToken=37********************************************KA
oauth.accessTokenSecret=FG************************************rs
```

Tendremos cinco ficheros *twitter4j.properties*, uno por cada usuario de *Twitter Developers* que hemos creado. Con esto, el siguiente paso es compilar el extractor y crear nuestro fichero ejecutable empaquetado en un *JAR* [63].

El objetivo es tener distintos ejecutables del extractor, cada uno dedicado a extraer una serie de topics y usuarios concretos.

Cada extractor se ocupará de un máximo de 400 topics, como marca la limitación de *Twitter*. Aunque ya comentamos que nuestra lista de términos estaba formada por 61 palabras, debemos sumarle a estos términos la lista de usuarios. Con esto se consigue obtener las menciones a un

usuario. Por tanto, nos quedará una lista de términos de 1695 + 61 = 1756. Por tanto, cuatro instancias del extractor seguirán 400 topics y la quinta seguirá 156. Con esto queda cubierto el seguimiento de términos.

En cuanto al seguimiento de usuarios, aunque la restricción de *Twitter*, de 5000 usuarios no nos afecta con los 1695 usuarios que queremos seguir, se ha decidido dividir la extracción de usuarios entre los extractores para balancear la carga de cada uno de ellos. Cada extractor se ocupará de 350 usuarios excepto el quinto, que se ocupará de 295.

Para el seguimiento a través de usuarios, es necesario saber el ID de cada cuenta que se desea seguir. Sin embargo, estas peticiones se hacen contra el API de REST, y existe una limitación de 350 peticiones a la hora. [64]

La solución adoptada fue obtener los IDs y almacenarlos de forma permanente, en cinco ficheros distintos; cuatro con 350 IDs de usuarios y otro con 295 IDs, coincidentes con los parámetros que se pasan al extractor.

Para empaquetar el proyecto en un fichero *JAR*, se utiliza el siguiente comando, situándonos en la raíz del proyecto:

```
jar -cvfm extractor.jar [ficheroDeManifiesto] -C [directorio con los .class]
.
```

En el fichero de manifiesto, *MANIFEST.MF*, se edita el classpath del proyecto para referenciar a las librerías externas y se indica cual es la clase que contiene el método main. Nuestro *MANIFEST.MF* tiene el siguiente contenido:

```
Manifest-Version: 1.0
Class-Path: . lib/log4j-1.2.16.jar lib/twitter4j-stream-2.2.5-SNAPSHOT.jar
lib/twitter4j-core-2.2.5-SNAPSHOT.jar
Main-Class: RunProgram
```

Para este caso, es necesario que las librerías *JAR* externas se encuentren en el directorio en que se encuentra el *extractor.jar*, dentro de un subdirectorio *lib/* . Además, en el mismo directorio en el que se encuentra el *extractor.jar* deberá existir un directorio *log/* en el que se utiliza la librería *log4J* para guardar logs de ejecución convenientemente. El fichero *log.properties* configurado convenientemente y, por supuesto, el fichero *twitter4j.properties* como hemos indicado anteriormente.

Cumpliendo esto pequeños prerrequisitos, el extractor ejecutaría con el siguiente comando:

```
java -jar [rutaDeFicheros] [ficheroDeIds] [ficheroDeTopics]
```

Teniendo todo esto y con la infraestructura de red montada, lo único que queda por hacer es ejecutar el extractor con distintas credenciales y parámetros en cada máquina. Una vez ejecutado, es

necesario monitorizar diariamente el estado del extractor, para re-lanzarlo en caso de caída y realizar copias de respaldo periódicas.

3.1.5 Análisis de la información extraída

En este apartado, se realizará un primer análisis de los datos extraídos del API de *Twitter*. Además, se va a organizar la información de los ficheros de manera que la información quede compacta.

El extractor estuvo en funcionamiento durante 60 días desde el 7 de Noviembre de 2011. Dado que las elecciones generales fueron el 20 de Noviembre, este intervalo de fechas nos permitió obtener información sobre las elecciones, antes, durante y después del día electoral.

El primer paso es compactar los ficheros. Ya que tenemos varios ficheros del tipo "*twitter_streaming_topic_YYYY_MM_DD_HH_MM*", vamos a agrupar el contenido de todos los ficheros con distintas fechas en un único fichero "*twitter_streaming_topic*" por cada topic que hemos seguido. Esto se hace de forma muy sencilla utilizando "*cat*" mediante el script que se muestra a continuación.

```
#! /bin/bash

# Terminos
while read line
do
        cat twitter_streaming_*_$line > twitter_streaming_$line
done < /home/ccaballe/listaTerminosUsuarios.txt

# Users Follows
cat twitter_streaming_*_UsersFollows > twitter_streaming_UsersFollows
```

El script lee los topics de un fichero que contiene los términos y usuarios y por cada uno, realiza la concatenación. Acto seguido, realiza la concatenación de los ficheros de *UsersFollows*.

Una vez compactados los ficheros, podemos analizarlos de forma más sencilla. El primer dato de interés es el tamaño total de la traza extraída. Sumando el tamaño de cada uno de los ficheros obtenidos de la operación anterior, vemos que el tamaño total de la traza es el siguiente:

$$Tamaño_{total} = \sum Tamaño_{ficheros} = 92{,}577{,}836\ KB \approx 89\text{GB}$$

Para ver el número de tweets que tiene la traza, simplemente podemos ver el número de líneas totales de todos los ficheros y sumarlas:

$$Nº\ de\ Tweets_{total} = \sum Nº\ de\ Tweets_{ficheros} = \sum Nº\ de\ lineas_{ficheros} = 50{,}863{,}414$$

El resultado es algo más de 50 millones de tweets, concretamente, 50,863,414. La ocupación media de cada tweet vendrá dado por la siguiente formula:

$$Ocupación_{media} = \frac{Ocupación\,total\,de\,la\,traza}{Número\,de\,tweets} = \frac{92{,}577{,}836\,B}{50{,}863{,}414} = 1.82\,KB$$

3.2 Configuración y puesta en marcha de Hadoop

En este capítulo hablaremos del trabajo realizado en cuanto a la configuración y utilización de *Hadoop*. Para el desarrollo del proyecto, se ha utilizado un cluster de 16 nodos.

3.2.1 Configuración de Hadoop

En este apartado se detalla la configuración de *Hadoop* para su ejecución en un cluster con hasta 16 nodos. Se describirán los requerimientos necesarias y la configuración de los principales ficheros de propiedades para una correcta instalación [65].

Prerrequisitos

Hadoop tiene únicamente dos prerrequisitos para cada uno de los nodos:

- Java debe estar instalado en una versión 1.6.x
- Debe estar instalado y configurado *ssh* para que los procesos de Hadoop puedan comunicarse de forma segura entre los nodos del cluster. Para ello, es necesario asegurarse de que podemos iniciar sesión sin contraseña en las máquinas del cluster, utilizando una clave pública.

Para esto último, hay que seguir dos sencillos pasos:

1. Generar la clave pública. Para ello, utilizamos el siguiente comando:

```
ssh-keygen -t rsa
```

Cuando nos pida la frase de paso (*passphrase*), debemos dejarlo en blanco. Obtendremos una salida similar a la siguiente:

```
cristian@cristian-HP-Pavilion-dv6-Notebook-PC:~$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/criswol/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/criswol/.ssh/id_rsa.
Your public key has been saved in /home/criswol/.ssh/id_rsa.pub.
The key fingerprint is:
c5:e6:57:bc:e2:57:3a:41:30:7c:db:80:16:5a:96:dc cristian@cristian-HP-
Pavilion-dv6-Notebook-PC
The key's randomart image is:
+--[ RSA 2048]----+
|           o*=   |
|          . +*+E  |
|           =. .++ |
|          +   o...|
|         S . o o .|
|            o . + |
|             . +  |
|              . . |
|                  |
+-----------------+
```

2. Guardar la clave pública en cada una de las máquinas del cluster utilizando:

```
ssh-copy-id usuario@maquina
```

Mediante estos dos comandos, debemos asegurar que se puede acceder desde cada nodo hacia los demás nodos del cluster.

Instalación de Hadoop

Para instalar *Hadoop*, basta con descargar el fichero comprimido tar.gz de la página de descargas de apache: http://hadoop.apache.org/common/releases.html

En este proyecto, se ha utilizado la versión 0.20.203.0. Dentro de ese fichero se encuentran todos los ficheros necesarios para correr y configurar *Hadoop*.

Configuración de Hadoop

A continuación, se muestran algunos de los más importantes ficheros de configuración que es necesario modificar.

- **core-site.xml**: Determina la url del *namenode*, además del puerto por el que se escuchan las peticiones de los clientes. También se configura cual es el directorio base para datos temporales. En el siguiente fichero, se indica que el *namenode* será el nodo "compute-0-1-13" y escuchará en el puerto 54310. Además, el directorio base para datos temporales será "/scratch/HDFS-ccaballe/tmp/".

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

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

<configuration>

<property>
        <name>fs.default.name</name>
        <value>hdfs://compute-0-1-13:54310</value>
</property>
<property>
        <name>hadoop.tmp.dir</name>
        <value>/scratch/HDFS-ccaballe/tmp/</value>
        <description>A base for other temporary directories.</description>
</property>

</configuration>
```

- **hadoop-env.sh:** Script que carga las variables de entorno que necesitemos para nuestra configuración.
- **hdfs-site.xml**: En este fichero se almacena, entre otros datos interesantes, la localización de los bloques de datos del *HDFS* utilizando la propiedad dfs.data.dir. En la siguiente configuración, se indica que los bloques se guarden en el directorio "/scratch/HDFS-ccaballe/data"

```
[...]
<property>
     <name>dfs.data.dir</name>
     <value>/scratch/HDFS-ccaballe/data</value>
     <description>Determines where on the local filesystem an DFS data
     node should store its blocks.  If this is a comma-delimited list of
     directories, then data will be stored in all named directories,
     typically on different devices. Directories that do not exist are
     ignored.
     </description>
```

```
</property>
[...]
```

- **mapred-site.xml:** Permite configurar la ubicación del *jobtracker*, entre otras.

```
[...]
<property>
  <name>mapred.job.tracker</name>
  <value>compute-0-1-13:54311</value>
  <description>The host and port that the MapReduce job tracker runs
  at.  If "local", then jobs are run in-process as a single map
  and reduce task.
  </description>
</property>
[...]
```

- **masters:** Se indica cual es nombre del host del nodo maestro.

```
[nombre_nodo_master]
```

- **slaves:** Se indica el nombre de los nodos esclavos.

```
[nombre_nodo_slave_1]
[nombre_nodo_slave_2]
[nombre_nodo_slave_3]
(...)
[nombre_nodo_slave_N]
```

3.2.2 Puesta en marcha de Hadoop

En los siguientes subapartados, vamos a ejecutar programas *MapReduce* de *Hadoop* sobre un cluster de 16 nodos. Este cluster no es únicamente *Hadoop*, sino que también esta soportado para dar otro tipo de servicios. En concreto, el cluster utiliza *TORQUE* [66] como planificador de trabajos en el cluster. *TORQUE* se compone de una máquina que funciona como front-end y N nodos que funcionan como máquinas de cómputo. El usuario únicamente lanzará el trabajo desde el front-end a algunas de las colas de trabajo de *TORQUE* [67] y demandará las máquinas que le sean necesarias

Por tanto, la estrategia que se ha seguido es realizar un script en bash que automáticamente haga todo el proceso para que los trabajos programados terminen correctamente, en concreto, este script se encargará de:

- Obtener la configuración indicada en los ficheros, como vimos en el apartado anterior.

- Cargar las variables de entorno necesarias
- Formatear el sistema de ficheros *HDFS*
- Comenzar la ejecución de los demonios encargados del sistema de ficheros y del *MapReduce*
- Copiar los ficheros de entrada a *HDFS*.
- Ejecutar las aplicaciones *MapReduce* y obtener los resultados
- Limpiar el entorno, esto es, detener todos los procesos de *Hadoop* y borrar todos los ficheros temporales.

Todo este proceso se realiza utilizando un script como el que se muestra a continuación:

```
#!/bin/bash

#PBS -q hadoop_jobs

HHOME="/opt/hadoop-0.20.203.0"
INPUT_DATA_FILE="/home/ccaballe/FilterTrace/data_result/FilterTrace"
CODE_JAR="/home/ccaballe/MRUserTweet.jar"
RESULT_PATH="/home/ccaballe/TweetsVSUsers_Filtered"
HDFS_INPUT_FILE="/ccaballe/input"
HDFS_OUTPUT_PATH="/ccaballe/tweets_users_filtered"

echo "Setting nodes for Hadoop . . ."

cat ${PBS_NODEFILE} |  sed -e "s/ /\n/g" | uniq > $HOME/hadoop/conf/slaves
cat ${PBS_NODEFILE}

head -n 1 $HOME/hadoop/conf/slaves > $HOME/hadoop/conf/masters

for machine in $(cat $HOME/hadoop/conf/slaves)
do
      echo "Recreate HDFS on "$machine
      ssh $machine "rm -rf /scratch/HDFS-ccaballe"
      ssh $machine "mkdir /scratch/HDFS-ccaballe"
done

echo "Update config files"
export HADOOP_MASTER=`cat ~/hadoop/conf/masters`

echo "Master: "$HADOOP_MASTER
sed 's/HOST/'${HADOOP_MASTER}'/g' $HOME/hadoop/conf/core-site.xml.template >
$HOME/hadoop/conf/core-site.xml
sed 's/HOST/'${HADOOP_MASTER}'/g' $HOME/hadoop/conf/mapred-site.xml.template >
$HOME/hadoop/conf/mapred-site.xml

echo "Loading hadoop environment variables . . ."
export JAVA_HOME=/usr/lib/jvm/java-6-sun
export HADOOP_OPTS=-server
export HADOOP_NAMENODE_OPTS="-Dcom.sun.management.jmxremote
```

```
$HADOOP_NAMENODE_OPTS"
export HADOOP_SECONDARYNAMENODE_OPTS="-Dcom.sun.management.jmxremote
$HADOOP_SECONDARYNAMENODE_OPTS"
export HADOOP_DATANODE_OPTS="-Dcom.sun.management.jmxremote
$HADOOP_DATANODE_OPTS"
export HADOOP_BALANCER_OPTS="-Dcom.sun.management.jmxremote
$HADOOP_BALANCER_OPTS"
export HADOOP_JOBTRACKER_OPTS="-Dcom.sun.management.jmxremote
$HADOOP_JOBTRACKER_OPTS"
export HADOOP_SLAVES=/home/${USER}/hadoop/conf/slaves
export HADOOP_LOG_DIR=~/hadoop/logs

echo "Formating HDFS . . ."
echo "Y" | $HHOME/bin/hadoop --config $HOME/hadoop/conf namenode -format
echo "Starting HDFS and Map-reduce . . ."
$HHOME/bin/start-dfs.sh --config $HOME/hadoop/conf
$HHOME/bin/start-mapred.sh  --config $HOME/hadoop/conf
sleep 60

echo "Status"
$HHOME/bin/hadoop --config $HOME/hadoop/conf dfsadmin -report

echo "Copying file request to HDFS"
ls -las $INPUT_DATA_FILE
$HHOME/bin/hadoop --config $HOME/hadoop/conf dfs -copyFromLocal
$INPUT_DATA_FILE $HDFS_INPUT_FILE
$HHOME/bin/hadoop --config $HOME/hadoop/conf dfs -ls $HDFS_INPUT_FILE
sleep 10

echo "Run the hadoop LP job"
echo "========================================="
$HHOME/bin/hadoop --config $HOME/hadoop/conf jar $CODE_JAR $HDFS_INPUT_FILE
$HDFS_OUTPUT_PATH

echo "Get results"
echo "========================================="
rm -r $RESULT_PATH
mkdir -p $RESULT_PATH/data_result
$HHOME/bin/hadoop --config $HOME/hadoop/conf dfs -ls $HDFS_OUPUT_PATH
$HHOME/bin/hadoop --config $HOME/hadoop/conf job -history all $HDFS_OUTPUT_PATH
> $RESULT_PATH/job.out
$HHOME/bin/hadoop --config $HOME/hadoop/conf dfs -getmerge $HDFS_OUTPUT_PATH
$RESULT_PATH/data_result

echo "Shutting down"
$HHOME/bin/stop-mapred.sh
$HHOME/bin/stop-dfs.sh

for machine in $(cat $HOME/hadoop/conf/slaves)
do
      echo "Cleaning "$machine
      ssh $machine "rm -rf /scratch/HDFS-ccaballe"
      ssh $machine "rm -rf /tmp/hadoop-*"
      ssh $machine "rm -rf /tmp/Jetty_0_0_0_0_500*"
      ssh $machine "rm -rf /tmp/hsperfdata_ccaballe"
```

```
      ssh $machine "rm -rf /tmp/reduce_toPlot_*"
      ssh $machine "ls -las /tmp/ | grep ccaballe"
      ssh $machine "killall java"
      echo "----------------------"
done
```

Al inicio del script, se indica que el trabajo se asigne a la cola de trabajos *hadoop-jobs*. Para ejecutar este script desde el cluster, utilizamos el comando "*qsub*" para suscribir el trabajo a la cola e indicamos cuantos nodos vamos a utilizar con "-l". Por ejemplo, para ejecutar el trabajo con dos nodos, ejecutamos:

```
qsub launchHadoopProgram.sh -l nodes=2
```

El script se encarga automáticamente de elegir los nodos *master* y *slaves* que se usarán entre el número de nodos elegidos, escribiendo en los ficheros de configuración los nombres correspondientes. El usuario solo deberá preocuparse de modificar las variables globales en las que indicamos el directorio de entrada, el fichero *JAR* con el código del trabajo, el directorio donde se guardará la salida del trabajo y los directorios que se utilizarán en *HDFS*, entre otros aspectos básicos de configuración.

Ya que los procesos de copia con el *HDFS* pueden llegar a ser bastante costosos en tiempo, cabe destacar que, para el caso en el que se van a ejecutar varias tareas, se podría realizar una pequeña modificación en el script para evitar trabajo repetido. Un escenario típico de esto es cuando la salida de un trabajo *MapReduce* es la entrada que tomará otro trabajo *MapReduce*. Para este caso, en lugar de ejecutar el script dos veces, volviendo a cargar el entorno, etc, se añadiría el fragmento de código de ejecución del segundo trabajo antes de el apagado y limpieza del entorno.

3.3 Filtrado de la traza mediante Hadoop

En este apartado detallaremos el trabajo de filtrado de la traza. Se ha filtrado la traza de forma que se han dejado únicamente los tweets que interesan. Es evidente que en el proceso de extracción se han extraído tweets que para nuestro estudio son despreciables.

Un ejemplo claro es el seguimiento de el término "*pp*" (refiriéndonos al partido popular). Sin embargo, 50,863,414 existen multitud de palabras, sobre todo en inglés, con "*pp*" como subcadena. [68]

Para nuestro estudio, nos interesan solo los tweets que cumplan las siguientes dos condiciones:

- El idioma del texto debe ser español
- El usuario que escribe el tweet debe ser alguno de los usuarios de la lista, o este debe estar mencionado en el tweet.

Los filtrados se han realizado mediante programas MapReduce escritos en Java utilizando Hadoop como plataforma distribuida de procesamiento de datos.

3.3.2 MapReduce para el filtrado por idioma

Mediante un programa *MapReduce* utilizando *Hadoop* filtraremos la traza para quedarnos únicamente con los tweets escritos en español. Por tanto, necesitamos de algún mecanismo que, dado un texto determinado, detecte su idioma.

Para esto, se ha elegido utilizar la librería *langDetect* [69] de Java. Esta librería se caracteriza por tener un 99% de fiabilidad, además de ser rápida y ser capaz de detectar el idioma del texto entre más de 50 idiomas. Para determinar su idioma, utiliza análisis de frecuencias de las palabras.

Para obtener el idioma de un texto solo hay que realizar lo siguiente:

```
        DetectorFactory.loadProfile(profileDirectory);
        Detector detector = DetectorFactory.create();
        detector.append("Texto a detectar el idioma");
       return detector.detect();
```

El fragmento anterior devuelve una cadena de texto del tipo "*en*", "*es*", "*fr*" para indicar que un texto está escrito en inglés, español o francés respectivamente. Dado que solo nos interesa el caso en el que los tweets están escritos en español, se ha escrito el siguiente método, que comprueba si el texto pasado por parámetro es español:

```
public boolean isSpanish(String text){
           try{
                 detector = DetectorFactory.create();
                 detector.append(text);
                 String lang = detector.detect();
                 if(lang.equals("es")){
                       return true;
                 }
                 return false;
           }
           catch (LangDetectException e){
                 return false;
           }
     }
```

El método recibe como parámetro una cadena determinada y obtiene su idioma. Solo en el caso de que el idioma detectado sea español ("es") el método devuelve true. En caso contrario, o en caso de lanzarse una excepción al detectar el idioma, el método devuelve false.

Como todos los programas *MapReduce* desarrollados en este proyecto, existen un mínimo de

tres clases para cada programa *MapReduce*. Para este caso son las siguientes:

MapperSpanishFilter

Contiene las funciones que realiza cada tarea *Map*. Para este caso, el pseudocódigo de la función *Map* es la siguiente:

```
FUNCION Map(clave, valor)
      tweet ← leerLinea()
      SI esTweet(tweet)
            texto ← capturarTexto(tweet)
            SI esEspañol(texto)
                  usuario ← obtenerUsuario(tweet)
                  emitirContexto(usuario, tweet)
                  Terminar()
            FIN SI
      FIN SI
FIN FUNCION
```

El algoritmo comprueba en primer lugar que el tweet tenga el formato adecuado (acorde al ya definido anteriormente). Acto seguido, comprueba si el texto del tweet es español, y en caso afirmativo, obtiene el usuario que escribió el tweet para agruparlos. Es decir, cada tarea *Map* creará un par (“*usuario*”, “*tweet*”), que después se agruparán en (“*usuario*”, “*lista(tweets)*”).

ReducerSpanishFilter:

Contiene las funciones que realiza cada tarea *Reduce*. Para este caso concreto, solamente debe escribir lo obtenido del paso anterior en un fichero, como se muestra en el siguiente pseudocódigo.

```
FUNCION Reduce(clave, lista(valores))
      Para Cada valor de lista(valores)
            emitirContexto(clave, valor)
      FIN Para Cada
FIN FUNCION
```

DriverSpanishFilter:

Clase principal del programa *MapReduce*. Contiene parámetros de configuración, como tipos de salida, ruta de la entrada etc. El siguiente código Java se muestra como ejemplo, ya que es bastante

intuitivo.

```
public static void main(String[] args) throws Exception {

            Configuration conf = new Configuration();

            Job job = new Job(conf, "spanishFilter");
            // Indicamos tipos de datos de salida
            job.setOutputKeyClass(Text.class);
            job.setOutputValueClass(Text.class);
            // Clases utilizadas para los Map y Reduce
            job.setMapperClass(MapperSpanishFilter.class);
            job.setReducerClass(ReducerSpanishFilter.class);
            // Formatos de entrada y salida
            job.setInputFormatClass(TextInputFormat.class);
            job.setOutputFormatClass(TextOutputFormat.class);
            // Rutas de la entrada y salida del programa
            FileInputFormat.addInputPath(job, new Path(args[0]));
            FileOutputFormat.setOutputPath(job, new Path(args[1]));


            job.setJarByClass(DriverSpanishFilter.class);
            job.waitForCompletion(true);

      }
```

El siguiente esquema muestra el flujo de ejecución de esta tarea *Hadoop* para la detección de idioma:

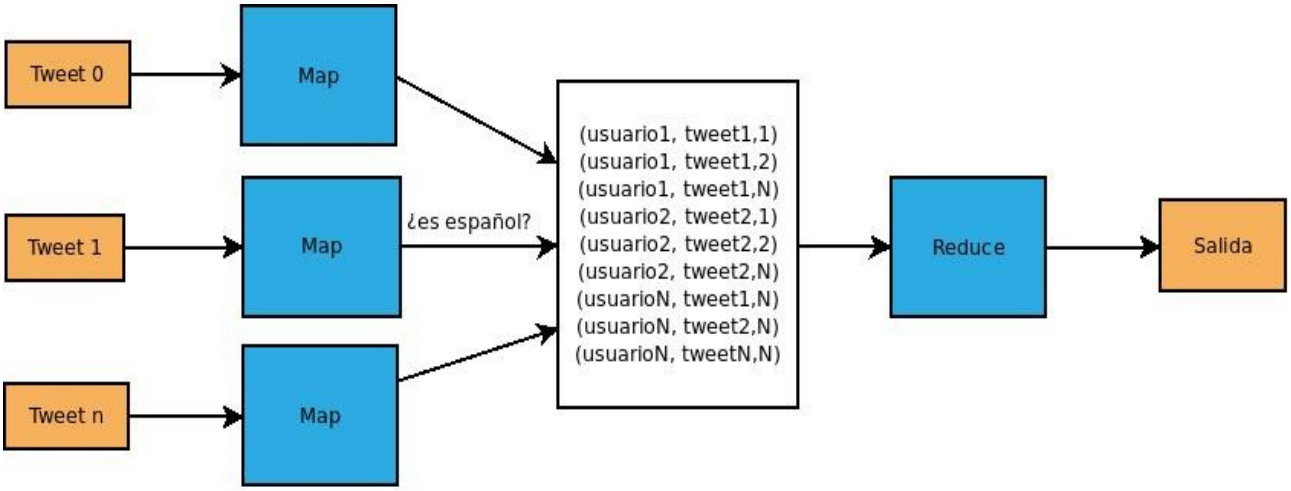

Figura 15: Flujo de datos MapReduce para el filtrado de idioma

Para este caso concreto en el que el Reduce no tiene ninguna función, la salida del programa será un fichero con todos los tweets obtenidos del procesamiento de las tareas *Map* ordenados alfabéticamente por usuarios.

3.3.3 MapReduce para el filtrado por usuario y menciones

Una vez filtrada la traza por idioma, el siguiente paso para terminar la limpieza de la traza antes de sacar estadísticas concretas es filtrar por usuarios y menciones de usuarios. Por tanto, esta tarea debe recoger como datos de entrada los datos obtenidos anteriormente.

Este programa de filtrado tiene características muy similares al anterior con respecto a la implementación. La función *Reduce* es exactamente idéntica ya que el objetivo es escribir los tweets a un fichero, sin realizar ningún calculo con ellos.

La función *Map* es la que se encarga de decidir que tweets se escriben o no. Se deben comprobar dos condiciones por separado.

1. El tweet pertenece a uno de los usuarios de nuestra lista de interés.
2. El tweet menciona a uno de los usuarios de nuestra lista de interés.

Si se da la primera condición, no es necesario evaluar la segunda, puesto que el tweet ya cumple una condición para ser filtrado. El siguiente fragmento muestra la implementación este algoritmo implementado en el *Map* en pseudocódigo:

```
FUNCION Map(clave, valor)
      tweet ← leerLinea()
      usuario ← obtenerUsuario(tweet)
      SI listaUsuarios.contiene(usuario) ENTONCES
            EscribirContexto (usuario, tweet)
            Terminar()
      SI NO
            menciones[] ← obtenerMenciones(tweet)
            Para Cada M de menciones
                  SI listaUsuarios.contiene(M)
                        EscribirContexto(usuario, tweet)
                        Terminar()
                  FIN SI
            FIN Para Cada
      FIN SI
FIN FUNCION
```

Como podemos ver, lo primero que comprobamos es si el usuario del tweet es uno de la lista, en caso afirmativo, se escribe el contexto y se termina el algoritmo. En otro caso, se debe mirar, por cada mención del tweet (si las hay) si alguna de estas se corresponde a alguno de los usuarios de la lista.

En caso afirmativo, se escribe el contexto y se termina.

En la siguiente figura vemos el flujo de datos, como podemos comprobar, similar al filtrado por idioma que vimos en el apartado anterior. Cada tarea *Map* creará, una vez más, un par ("*usuario*", "t*weet*"), que después se agruparán en ("*usuario*", "lista(t*weets*)")

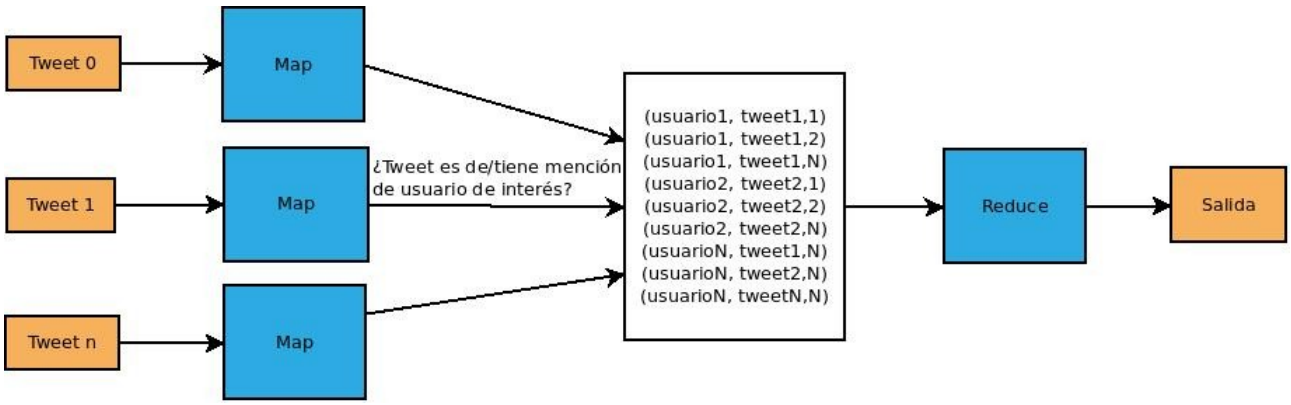

Figura 16: Flujo de datos MapReduce para el filtrado por usuarios y menciones

3.3.4 Resumen de los resultados obtenidos

La siguiente tabla resumen de forma clara los resultados obtenidos después de cada uno de los filtrados, tanto en número de tweets como en ocupación en disco.

Tipo de Traza	Ocupación (GB)	Ocupación (KB)	Nº de tweets	Ocupación media tweet (KB)	Relación de filtrado
Completa	~89	92,577,836	50,863,414	1.82	1
Filtrada por idioma	~30	30,773,252	16,156,836	1.90	0.33%
Filtrada por términos y menciones	~19	19,438,428	9,967,583	1.95	0.20%

Tabla 7: Resumen de los resultados obtenidos del filtrado de la traza

La columna de ocupación media se ha obtenido dividiendo la ocupación entre el número de tweets, mientras que la columna de relación de filtrado se ha obtenido aplicando la siguiente formula:

$$1-\left(\frac{Ocupacion_{filtrada}}{Ocupacion_{completa}}\right)$$

Como vemos, una vez concluido el filtrado completo de la traza, podemos observar que nuestra traza útil tiene un tamaño de 19 GB aproximadamente, es decir, un 20% de la traza total extraída. También podemos observar como aproximadamente un 33% de la traza se corresponden con tweets escritos en español. A continuación, se muestra gráficamente la variación de tamaño de las trazas.

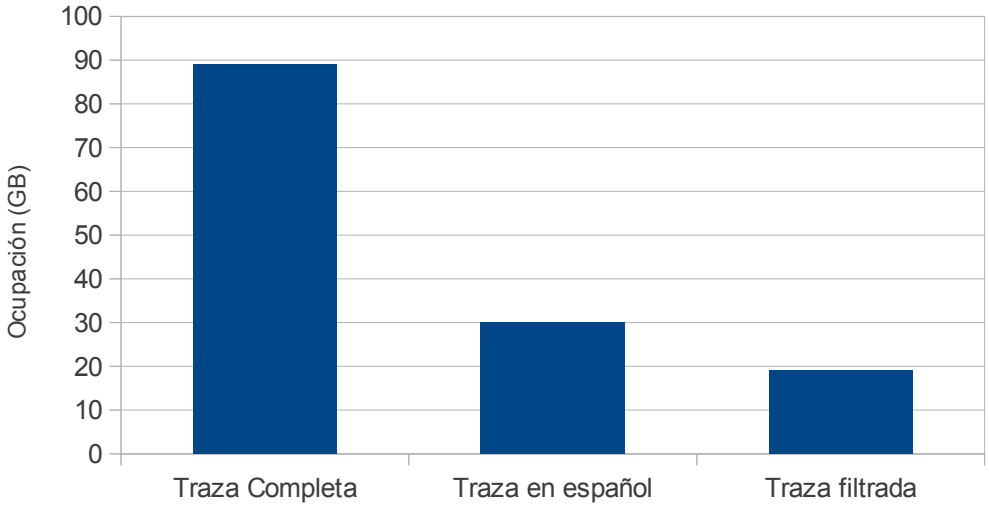

Figura 17: Relación del tamaño de las trazas completa y filtradas

3.4 Análisis de los datos mediante Hadoop

Una vez filtrada la traza, vamos a analizarla utilizando *Hadoop* una vez más. La idea de este hito es obtener resultados estadísticos que sirvan de aplicación práctica. En nuestro caso, obtendremos resultados acerca de los tweets escritos con temática de las elecciones. Se analizarán datos para sacar conclusiones como cual es el usuario que más tweets tiene o el día que más tweets se recibieron, entre otras.

Se puede comprobar que este tipo de análisis puede tener un gran interés para estudios de mercado y tendencias. En los siguientes apartados, detallaremos la implementación de estos programas y los resultados obtenidos. En concreto, se han desarrollado tres aplicaciones *MapReduce*:

- *MapReduce* para obtener el total de tweets escritos por cada día
- *MapReduce* para obtener el total de tweets escritos por usuario
- *MapReduce* para obtener los tweets de cada usuario en cada uno de los días

3.4.1 MapReduce para obtener los tweets por fecha

Un primer estudio nos lleva a analizar como se distribuye la traza a lo largo del tiempo, es decir, ver cuantos tweets se han escrito en cada día. De esta forma, podemos detectar comportamientos atípicos y analizar estos. Por ejemplo, es posible que un pico muy bajo venga dado por un fallo del extractor, o un día con baja actividad política mientras que un pico muy alto nos indica que es un día con alta actividad política.

A priori, se puede ver que la definición del problema se adapta perfectamente al paradigma de *MapReduce*. El siguiente fragmento de pseudocódigo muestra lo más destacable de la clase *Mapper* de este proyecto.

```
FUNCION Map(clave, valor)
      tweet ← leerLinea()
      fecha ← obtenerFecha(tweet)
      emitirContexto(fecha, 1)
      Terminar()
FIN FUNCION
```

El método *obtenerFecha* recibe como parámetro el tweet (linea del split de entrada) y obtiene la fecha mediante el campo "*Created at*" que definimos al describir el modo de almacenamiento del extractor (ver tabla 7). El campo '*valor*' no interesa en el *Mapper*, por tanto, contiene un objeto de tipo *IntWritable*, (objeto de tipo entero propio de *Hadoop*) con valor 1. Esto es así por cuestiones de ahorro de memoria (un *IntWritable* ocupa menos que, por ejemplo, un tipo *Text*).

La función *Reduce*, a diferencia de los programas de filtrado de la traza, si cumple una función importante. Una vez terminados los mapeos, tendremos pares del tipo ("*fecha*", lista(*valores*)) donde fecha será la clave. Puesto que hemos mapeado cada día de la fecha con un valor para cada tweet de entrada, lista(*valores*) tendrá tantos elementos como tweets existan con esa fecha. El pseudocódigo aproximado del *Reducer* es el siguiente:

```
FUNCION Reduce(clave, lista(valores))
      contadorTweets ← 0
      Para cada Valor de lista(valores)
            contadorTweets ← contadorTweets + 1
      FIN Para Cada
      emitirContexto(clave, contadorTweets)
      Terminar()
FIN FUNCION
```

Como vemos, únicamente se recorren los distintos elementos para ir sumando a un contador, de esta forma, el valor del contador al salir del bucle for tendrá el número de tweets con la fecha determinada en la clave del Reduce. Por último, se propaga la misma clave de entrada del *Map*, quedando un resultado del tipo: "día" "nº de tweets"

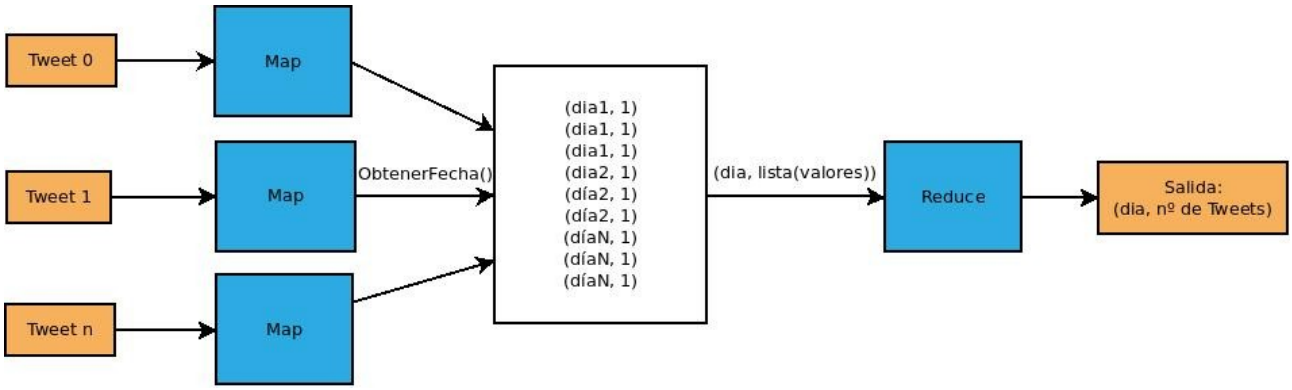

Figura 18: Flujo de datos MapReduce para obtener los tweets por fecha

Ya que este programa nos muestra la distribución de la traza a lo largo del tiempo, se ha ejecutado tanto para la traza completa como para la traza filtrada para comparar la variación entre estas. Los resultados obtenidos se muestran a continuación.

Fecha	Traza completa	Traza Filtrada
07/11/11	1089702	669670
08/11/11	991694	430060
09/11/11	817149	327344
10/11/11	785848	300608
11/11/11	620996	199795
12/11/11	564408	179176
13/11/11	645804	228419
14/11/11	636211	155776
15/11/11	224	24
16/11/11	8	0
17/11/11	201297	69944
18/11/11	970195	328899
19/11/11	653438	149137
20/11/11	1687240	685574
21/11/11	1134460	437235
22/11/11	697141	186475
23/11/11	707927	162077
24/11/11	684648	156237
25/11/11	702962	170328
26/11/11	526834	93757
27/11/11	0	0
28/11/11	221185	54536
29/11/11	1419753	279237
30/11/11	1418035	273826
01/12/11	1468133	273101
02/12/11	1418624	287457
03/12/11	1152822	175168
04/12/11	453001	146084
05/12/11	1135667	188583
06/12/11	714034	110943
07/12/11	301960	48121
08/12/11	706132	101721
09/12/11	697065	99198
10/12/11	693752	98468
11/12/11	741467	96025

Fecha	Traza completa	Traza Filtrada
12/12/11	254146	47316
13/12/11	882253	123076
14/12/11	833564	119816
15/12/11	333708	49000
16/12/11	0	0
17/12/11	154838	25965
18/12/11	655751	101931
19/12/11	920775	225535
20/12/11	825848	160550
21/12/11	854494	170223
22/12/11	804451	125007
23/12/11	749015	96898
24/12/11	400444	37099
25/12/11	0	0
26/12/11	0	0
27/12/11	575090	63287
28/12/11	1168913	112649
29/12/11	1193706	109050
30/12/11	1339049	160776
31/12/11	1223592	104194
01/01/12	0	0
02/01/12	856689	86243
03/01/12	1474261	136310
04/01/12	1388813	132728
05/01/12	1263753	119397
06/01/12	542330	24251
07/01/12	0	0
08/01/12	0	0
09/01/12	0	0
10/01/12	0	0
11/01/12	595749	72653
12/01/12	1549290	135029
13/01/12	1659840	137656
14/01/12	1447608	108944
15/01/12	255623	19086

Tabla 8: Resultados obtenidos del análisis de la traza a lo largo del tiempo

La tabla anterior muestra los datos de los tweets extraídos durante el tiempo de ejecución del extractor (desde el 7 de Noviembre de 2011 hasta el 15 de Enero de 2012, ambos incluidos. Se puede ver como en la tabla anterior existen días en los que no se ha extraído ningún Tweet. Esto es debido a que esos días no se realizó extracción de datos debido a problemas de red e infraestructura. El extractor falló los días 15, 16 y 27 de Noviembre, 16, 25 y 26 de Diciembre y 1, 7 y 8 de Diciembre. Las causas de los fallos fueron, en mayor medida, problemas de red y las consecuencias fueron la parada de extracción de datos o una extracción de datos despreciable.

La siguiente figura muestra la distribución de la traza completa a lo largo del tiempo de extracción, que va desde el 7 de Noviembre de 2011 hasta el 15 de Enero de 2012. Para una mejor visualización, se han eliminado de la gráfica los días que no se realizó extracción de datos.

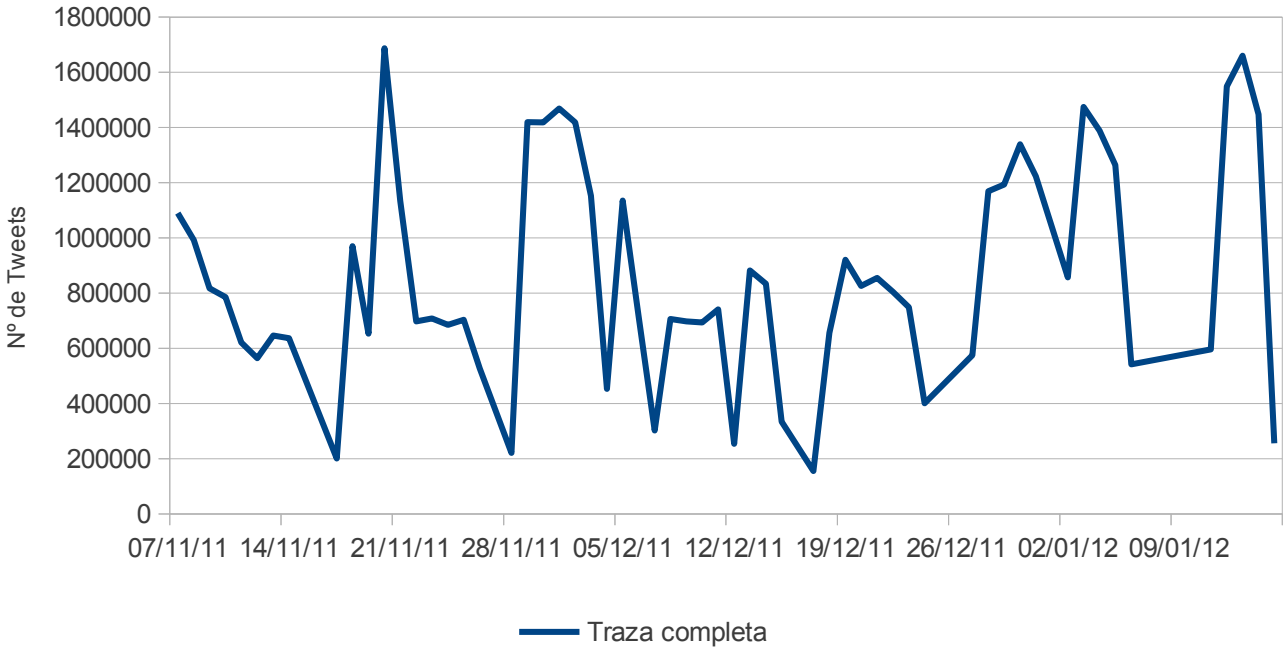

Figura 19: Distribución de la traza completa a lo largo del tiempo de extracción

Antes de realizar un análisis estadístico en profundidad, ya podemos observar los picos superiores el 20 de Noviembre de 2011 y el 13 de Enero de 2012. El primer valor es esperado, puesto que se corresponde con el día de las elecciones, mientras que el 13 de Enero tiene un valor bastante atípico, y, a priori, no se esperaba que tuviera un valor tan alto puesto que se llevan más de 40 días de diferencia.

Para nuestro estudio, nos interesa ver los días que se twittea más o menos de lo "normal". Para calcular cuales son los días que se salen del rango, necesitamos calcular la desviación estándar de la distribución. La desviación estándar [70] mide como se separan los datos con respecto a la media aritmética.

Calculando la media y la desviación estándar, podemos establecer un intervalo superior, que vendrá dado de suma de las dos. El intervalo inferior se calcula mediante la resta. Los datos que serán de interés de estudio serán los que estén por encima del intervalo superior y los que estén por debajo del intervalo inferior. El conjunto de formulas a utilizar son las siguientes:

$$Media\ aritmética = \bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i ;$$

$$Varianza = S^2 = \frac{1}{n}\sum_{i=1}^{n} (x_i - \bar{x})^2 ;$$

$$Desviación\ típica = \sigma = \sqrt{S^2} ;$$

$$Intervalo_{superior} = \bar{x} + \sigma ;$$

$$Intervalo_{inferior} = \bar{x} - \sigma;$$

En resumen, los puntos a estudiar serán:

$$Punto\ de\ estudio = x_i;\ SI\ x_i \geq Intervalo\ superior\ ó\ x_i \leq Intervalo\ inferior$$

Realizando los cálculos, el intervalo superior es, aproximadamente 1,258,847 mientras que el intervalo inferior es 456,268. La siguiente figura muestra gráficamente los limites superior e inferior de la traza:

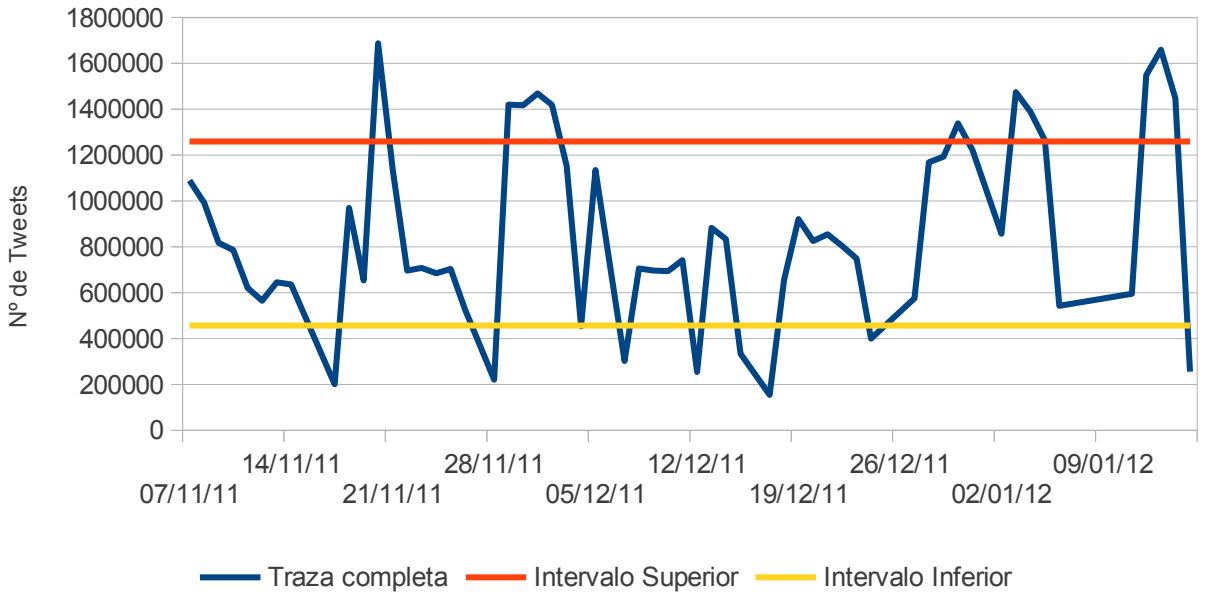

Figura 20: Intervalos superior e inferior de la traza completa

Viendo la figura, ya podríamos localizar cuales son los días que se ha twitteado más o menos de "lo normal", por lo que se podría realizar algún tipo de análisis de tendencias, estudios de mercado, etc. Los resultados obtenidos son los siguientes:

- **Días con menos tweets**: 17/11/11, 28/11/11, 04/12/11, 07/12/11, 12/12/11, 15/12/11, 17/12/11, 24/12/11, 15/01/12.
- **Días con más tweets**: 20/11/11, 29/11/11, 30/11/11, 01/12/11, 01/12/11, 30/12/11, 03/01/12, 04/01/12, 05/01/12, 12/01/12, 13/01/12, 14/01/12

Como conclusión a estos datos, vemos que existen 7 valores por debajo del límite inferior y 12

valores por encima del límite superior, de un total de 60 valores que forman la distribución. A partir de aquí se podría realizar un análisis individual por cada día para determinar las posibles causas de esos picos en la distribución de la traza. Por ejemplo, el día 20/11/11 la causa clara fue las elecciones generales, mientras que para el día 13/01/12 (día con más tweets después del 20/11/11) las causas pudieron ser diversas [71].

A continuación, se muestra la distribución de la traza filtrada, junto a sus límites superiores e inferiores:

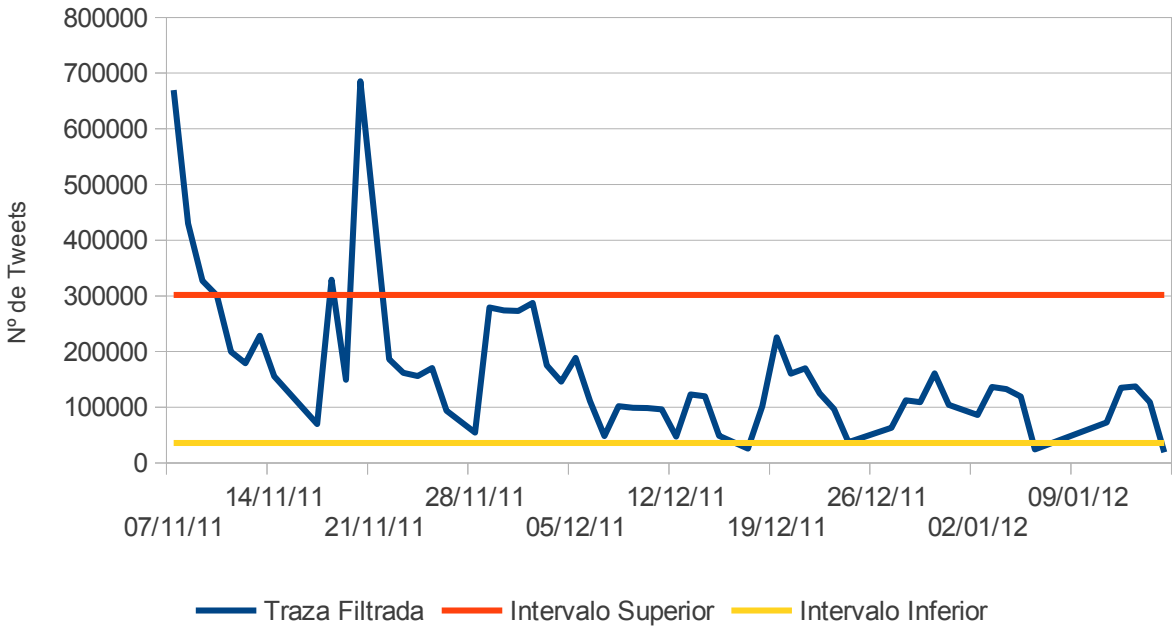

Figura 21: Distribución de la traza filtrada a lo largo del tiempo de extracción

Una vez filtrada la traza, vemos que la información obtenida si que es coherente con los datos buscados. Podemos observar que existen dos días en los que los valores son atípicos con respecto a los demás, ya que son más del doble del límite superior. Además, vemos que los otros días que salen fuera de los límites definidos en los intervalos superior e inferior lo hacen con una diferencia mínima. Los días 7 de Noviembre y 20 de Noviembre son los días de interés de estudio con respecto a esta distribución. Respecto al día 7, es muy probable que la causa por las que ese día se recogió tanta información útil es, con bastante probabilidad, la celebración del debate cara a cara entre los dos candidatos principales a las elecciones [72].

En la siguiente tabla se resumen los distintos resultados estadísticos recuperados hasta el momento:

Traza	Varianza	Media	Desviación típica	Intervalo superior	Intervalo inferior
Completa	157,418,188,707	862,087	396,759	1,258,847	456,268
Filtrada	17,677,880,490	168,943	132,958	301,901	35,984

Tabla 9: Resultados estadísticos de las trazas completa y filtrada

Para concluir este análisis, se indica la distribución temporal de la traza completa y filtrada en una misma imagen, de tal forma que se puede ver de forma sencilla la variación de tamaño entre el total de tweets extraídos y los tweets de interés para el estudio. Como indicamos en la tabla 7, es únicamente alrededor del 20%.

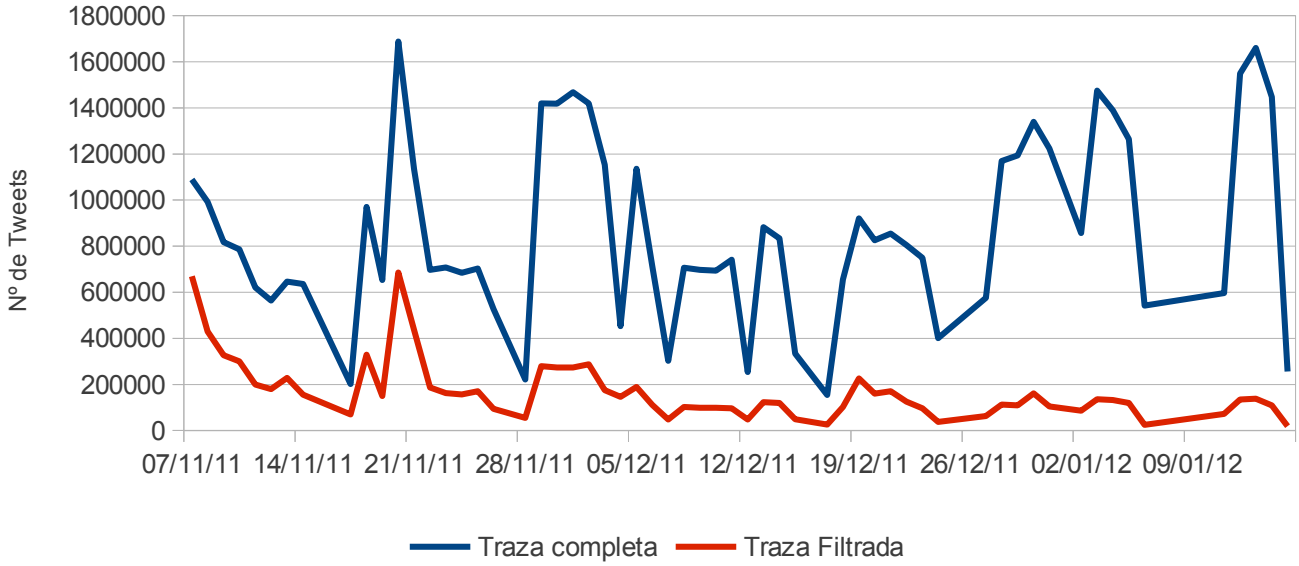

Figura 22: Relación de la distribución de las trazas completa y filtrada a lo largo del tiempo

3.4.2 MapReduce para obtener los tweets por usuario

De manera similar al apartado anterior, vamos a analizar los tweets escritos por cada usuario, de tal forma que podamos obtener distinta información de estudio como puede ser el usuario que más tweets ha escrito, o un análisis más detallado dependiendo del tipo de usuario.

Se ha escrito un programa *MapReduce* que obtiene el número de tweets escritos por cada usuario. Una vez obtenido el resultado, es fácil filtrarlo por un usuario concreto utilizando "*grep*". Filtrando los usuarios, podemos realizar estudios concretos de los usuarios que teníamos listados, o, como trabajo futuro, se podría analizar algún otro usuario que no se haya tenido en cuenta en la lista inicial.

El código del *Reduce* será idéntico al del apartado anterior, puesto que su única función es obtener el sumatorio de los elementos de un conjunto. El pseudocódigo del *Map* quedaría similar al siguiente fragmento:

```
FUNCION Map(clave, valor)
      tweet ← leerLinea()
      usuarioJSON ← obtenerUsuarioJSON(tweet)
      screenName ← getScreenNameUsuario(usuarioJSON)
      emitirContexto(screenName, 1)
      Terminar()
FIN FUNCION
```

El funcionamiento es el siguiente: primero se obtiene el texto completo del JSON del usuario. Acto seguido, se obtiene el "*screenName*", que se corresponde con el nick o alias del usuario. Por último, se propaga el par clave-valor, siendo la clave el "*screenName*" obtenido y el valor, un *IntWritable*(como veíamos en el apartado anterior).

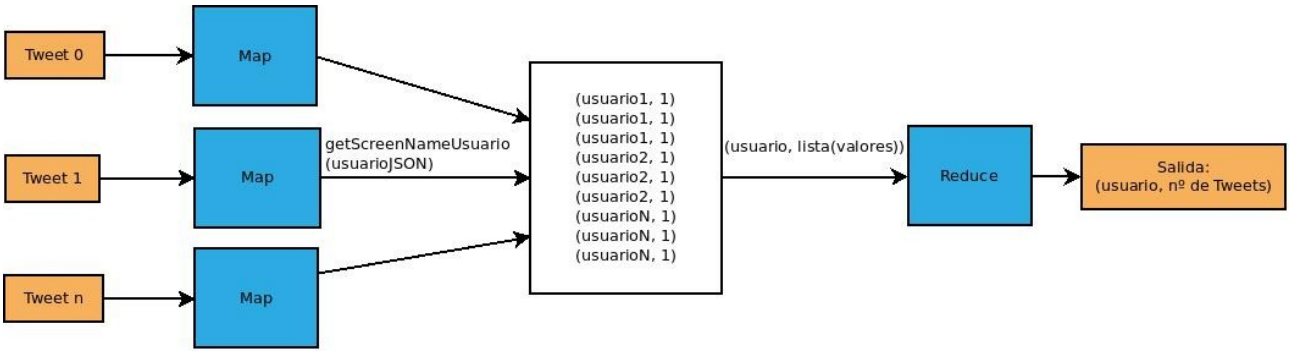

Figura 23: Flujo de datos MapReduce para obtener los tweets por usuario

En resumen, la función *Map* obtendría una lista de pares (usuario, valor), siendo valor el objeto *IntWritable*. Acto seguido, se agruparían todos los pares con el mismo usuario en pares (*usuario*, lista(*valores*)). Por último, el *Reduce* recorrería la lista de valores para obtener el número de elementos que esta lista contiene. Este número coincide con el número de tweets de cada usuario, por tanto, el resultado final seguiría la convención (*usuario, nº de tweets*).

Teniendo este fichero de resultados, podemos filtrar por cada usuario para ver los tweets de un usuario de forma individual, utilizando el siguiente comando:

```
grep "usuario" "fichero_resultados"
```

A modo de ejemplo, se busca un usuario concreto de la lista: '*psoetriacastela*'

```
ccaballe@tucan:~$ grep "psoetriacastela" tweets_users_filtered
'psoetriacastela'     1350
```

Como vemos, la salida del comando nos indica que el usuario con alias '*psoetriacastela*' tiene un total de 1350 tweets en total en toda la traza extraída. Así se puede consultar en cualquier momento el número de tweets extraídos de cualquier usuario.

Ya que tenemos un listado con usuarios de interés, realizamos un filtrado por cada uno de esos usuarios mediante el siguiente script en bash.

```
#!/bin/bash

while read user
do
grep $user tweets_users >> tweets_users_filtered
done < users.txt
```

El script lee el usuario del fichero y una vez obtenido este, lo utiliza para el "*grep*" y lo añade a un nuevo fichero. Este fichero tendrá, finalmente, los usuarios de la lista junto a su número de tweets. Teniendo esto, podemos obtener un ranking de usuarios para ver cuales son los 10 usuarios que más han twitteado.

Posición	Usuario	Nº de Tweets Traza Filtrada	Nº de Tweets Traza Completa
1º	@mariviromero	15919	19002
2º	@alcaldehuevar	10309	10779
3º	@ismaelbosch	9025	9881
4º	@patriestevez	6901	7806
5º	@rosamariaartal	6566	7400
6º	@maruhuevar	6027	6186
7º	@cmgorriaran	5765	6219
8º	@garciaretegui	5512	5670
9º	@dgpastor	4627	4979
10º	@pedroj_ramirez	4580	4763

Tabla 10: Ranking de usuarios con más tweets

La tabla se ha ordenado según la traza filtrada, se puede ver como se respetan las posiciones en las trazas completa y filtrada, a excepción de los puestos 6º y 7º, que se intercambiarían si se ordenará

por la traza completa. En la siguiente figura, podemos ver de forma gráfica, la variación de valores entre la traza filtrada y completa. Vemos que las diferencias no son muy significativas entre las dos traza, a excepción del primer puesto, en el que la diferencia es de más de 3000 tweets.

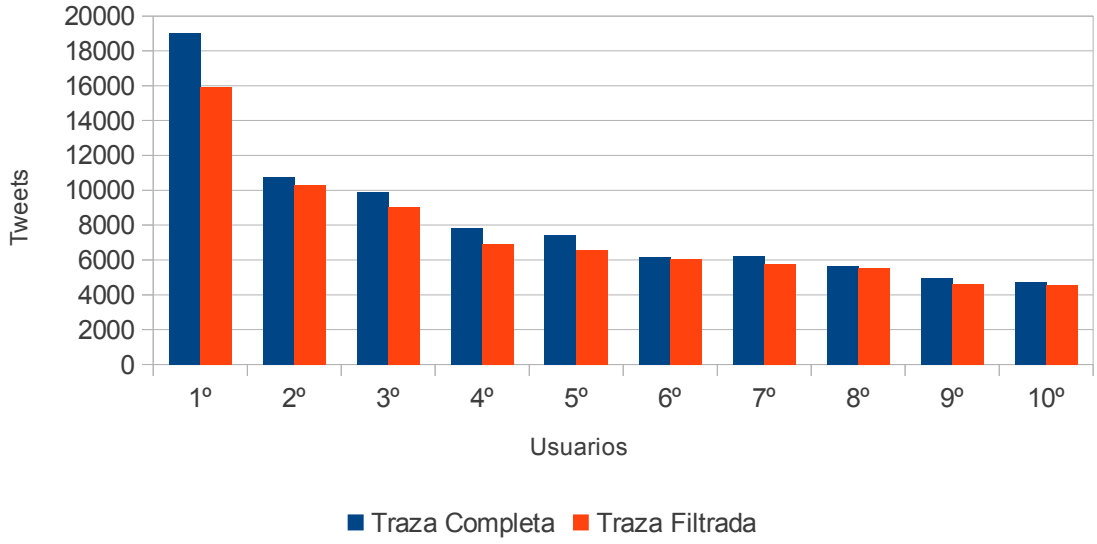

Figura 24: Relación entre las trazas completa y filtrada del ranking de usuarios con más tweets

En la siguiente figura, se representa el ranking de la traza filtrada.

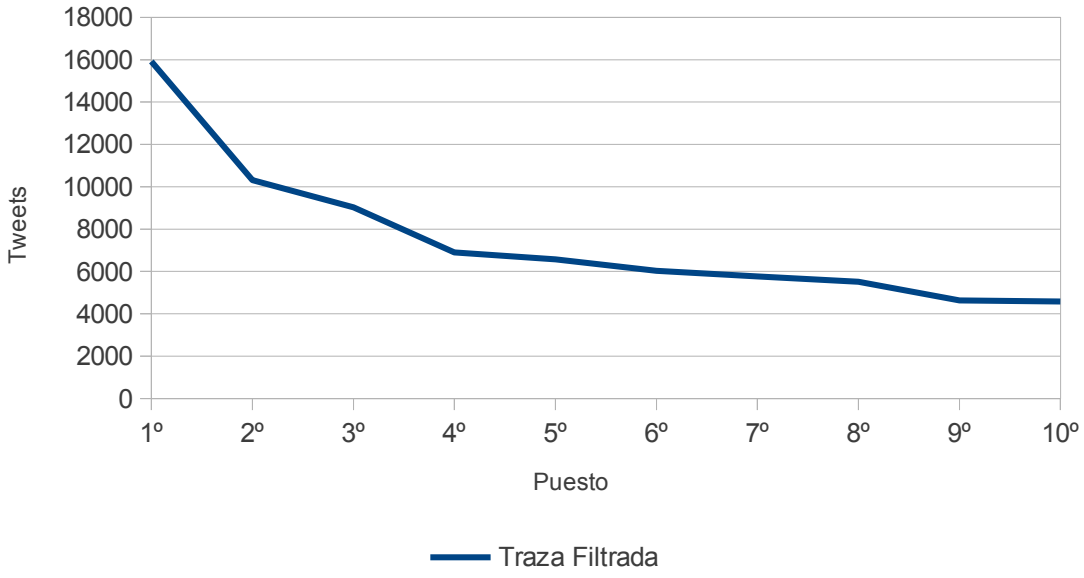

Figura 25: Top 10 de usuarios con más tweets para la traza filtrada

Como podemos ver en la gráfica, llama la atención el pico tan alto que existe en el primer puesto (*@mariviromero*), con casi 16000 tweets para la traza filtrada, seguido por el segundo puesto con algo más de 10000. Se puede ver que el usuario *@mariviromero* destaca con respecto al resto, ya que la diferencia entre el primer y segundo puesto es prácticamente igual a la diferencia entre el segundo y décimo puesto, lo que nos indica una fuerte dispersión en ese punto. Además, entre el primer puesto y el décimo del ranking hay una diferencia de más del triple.

Una primera aproximación al analizar la gráfica nos lleva a la conclusión de que los valores se van estabilizando cuando avanzamos de puestos, siendo la variación entre cada posición consecutiva algo menor cada vez. Para una mayor certeza de la anterior hipótesis, a continuación se muestra una gráfica con un ranking de 100 usuarios. Dada la complejidad de formar este tipo de gráficas, se han implementado sencillos programas Java utilizando una librería de creación de gráficos llamada *JfreeChart*. [73]

Figura 26: Top 100 de usuarios con más tweets para la traza filtrada

Viendo la gráfica, podemos confirmar la hipótesis anterior. Sabiendo que en el puesto 10 el número de tweets es de 4580, si nos fijamos en la gráfica, el puesto número 100 tiene algo más de 1000 tweets. Entre los puestos 10 y 100 hay una variación de +- 3000 tweets y 90 puestos mientras que entre los dos primeros puestos hay una variación de más de 5000 tweets, siendo estos puestos consecutivos. Además, gráficamente se puede ver como la pendiente de la gráfica va disminuyendo a medida que la gráfica tiende a infinito en el eje de posiciones.

3.4.3 MapReduce para obtener los tweets por fecha de cada usuario

En los anteriores apartados, hemos analizado por un lado el número de tweets extraídos por día y por otro el número de tweets que se corresponde con cada usuario. Siguiendo la misma metodología, en este apartado vamos a ver el número de tweets de cada usuario en cada día concreto. Una vez obtenidos los resultados, podemos analizar cada usuario en concreto para obtener un seguimiento individual de los tweets que ha escrito en un rango de fechas determinado.

Una vez más, el Reduce de este programa solamente tiene la función de obtener el número de elementos de una lista. Por tanto, no es necesario explicar de nuevo la implementación.

La clase *Mapper* tiene un componente especial con respecto a los dos apartados anteriores. En este caso, se ha de mapear la traza sobre dos claves: usuario y fecha. Para seguir con el paradigma de programación de *MapReduce*, la estrategia a seguir ha sido mapear la clave como un conjunto de dos claves. Es decir, cada clave será, a su vez, un par de claves (usuario, fecha). El siguiente fragmento de pseudocódigo muestra las partes más importantes de la clase:

```
FUNCION Map(clave, valor)
      tweet ← leerLinea()
      usuarioJSON ← obtenerUsuarioJSON(tweet)
      screenName ← getcreenNameUsuario(usuarioJSON)
      SI listaUsuarios.contiene(screenName)
            fecha ← obtenerFecha(tweet)
            nuevaClave ← screenName + ":" + fecha
            emitirContexto(nuevaClave, 1)
FIN FUNCION
```

La función *Map* realiza los siguientes pasos:

1 Obtiene el *screenName* del usuario del tweet

2 Comprueba si el usuario se corresponde con un usuario de la lista. En caso afirmativo:

2.1 Se obtiene la fecha del tweet.

2.2 Se forma la clave con el texto “usuario”:”fecha”

2.3 Se propaga el par (*clave, valor*).

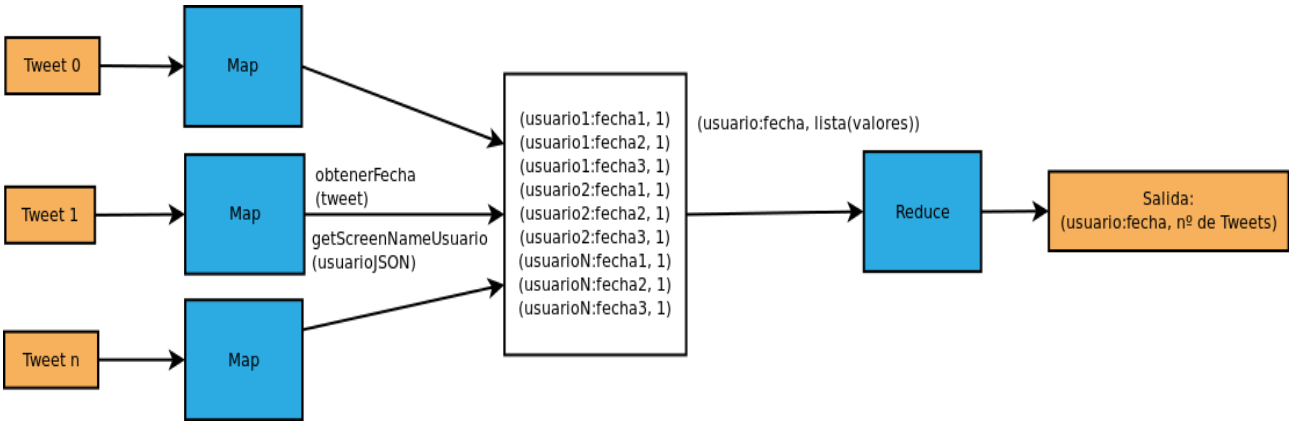

Figura 27: Flujo de datos MapReduce para obtener los tweets por fecha de cada usuario

Como se indica en el paso 2, en la función *Map* se comprueba si el tweet es de los usuarios de interés de estudio. Esto se ha diseñado con la idea de analizar únicamente el conjunto de usuarios definidos para no aumentar el tamaño del problema. Sin embargo, es sencillo cambiar los usuarios a filtrar simplemente modificando el ficheros de usuarios o introduciendo un pequeño cambio en el sistema de computo.

Como resultado, obtenemos un fichero formado en el que cada linea muestra el usuario seguido del día y el número de tweets. A modo de ejemplo, se muestra un fragmento del fichero para el usuario *@20m* y otro para el usuario *@mariviromero*.

```
'20m':2011/11/18 67
'20m':2011/11/19 37
'20m':2011/11/20 51
'20m':2011/11/21 104
'20m':2011/11/22 57
'20m':2011/11/23 48
'20m':2011/11/24 39
'20m':2011/11/25 44
```

```
'mariviromero':2011/12/01   728
'mariviromero':2011/12/02   597
'mariviromero':2011/12/03   640
'mariviromero':2011/12/04   659
'mariviromero':2011/12/05   490
'mariviromero':2011/12/06   278
'mariviromero':2011/12/07   134
'mariviromero':2011/12/08   441
```

Para obtener este tipo de resultados, podemos filtrar por usuario de forma sencilla con:

```
grep 'usuario' 'fichero'
```

De esta forma, podemos ver a modo de ejemplo los tweets de *@mariviromero* ya que es el usuario que más tweets ha escrito de todos los estudiados.

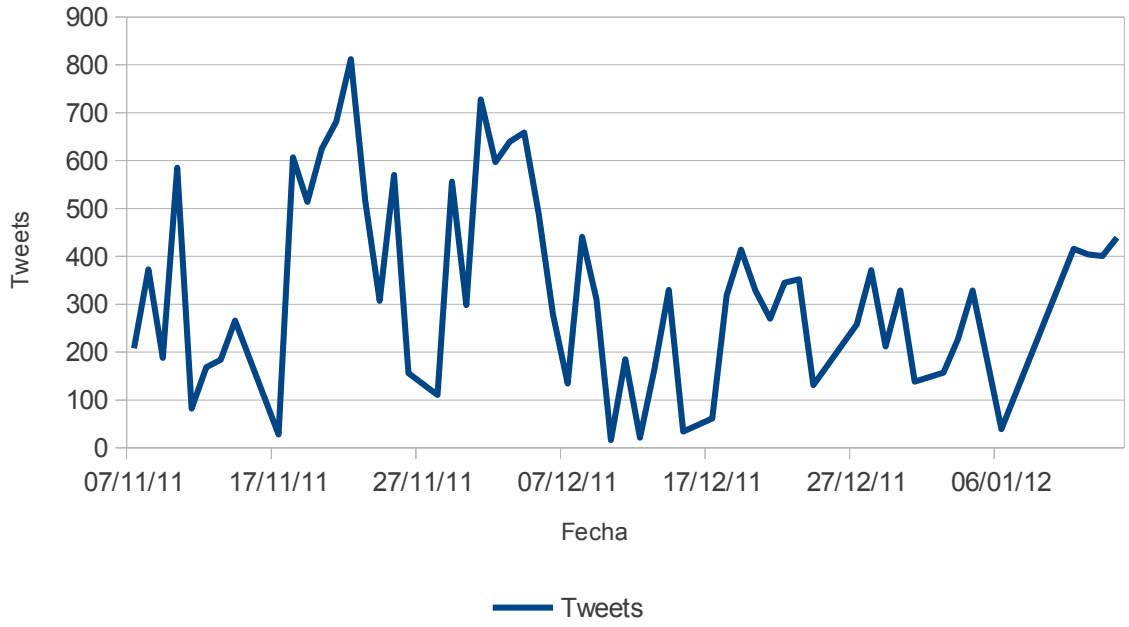

Figura 28: Seguimiento de tweets para el usuario @mariviromero

Vemos que hay un día que llega incluso a twittear más de 800 veces. Concretamente, el 22 de Noviembre de 2012 con 812 tweets. También podemos ver que únicamente baja de 100 tweets 7 días de todos los analizados, y, si calculamos el promedio de la distribución, obtenemos un resultado de ~327 tweets. Con una media de algo más de 300 tweets diarios, los resultados que se obtienen son los siguientes:

$$\frac{300\text{Tweets}}{10\text{horas}} = \frac{30\text{Tweets}}{1\text{hora}} = \frac{1\text{Tweet}}{4\text{minutos}}$$

Es decir, con esos datos podemos decir que se twittea cada 4 minutos. Suponiendo el día de 812 tweets, con datos similares en cuanto a tiempo de uso diario de Twitter se puede ver fácilmente como este usuario realizaría un tweet con frecuencia de algo menos de dos minutos. Con todo esto, podemos afirmar que este usuario utiliza *Twitter* con bastante frecuencia.

De igual forma que ocurría en el apartado 3.3.1, el análisis de estas distribuciones nos lleva a buscar que días sobresalen de los intervalos superior e inferior, calculando la media y la desviación

típica. En la siguiente gráfica se muestran estos intervalos:

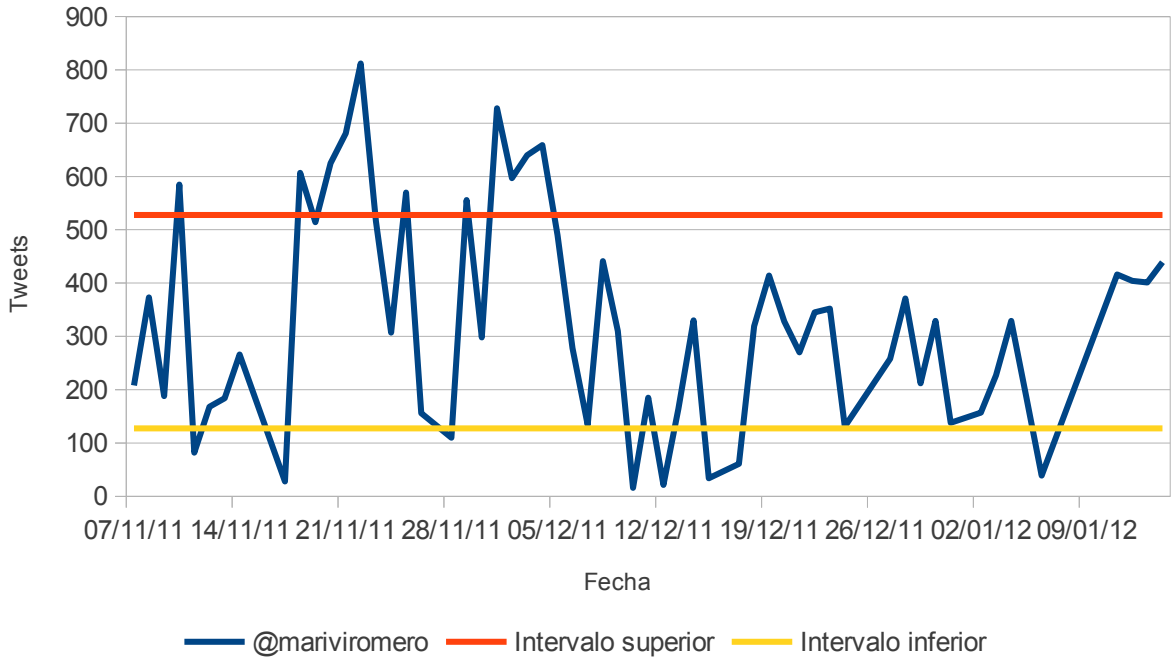

Figura 29: Intervalos superior e inferior para los tweets de @mariviromero

Como podemos ver, existen bastantes valores por encima y por debajo de los intervalos, en concreto, los valores que se salen de los intervalos son los que corresponden a los siguientes días:

- **Días con más tweets**: 10/11/11, 18/11/11, 20/11/11, 21/11/11, 22/11/11, 25/11/11, 01/12/11, 02/12/11, 03/12/11, 04/12/11, 05/12/11.
- **Días con menos tweets**: 09/11/11, 11/11/11, 12/11/11, 13/11/11, 17/11/11, 26/11/11, 28/11/11, 07/12/11, 10/12/11, 11/12/11, 12/12/11, 13/12/11, 15/12/11, 17/12/11, 24/12/11, 31/12/11, 02/01/12, 05/01/12, 06/01/12.

Podemos ver que la distribución es muy dispersa ya que son muchos los valores que no están dentro de los límites, en concreto, 30 valores de un total de 58 días. De estos 30, 11 valores se exceden del límite superior de 458 tweets. Se puede observar como los días en los que se excede el intervalo superior están más cerca del 20/11/11, al contrario que la mayoría de los días con menos tweets, que están más alejados Por tanto, se podría suponer que el usuario aumentó su actividad de uso de Twitter para las temporadas de pre elecciones y post elecciones.

Mediante estas representaciones gráficas, podemos comparar dos o más usuarios para ver sus diferencias. A modo de ejemplo, se van a estudiar los dos usuarios con más tweets: *@mariviromero* y *@alcaldehuevar*. La siguiente tabla muestra un resumen de los resultados estadísticos de los dos usuarios.

Usuario	Varianza	Media	Desviación típica	Intervalo superior	Intervalo inferior
@mariviromero	40,059.77	327.34	200.14	527.49	127.19
@alcaldehuevar	17,108.12	185.81	185.81	316.6	55.01

Tabla 11: Resultados estadísticos para las trazas de @mariviromero y @alcaldehuevar

Comparando los resultados, vemos que *@mariviromero* tiene los resultados bastante más altos que *@alcaldehuevar* para ser puestos consecutivos en el ranking. Se puede ver como @mariviromero tiene de media 327.34 y *@alcaldehuevar* 185.91. Aunque la diferencia entre los dos valores es alta, en el contexto de tweets diarios son valores extremadamente altos.

Ya hemos estudiado cuales son los días superiores e inferiores para *@mariviromero*. Para *@alcaldehuevar*, obtenemos los siguientes resultados:

- Días con más tweets: 18/11/11, 21/11/11, 29/11/11, 30/11/11, 01/12/11, 02/12/11, 19/12/11, 30/12/11, 31/12/11.
- Días con menos tweets: 07/11/11, 12/11/11, 17/11/11, 07/12/11, 10/12/11, 12/12/11, 17/12/11, 06/01/12.

La siguiente gráfica compara a los usuarios *@mariviromero* y *@alcaldehuevar*.

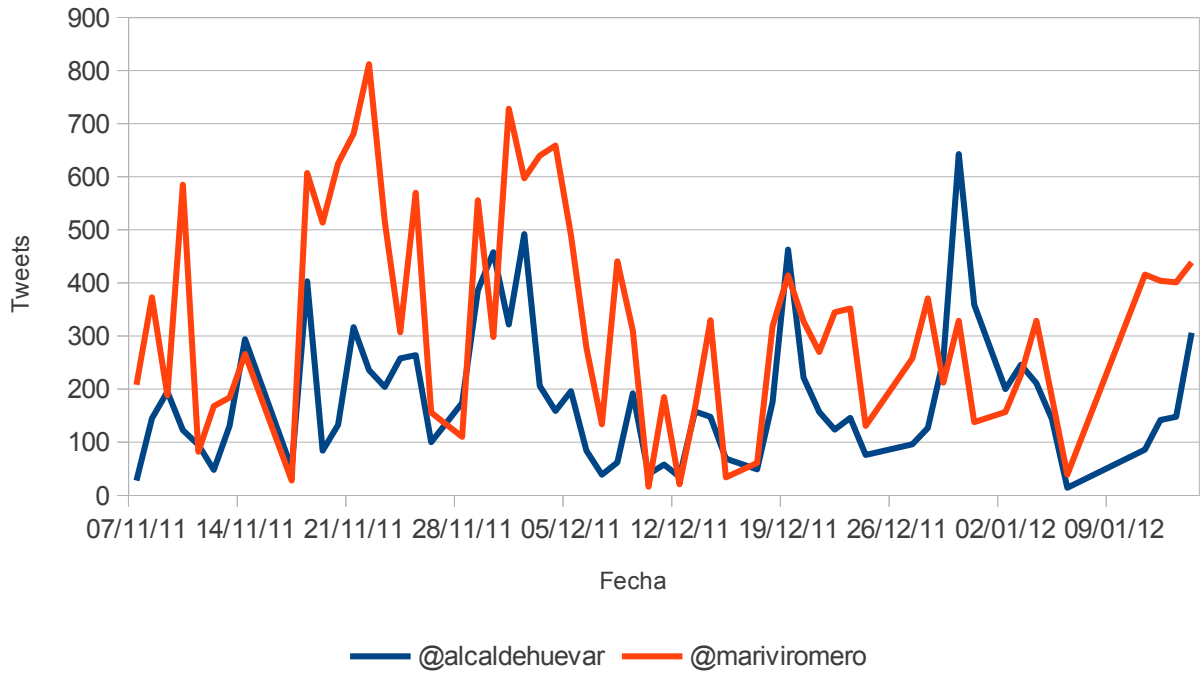

Figura 30: Comparativa de tweets entre @mariviromero y @alcaldehuevar

Generando gráficas como la anterior podemos obtener gran cantidad de información útil. Para este ejemplo, podemos ver que *@mariviromero* supera a *@alcaldehuevar* en casi todas las fechas, destacando los días iniciales y los cercanos al día de las elecciones. Sin embargo, *@alcadehuevar* supera en casi el doble a *@mariviromero* para el día 31 de Diciembre de 2011.

Para concluir en el análisis de la comparación entre estos dos usuarios, destacar que el día que más tweets envió *@mariviromero* fue el 22/11/11 con un total de 812 cambios de estado. Al contrario, el día de más tweets para *@alcaldehuevar* fue el 30/12/11 con un total de 643.

Para una mayor profundidad en el análisis de tendencias, la siguiente gráfica muestra una comparativa entre los tweets escritos por usuarios pertenecientes a partidos políticos y usuarios periodistas.

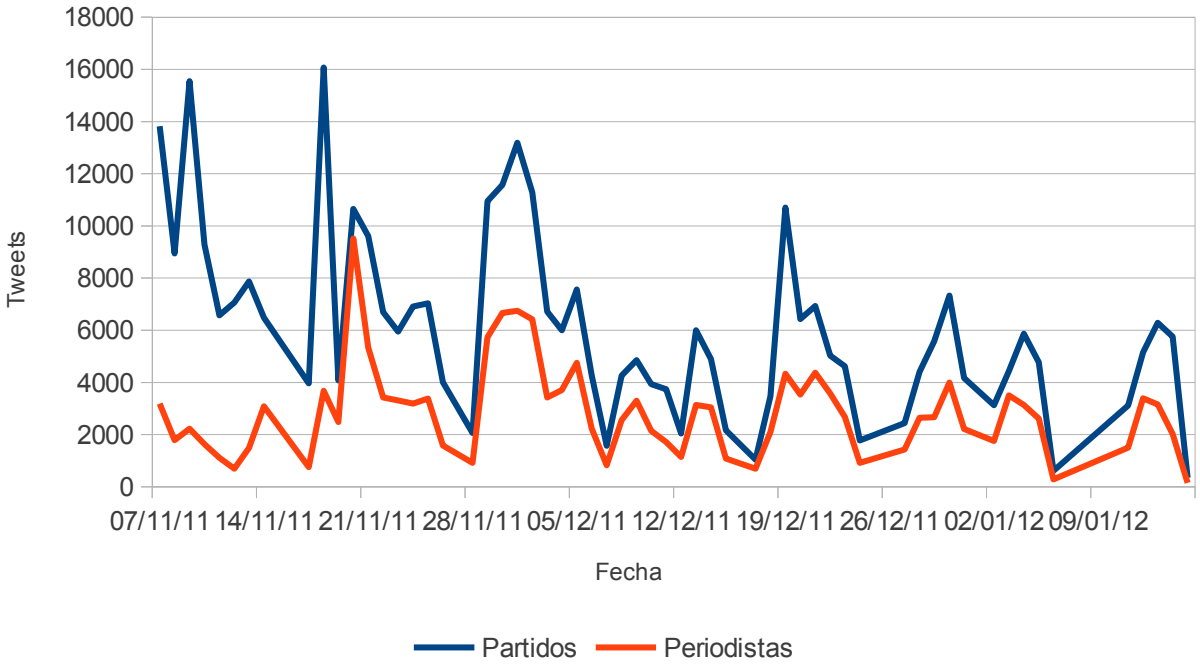

Figura 31: Comparativa de tweets para partidos políticos y periodistas

Como vemos, los tweets de usuarios pertenecientes a partidos políticos superan a los de los usuarios periodistas. Podemos ver como, en muchas ocasiones, los máximos y mínimos coinciden en varias distribuciones, haciendo que los intervalos de crecimiento y decrecimiento solo se diferencien por un incremento proporcional en el número de tweets.

También llama la atención que, aunque hay días que los partidos políticos duplican o triplican los tweets de los periodistas, otros días la diferencia entre estos es mínima. El ejemplo de esto es precisamente el 20 de Noviembre, ya que mientras es el valor máximo de los usuarios políticos, no ocurre lo mismo para los periodistas.

4. EVALUACIÓN DEL PROYECTO

En este apartado se describen algunos de los procesos realizados para evaluar el correcto funcionamiento del proyecto. Para ello, cada aplicación realizada ha sido sometida a un banco de pruebas necesarias para validar si el diseño inicial de cada una de las tareas es el adecuado para cada una de las tareas realizadas.

En general, para cada uno de las aplicaciones realizadas, se han realizado las siguientes pruebas:

- **Pruebas unitarias**: Dado que todas las aplicaciones están implementadas utilizando programación orientada a objetos (POO), se han realizado pruebas unitarias de cada método público realizado (siempre que, en el contexto del proyecto, tengan sentido estas pruebas). Mediante estas pruebas se valida el correcto funcionamiento de los métodos para cada una de sus posibles entradas, verificando que la salida obtenida en cada caso se corresponde con la esperada a priori. Estas pruebas se realizan utilizando un framework de pruebas para el lenguaje Java: *Junit* [74].
- **Pruebas funcionales/de aceptación**: Una vez realizadas las pruebas unitarias necesarias, se han realizado pruebas sobre la funcionalidad de las aplicaciones. El objetivo de estas pruebas es verificar el funcionamiento completo de las aplicaciones. Como norma general, se han realizado pruebas utilizando muestras o pequeños fragmentos de las entradas originales que queremos realizar. De esta forma, la ejecución completa de la traza simplemente será escalar sobre las pruebas de concepto iniciales.
- **Mediciones de rendimiento**: Una vez que hemos verificado que nuestra aplicación funciona correctamente, realizaremos una serie de ejecuciones sobre distintas condiciones para analizar la eficiencia de la aplicación. Las pruebas consistirán en obtener tiempos de ejecución con distintos recursos para evaluar, entre otros aspectos, la escalabilidad de la aplicación.

En los siguientes subapartados, se detallarán los resultados obtenidos tras la realización de la pruebas más relevantes para la evaluación del extractor de *Twitter*, las aplicaciones de filtrado y las aplicaciones de análisis de datos.

4.1 Evaluación del extractor de Twitter

En primer lugar, se evalúa el extractor de *Twitter* para verificar que se adapta a las necesidades

del proyecto. Esta evaluación consta de dos fases diferenciadas: pruebas para comprobar que el funcionamiento del extractor es el esperado y una validación de los requisitos del extractor.

4.1.1 Pruebas de comprobación del correcto funcionamiento del extractor

Una vez desarrollado el extractor de Twitter, se han realizado una serie de pruebas en la ejecución del extractor. Se pretende comprobar si los tweets recuperados se corresponden con las querys enviadas al API de Twitter. En concreto, los tweets deben, necesariamente, contener un término de búsqueda y/o pertenecer a un usuario de los seguidos por el extractor.

Estos tweets se guardan en ficheros de texto plano con un formato y nombre indicados. Por tanto, también se debe comprobar que el fichero tiene el nombre correspondiente y que cada fichero contiene los datos esperados. Esto se resumen en que, en los ficheros de términos deben estar los tweets recuperados por el Streaming de topics concretos, mientras que en el fichero de seguimiento de usuarios (*twitter_streaming_UsersFollows*), deben estar los tweets referentes al seguimiento de usuarios, es decir, deben ser tweets que pertenezcan a alguno de los usuarios listados.

Otro punto a probar es la tolerancia a fallos. Se debe comprobar que, aunque haya una caída de red, el extractor volverá a extraer automáticamente cuando se restablezca el servicio de red sin que sea necesario volver a relanzarlo manualmente.

Por último, se comprobará que es posible levantar varias instancias del extractor sin que afecte al funcionamiento normal de cualquiera de ellas. Una vez probado que el extractor funciona correctamente, se tiene que comprobar que se puede lanzar el mismo programa con distintos parámetros de entrada, siempre y cuando cada instancia tenga configurado un usuario distinto.

En resumen, algunas de las pruebas realizadas son las siguientes:

ID	Descripción de la prueba
P1	Lanzar una instancia del extractor con un término y un usuario a seguir. Después de la extracción, se verifica que existen dos ficheros: *twitter_streaming_[fecha]_[hora]_[término]* y *twitter_streaming_[fecha]_[hora]_[UserFollows]*. El primero debe contener en el texto del Tweet el término correspondiente, mientras que el segundo debe, necesariamente, tener Tweets que pertenezcan al usuario elegido.
P2	Lanzar una instancia del extractor con T términos y U usuarios. Después de la extracción, se verifica que existen T ficheros *twitter_streaming_[fecha]_[hora]_[término]* y un fichero *twitter_streaming_[fecha]_[hora]_[UserFollows]*. En el texto de los Tweets almacenados en los T ficheros de términos debe estar alguno de los términos indicados, mientras que en el fichero de usuarios deben ser todos los Tweets de alguno de los U usuarios indicados.
P3	Lanzar varias instancias del extractor, cada una con parámetros de entrada distintos y con un usuario distinto configurado en el fichero twitter4j.properties. Verificar que cada instancia por separado cumple lo impuesto en la prueba anterior.
P4	Lanzar una instancia del extractor con T términos y U usuarios y, en pleno proceso de extracción, deshabilitar la conexión a internet del ordenador durante un tiempo de 10 minutos. Acto seguido, rehabilitar la conexión y comprobar que el extractor sigue su proceso normal de ejecución.

Tabla 12: Pruebas realizadas para el correcto funcionamiento del extractor

4.1.2 Validación de los requisitos del extractor

En el capítulo 3.1.1 definíamos algunos requisitos que el extractor tendría que cumplir obligatoriamente para poder satisfacer las necesidades del proyecto. Una vez desarrollado el extractor, es necesario comprobar si se cumplen todos los requisitos impuestos. En caso contrario, se plantearía un rediseño de la aplicación para volver a implementarlo hasta que se valide que se superan todos los requisitos.

La forma de validar estos requisitos es mediante las pruebas definidas en el apartado anterior. En la siguiente tabla, se muestra la matriz de trazabilidad, en la que relacionamos los requisitos definidos en la tabla 5 con las pruebas de aceptación definidos en el apartado anterior. Como vemos, existe un mínimo de una prueba para cada requisito, por lo que concluimos en que el extractor queda validado y es apto para las tareas de extracción propuestas en el proyecto.

ID Requisito	Descripción del Requisito	Pruebas relacionadas
R1	El extractor debe ser capaz de obtener los tweets filtrados por términos utilizando el API de Streaming.	1, 2,3
R2	El extractor debe ser capaz de obtener los tweets por usuario utilizando el API de Streaming.	1,2,3
R3	El número de usuarios y términos que se pueden seguir al mismo tiempo estará limitado únicamente por el API de Twitter.	2
R4	El extractor debe ser capaz de recuperarse de un error de red.	4
R5	El extractor se apoyará en la librería Twitter4J.	3
R6	Debe ser posible lanzar varias instancias del extractor en máquinas distintas con distintos usuarios mediante autenticación OAuth.	3
R7	El extractor almacenará los tweets en ficheros de texto.	1,2,3

Tabla 13: Validación de los requisitos del extractor de Twitter

4.2 Evaluación del filtrado de la traza

En este apartado se realizará la evaluación de las aplicaciones *MapReduce* implementadas para filtrar la traza por idioma y por usuario y menciones.

4.2.1 Pruebas de comprobación del correcto funcionamiento del filtrado de la traza

Antes de su ejecución en el cluster, se realizaron pruebas en local (standalone) para verificar el correcto funcionamiento de los programas de filtrado de traza desarrollados. La entrada que se utilizó en las pruebas era un fragmento de la traza completa que se ejecutaría en el cluster, por lo que de esta forma se podía comprobar que el formato de entrada es correcto.

Se realizó una prueba para cada una de las dos aplicaciones de filtrado de la traza:

- **Prueba de filtrado por idioma**: Se valida que, dada una entrada con tweets en varios idiomas, se deben obtener tweets en los que el texto sea español. Dada la naturaleza de la prueba, se permitió un margen de error de un 1% en cuanto a los tweets recuperados.
- **Prueba de filtrado por usuario y menciones**: Se toma como entrada la salida de la prueba anterior y se valida que todos los tweets correspondientes cumplen los filtros. Es decir, todos los tweets filtrados deben ser de un usuario de la lista o mencionar a alguno

4.2.2 Rendimiento de las aplicaciones MapReduce de filtrado

Para medir el rendimiento, se han ejecutado las aplicaciones de *MapReduce* de filtrado de idioma y de filtrado de menciones/usuarios de la traza utilizando distintos número de nodos del cluster un gran número de veces. Esto dará una idea aproximada del tiempo de ejecución bajo distintos nodos, aunque hay que tener en cuenta los tamaños de entrada y salida de los programas, además de la gran cantidad de parámetros de *Hadoop*. Los resultados obtenidos son los siguientes:

Filtrado de idioma

- **Entrada del programa:** Traza completa con ~89 GB de tamaño.
- **Salida del programa:** Traza filtrada por idioma con ~30GB de tamaño.

Nodos	Tiempo (hh:mm:ss)
2	04:37:22
4	02:48:35,50
6	02:06:59,50
8	02:03:17
10	01:29:05
12	01:31:07,50
14	01:33:51,50
16	01:37:49

Tabla 14: Tiempos de ejecución de MapReduce filtrado de idioma

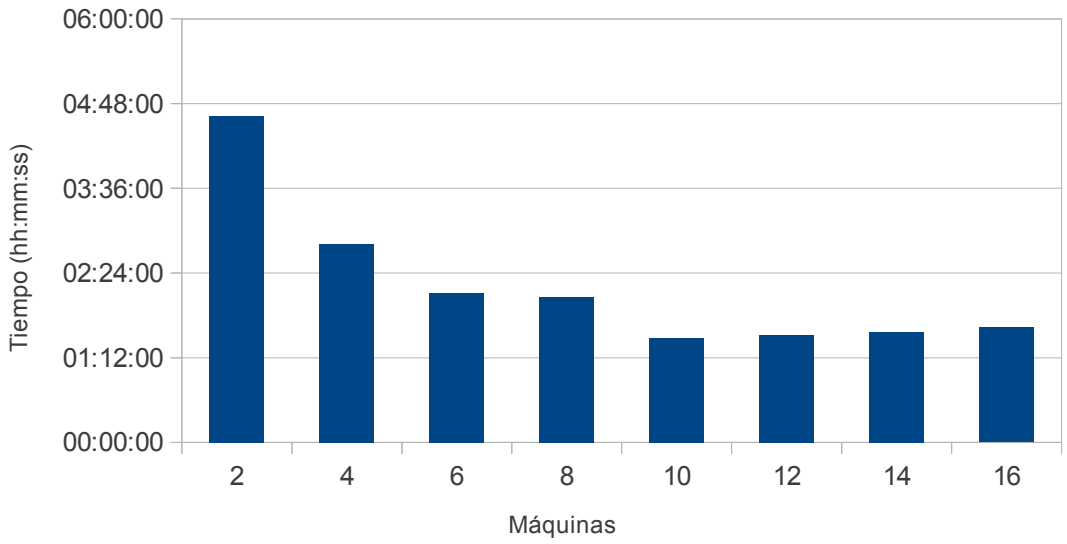

Figura 32: Tiempos de ejecución de MapReduce filtrado de idioma

Analizando la gráfica, llama la atención el gran salto de tiempo que existe entre realizar la ejecución con 2 o con 4 nodos, concretamente, casi 2 horas menos. Lo que nos indica que, en ese intervalo, la escalabilidad es alta ya que el tiempo de reparto entre las tareas de *Map* y *Reduce* se compensa con el tiempo de ejecución de estas en cada uno de los nodos.

Si seguimos aumentando las máquinas, aunque la escalabilidad es menor, notamos cierta mejoría hasta 10 máquinas. Sin embargo, a partir de 10, notamos un ligero empeoramiento en los tiempos al aumentar máquinas. Esto se puede deber, entre otros aspectos, a que el tiempo de repartición de los split de entrada entre los nodos es, evidentemente, mayor cuanto más máquinas hay que repartir y, aunque el procesamiento “local” en cada nodo sea menor, no se compensa con el tiempo anterior.

Resumiendo, los pequeños empeoramientos de tiempo que se realizan a partir de los 10 nodos se pueden debe a las operaciones de entrada/salida que realiza el proceso *datanode* de Hadoop, sin embargo, pueden influir otro tipo de factores:

- Los nodos del cluster son heterogéneos, es decir, no tienen las mismas características, lo que hace que si una máquina tiene un rendimiento bastante menor que las demás, se genere un cuello de botella con respecto al tiempo de ejecución.
- Cambiar parámetros de configuración de *Hadoop*.

Una solución que se adopta en los entornos corporativos es lo que se conoce como la utilización de commodity Hardware [75]. Se conoce como commodity Hardware a la utilización de Hardware de gama media-baja y de bajo coste, normalmente para utilizarlo en clusters o granjas de computación con cientos o miles de nodos. De esta forma, se consigue tener un cluster de equipos de igual rendimiento a bajo coste para tareas distribuidas y manteniendo la filosofía de *MapReduce* y *Hadoop*.

Aun con esto, podemos comprobar que hemos filtrado por idioma una traza entera con más de 89GB y utilizando 10 nodos como mínimo en, más o menos, 1 hora y 30 minutos, por lo que podemos decir que los resultados son positivos.

Filtrado de usuarios y menciones

- **Entrada del programa:** Traza filtrada por idioma con ~30 GB de tamaño.
- **Salida del programa:** Traza filtrada por idioma, usuario y menciones con ~19 GB de tamaño.

Nodos	Tiempo (hh:mm:ss)
2	01:17:39
4	01:04:19
6	00:39:31
8	00:34:36
10	00:48:01
12	00:41:16
14	00:34:51
16	00:43:50

Tabla 15: Tiempos de ejecución de MapReduce filtrado de usuarios y menciones

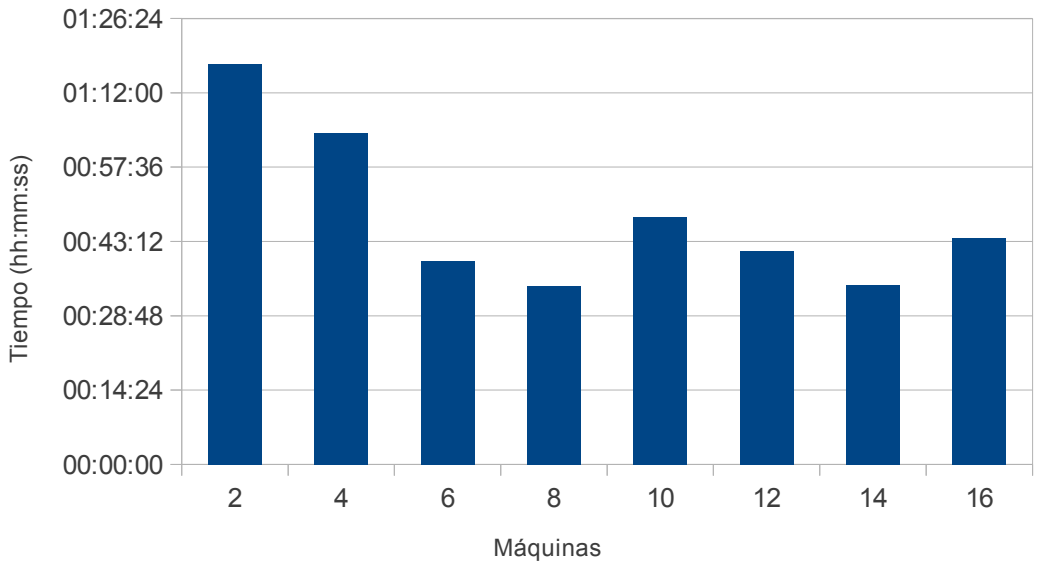

Figura 33: Tiempos de ejecución MapReduce filtrado de usuarios y menciones

Para este caso, vemos como va disminuyendo el tiempo empleado hasta llegar a los 10 nodos, en los que aumenta considerablemente. A partir de 10 nodos, disminuye cierto tiempo, aunque al aumentar a 16, vuelve a aumentar el tiempo. Todo parece indicar que, la dispersión que sufren los datos a partir de 8 nodos se pueda deber a los procesos de lectura y escritura, ya que, para este programa, cada *Map* debe leer de la lista de usuarios para saber que usuarios filtrar. Sin embargo, para el caso de 8 máquinas realizamos el filtrado completo de 30 GB en poco más de media hora, lo que es un tiempo excelente.

4.3 Evaluación del análisis de datos

Una vez filtrada la traza, esta sirve como entrada para aplicaciones *MapReduce* en las que obtenemos distintos datos de estudio, como pueden ser el número de tweets de un usuario concreto, los tweets por día o la "combinación" de los dos. En este apartado se evalúan estas aplicaciones utilizadas para realizar distintos tipos de análisis de la traza filtrada.

4.3.1 Pruebas de comprobación del correcto funcionamiento del análisis de la traza filtrada

Una vez más, antes de llevar todo el procesamiento al cluster, se realizaron pruebas en local (standalone) para verificar el correcto funcionamiento de los programas de análisis de datos. La entrada utilizada para estas evaluaciones se correspondía con un fragmento de la traza filtrada, que es la que nos interesa analizar en el cluster. Con esto, se pudo comprobar que el formato de esta traza filtrada es el esperado y, en caso contrario, se corregiría en el paso que correspondiera (ya sea en el programa de filtrado de la traza o en el de análisis de datos).

Se realizó una prueba para cada una de las tres aplicaciones de análisis de la traza filtrada:

- **Prueba de obtención de tweets por fecha**: Se valida que el resultado final se corresponde con un fichero compuesto por una linea por cada día y en cada linea se indica además el número de tweets de ese día determinado. A continuación se muestra un fragmento de salida válida:

```
2011/11/08 246
2011/11/09 204
2011/11/10 167
2011/11/11 120
2011/11/12 101
2011/11/13 129
2011/11/14 89
```

```
2011/11/17  44
2011/11/18  179
2011/11/19  86
```

- **Prueba de obtención de tweets por usuario:** Se valida que el resultado final se corresponde con un fichero compuesto por una linea por cada usuario y en cada linea se indica además el número de tweets de ese usuarios determinado. A continuación se muestra un fragmento de salida válida:

```
'1centella' 159
'20m' 1869
'360gradospress' 861
'abalosmeco'     1014
'abasagoiti'     93
'abc_es'   2065
'aberron'  1685
'acerobolche'    74
'aclotet'  78
'adelacastano'   169
```

- **Prueba de obtención de tweets por fecha y usuario:** Se valida que el resultado final se corresponde con un fichero compuesto por una linea por cada día y usuario concreto y en cada linea se indica además el número de tweets de ese usuario y día determinados. En total, el fichero tendrá *DxU* lineas, donde D es en número de días en los que se extrajeron datos y U el número de usuarios analizados. Un fragmento de salida valida tendría que ser similar al siguiente (para el caso del usuario *@20m*)

```
20m;2011 11 07   6
20m;2011 11 08   15
20m;2011 11 09   18
20m;2011 11 10   21
20m;2011 11 11   14
20m;2011 11 12   5
20m;2011 11 13   11
20m;2011 11 14   32
20m;2011 11 17   5
20m;2011 11 18   60
```

4.3.2 Rendimiento de las aplicaciones MapReduce de análisis de datos

De igual forma que se realizaba en el apartado 4.2.1, se han ejecutado las aplicaciones de análisis de datos con distintos números de nodos para obtener los tiempos de ejecución en cada caso. Con esto, se pretende realizar un estudio del rendimiento de las aplicaciones *MapReduce* de análisis de datos, para evaluar los tiempos de ejecución utilizando como variable el número de nodos del cluster que se utilizan cada vez. Los resultados obtenidos se muestran a continuación:

Tweets por fecha

- **Entrada del programa:** Traza filtrada por idioma, usuario y menciones con ~19 GB de tamaño.
- **Salida del programa:** Número de tweets por fecha.

Nodos	Tiempo (hh:mm:ss)
2	00:13:29
4	00:11:00
6	00:08:47
8	00:08:19
10	00:09:38
12	00:09:30
14	00:10:17
16	00:10:20

Tabla 16: Tiempos de ejecución MapReduce tweets por fecha

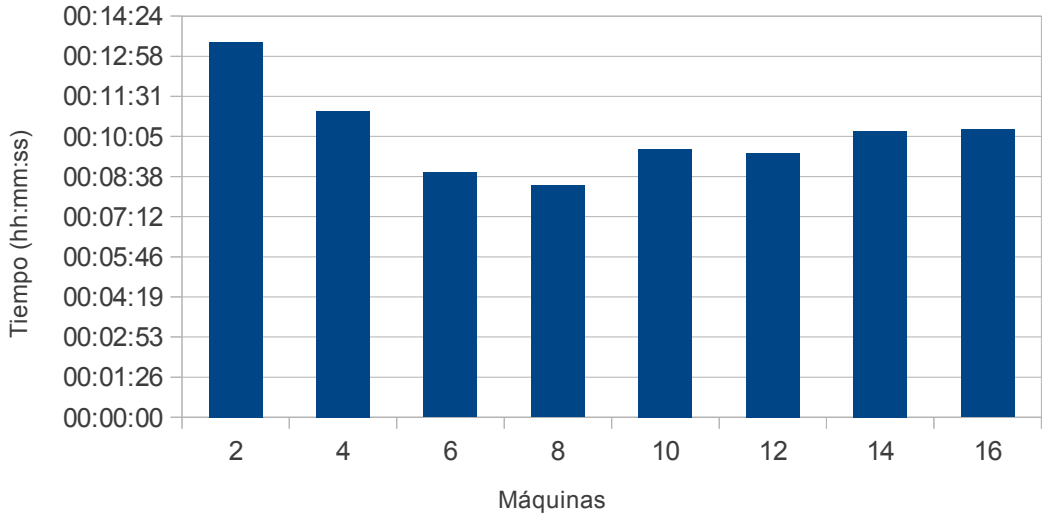

Figura 34: Tiempos de ejecución MapReduce tweets por fecha

Tweets por usuario

- **Entrada del programa:** Traza filtrada por idioma, usuario y menciones con ~19 GB de tamaño.
- **Salida del programa:** Número de tweets de cada usuario.

Nodos	Tiempo (hh:mm:ss)
2	00:14:03
4	00:11:50
6	00:08:47
8	00:08:34
10	00:09:26
12	00:09:48
14	00:09:51
16	00:10:54

Tabla 17: Tiempos de ejecución MapReduce tweets por usuario

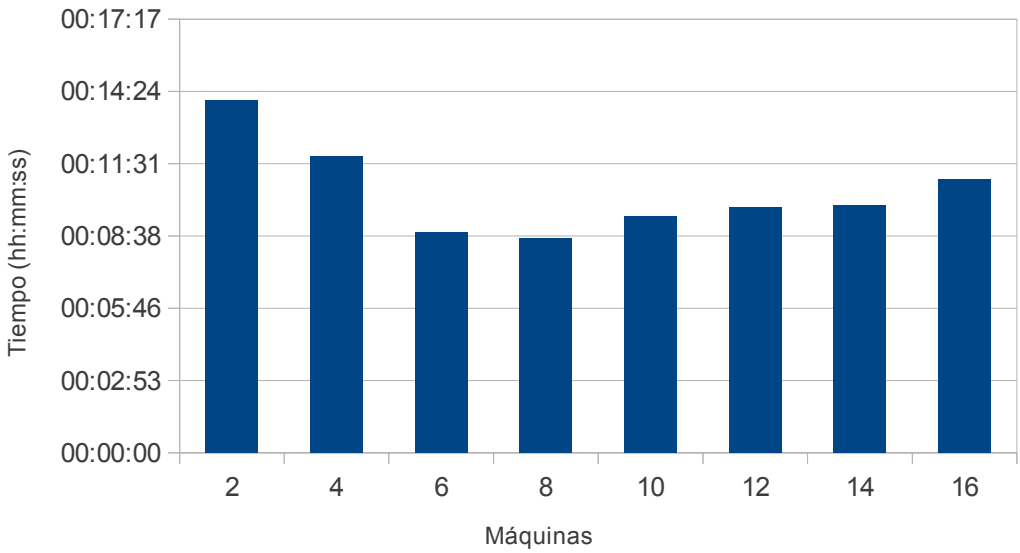

Figura 35: Tiempos de ejecución MapReduce tweets por usuario

Tweets por fecha y usuario

- **Entrada del programa:** Traza filtrada por idioma, usuario y menciones con ~19 GB de tamaño.
- **Salida del programa:** Número de tweets por usuario y fecha.

Nodos	Tiempo (hh:mm:ss)
2	00:14:40
4	00:11:30
6	00:08:52
8	00:08:36
10	00:09:21
12	00:08:38
14	00:10:40
16	00:11:10

Tabla 18: Tiempos de ejecución MapReduce tweets por fecha y usuario

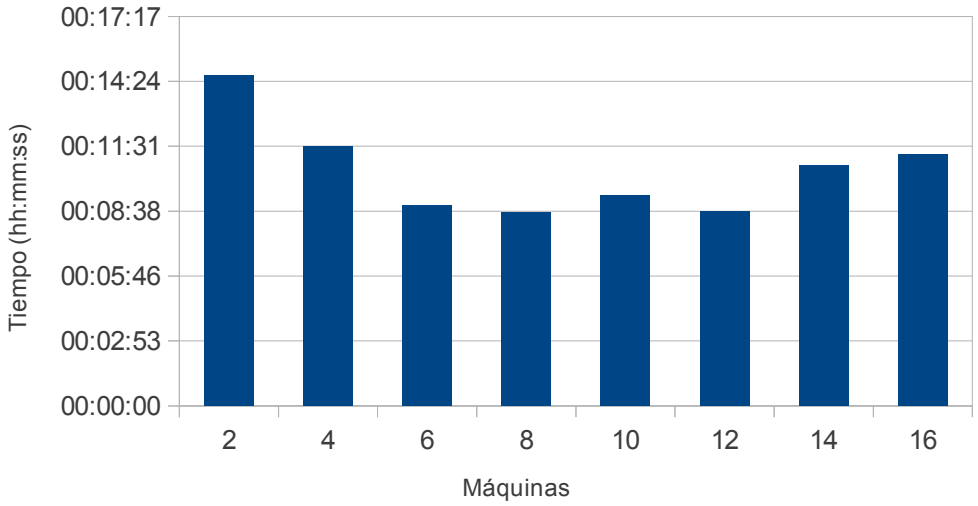

Figura 36: Tiempos de ejecución MapReduce tweets por fecha y usuario

Después de ver todas las gráficas, podemos sacar gran cantidad de conclusiones de ellas. Después de una pequeña observación de los resultados, obtenemos que:

- El menor tiempo se obtiene utilizando 8 nodos.
- Los tiempos de ejecución siempre disminuyen hasta llegar a 8 nodos, punto en el cual empieza a subir.
- Una vez que empieza a subir después de los 8 nodos, se realiza una ligera bajada para el caso de 12 nodos en las aplicaciones de tweets por fecha y tweets por fecha y usuario.
- Los tiempos obtenidos para cada una de las 3 aplicaciones son muy parecidos, a pesar de que la tercera (tweets por fecha y usuario) podría tomarse como una combinación de las otras dos.

Habiendo analizado lo anterior, podemos ver que, para el caso mejor (8 nodos), obtenemos unos tiempos de entre 8 minutos y 19 segundos y 8 minutos y 38 segundos. Además, dada la facilidad con la que están implementadas estas aplicaciones, se podrían realizar cualquier otro tipo de análisis con una pequeña modificación en el sistema de cómputo y, sin embargo, el tiempo no variaría considerablemente.

5. GESTIÓN DEL PROYECTO

Este apartado contiene la información referente a la gestión del proyecto. Se explica la metodología utilizada durante el desarrollo del TFG. En primer lugar, se ha realizado una descomposición en tareas del proyecto, de forma que se pueda organizar el proyecto en distintos hitos de trabajo. Seguidamente, mostramos un diagrama de *Gantt*, en el que se resume el periodo de duración de cada una de las tareas y subtareas realizadas. Por último, se muestra el presupuesto calculado, incluyendo los distintos costes.

5.1 Descomposición en tareas

Para una mejor realización y organización del proyecto, se ha decidido descomponerlo en hitos diferenciados. Cada uno de estos hitos marca una actividad, que a su vez puede estar dividida en varias tareas, ya sean de investigación, desarrollo o documentación del trabajo realizado y los resultados obtenidos.

En este apartado, mostramos los distintos hitos por los que queda dividido el proyecto.

5.1.1 Hito 1. Estudio inicial y estado del arte

La primera actividad a realizar consiste en un breve análisis del problema y una investigación sobre las tecnologías que guardan relación con el proyecto y el contexto actual.

- **Tarea 1-1. Análisis del problema**: Consiste en determinar cuales son los objetivos del proyecto y realizar un análisis de viabilidad, es decir, analizar si los objetivos planteados están al alcance y pueden ser implementados.
- **Tarea 1-2. Investigación de las tecnologías y estado del arte**: En esta tarea se ha realizado un análisis del estado actual de las tecnologías utilizadas, como son: la extracción de datos de Twitter mediante el API de Streaming, tecnologías de Cloud Computing y tecnologías de procesamiento de gran cantidad de datos, destacando las implementaciones de *MapReduce* de *Google* y su alternativa libre: *Hadoop*.
- **Tarea 1-3. Realización de pruebas de concepto**: Una vez investigadas las distintas tecnologías, se realizaron unas pruebas de concepto creando aplicaciones simples y similares a las desarrolladas en este TFG. En concreto, se realizaron dos aplicaciones: un extractor de términos de *Twitter* capaz de lanzar una instancia de varias palabras y un ejemplo de *WordCount* en *Hadoop* [76].

5.1.2 Hito 2. Desarrollo de un extractor de datos para Twitter

Una vez estudiadas las distintas tecnologías y teniendo en cuenta los objetivos establecidos, el siguiente paso es el desarrollo de un extractor de datos masivo de *Twitter*.

- **Tarea 2-1. Establecer requisitos del extractor**: Antes de realizar el diseño y la implementación del extractor, es necesario definir cuales serán los objetivos que queremos que cumpla nuestra aplicación. Para ello, se definirán una serie de requisitos que validaran el buen funcionamiento del extractor.
- **Tarea 2-2. Arquitectura y diseño del extractor**: Una vez establecidos los requisitos que debe pasar el extractor, se realizará un diseño de alto nivel que muestre de forma clara la interacción del extractor con el API de Twitter y las clases y métodos Java que se implementarán.
- **Tarea 2-3. Implementación del extractor**: En esta tarea se implementarán en código Java el conjunto de clases y operaciones definidas anteriormente.
- **Tarea 2-4. Ejecución,despliegue y monitorización del extractor**: Una vez terminado el desarrollo del extractor, se ejecutarán varias instancias en un cluster con el fin de paralelizar la recogida de datos. De esta forma, se intentará obtener la traza de mayor tamaño posible. En esta tarea se incluye la monitorización periódica del extractor, utilizando técnicas de backup para casos de desastre y relanzando los trabajos si fuera necesario.

5.1.3 Hito 3. Limpieza y filtrado de la traza

Habiendo obtenido un conjunto grande de información, el paso previo al análisis es filtrar estos datos según nuestros intereses.

- **Tarea 3-1. Análisis previo de la traza**: Antes de filtrar los tweets, realizamos un análisis de la traza completa. En esta tarea analizaremos aspectos como el tamaño de la traza, el número de tweets recogidos, formato de la traza y posibles errores durante la recolección de datos.
- **Tarea 3-2. Limpieza de la traza**: Una vez analizada la traza, se hará una primera limpieza utilizando herramientas propias de Linux. En esta tarea se pretende, entre otras cosas, organizar la traza, eliminar las lineas corruptas y compactar los ficheros.
- **Tarea 3-3. Filtrado de la traza**: Teniendo una traza bien formada, el siguiente paso consiste en el desarrollo de una aplicación Hadoop para filtrar la traza por idioma y por usuarios. La aplicación deberá filtrar los tweets escritos en español. Acto seguido, se filtrarán los tweets pertenecientes a un grupo determinado de usuarios, así como los que mencionen a estos.

5.1.4 Hito 4. Análisis de los datos mediante Hadoop

Uno de los objetivos principales del proyecto es obtener datos estadísticos de un conjunto grande de datos. En este hito, se realizará ese análisis mediante aplicaciones desarrolladas en

Hadoop.

- **Tarea 4-1. Configuración de Hadoop**: Para ejecutar las aplicaciones *MapReduce* desarrolladas con *Hadoop*, es necesario configurar el cluster a utilizar para que soporte *Hadoop*. Además, es necesario configurar y levantar el sistema de ficheros de *Hadoop* (*HDFS*), así como otro tipo de procesos que permitan la correcta ejecución de *Hadoop* de forma distribuida.
- **Tarea 4-2. Implementación de aplicaciones Hadoop para el análisis de datos**: Se crearán distintas aplicaciones *MapReduce* mediante *Hadoop* para obtener datos estadísticos sobre la traza. De esta forma, obtendremos datos de interés sociológico, como por ejemplo, el número de tweets por usuario, número de tweets por día, etc.
- **Tarea 4-3. Elaboración de gráficas con los resultados obtenidos**: Partiendo de la tarea anterior, se elaborarán un conjunto de gráficos que nos permita ver de forma sencilla los resultados de nuestro análisis.

5.1.5 Hito 5. Evaluación del trabajo realizado

Para verificar que el código desarrollado funciona y cumple con las expectativas, debemos realizar un conjunto de pruebas, tanto unitarias como funcionales.

- **Tarea 5-1. Evaluación del funcionamiento del extractor**: Para evaluar el extractor, se realizarán una serie de pruebas teniendo en cuenta los requisitos obtenidos de la tarea 2-1. En estas pruebas, se estudiarán, entre otras cosas, la tolerancia a fallos del extractor y la capacidad de extracción de datos.
- **Tarea 5-2. Evaluación de las aplicaciones MapReduce**: Las aplicaciones desarrolladas con Hadoop se probarán en local utilizando pequeñas muestras tomadas de la traza completa. Además, también se realizarán estas pruebas en el cluster, con lo que obtendremos el rendimiento de las aplicaciones desarrolladas y verificaremos la correcta configuración de *Hadoop*.

5.1.6 Hito 6. Elaboración de la memoria

Por último, se redacta esta memoria, en la que se muestra el trabajo realizado en cada una de las etapas. Este hito se realiza paralelamente a las actividades descritas anteriormente.

5.2 Planificación del calendario y diagrama de Gantt

En este apartado se muestra la planificación realizada en el periodo de tiempo en el que se ha desarrollado el proyecto. En primer lugar, se muestra el tiempo dedicado a cada hito definido en el

apartado anterior, tanto en días como en horas y acto seguido se resumen esta información mediante un diagrama de *Gantt*. [77]

5.2.1 Planificación del calendario

El proyecto comenzó el día 26 de Septiembre de 2011 y se realizó en aproximadamente 243 días (obviando fines de semana, festivos, etc). En la siguiente tabla, mostramos el desglose de tiempo entre los 6 hitos principales del proyecto definidos en el apartado anterior.

Hito	Horas empleadas	Fecha inicio	Fecha fin	Duración (días)
1. Estudio inicial y estado del arte	145	26/09/11	23/12/11	65
2. Desarrollo de un extractor de datos de Twitter	150	03/10/11	13/01/12	75
3. Limpieza y filtrado de la traza	125	16/01/12	12/03/12	41
4. Análisis de los datos mediante Hadoop	145	06/03/12	29/05/12	60
5. Evaluación del trabajo realizado	85	04/06/12	16/07/12	31
6. Elaboración de la memoria	250	10/10/11	29/08/12	233

Tabla 19: Planificación del calendario de planificación de todos los hitos

Si sumamos el total de horas empleadas, nos da un total de 900 horas, que son, aproximadamente, las horas que se han empleado durante todo el trabajo realizado en el proyecto. En las siguientes tablas, se muestra la dedicación temporal y el calendario de tareas planificadas de cada uno de los hitos para el desarrollo del proyecto, es decir, el desglose de cada uno de los hitos con respecto a cada subtarea.

Hito 1. Estudio inicial y estado del arte

Tarea	Horas empleadas	Fecha inicio	Fecha fin
1-1. Análisis del problema	15	26/09/11	30/09/11
1-2. Investigación de las tecnologías y estado del arte	100	03/10/11	23/12/11
1-3. Realización de pruebas de concepto	20	03/10/11	14/10/11

Tabla 20: Calendario de planificación para el hito 1: Estudio inicial y estado del arte

Hito 2. Desarrollo de un extractor de datos de Twitter

Tarea	Horas empleadas	Fecha inicio	Fecha fin
2-1. Establecer requisitos del extractor	15	03/10/11	07/10/11
2-2. Arquitectura y diseño del extractor	20	10/10/11	14/10/11
2-3. Implementación del extractor	50	17/10/11	04/11/11
2-4. Ejecución, despliegue y monitorización del extractor	65	07/11/11	13/01/12

Tabla 21: Calendario de planificación para el hito 2: Desarrollo de un extractor de datos de Twitter

Hito 3. Limpieza y filtrado de la traza

Tarea	Horas empleadas	Fecha inicio	Fecha fin
3-1. Análisis previo de la traza	35	16/01/12	30/01/12
3-2. Limpieza de la traza	60	31/01/12	27/02/12
3-3. Filtrado de la traza	30	28/02/12	12/03/12

Tabla 22: Calendario de planificación para el hito 3: Limpieza y filtrado de la traza

Hito 4. Análisis de los datos mediante Hadoop

Tarea	Horas empleadas	Fecha inicio	Fecha fin
4-1. Configuración de Hadoop	15	06/03/12	13/03/12
4-2. Implementación de aplicaciones Hadoop para el análisis de datos	80	19/03/12	7/05/12
4-3. Elaboración de gráficas con los resultados obtenidos	50	07/05/12	29/05/12

Tabla 23: Calendario de planificación para el hito 4: Análisis de los datos mediante Hadoop

Hito 5. Evaluación del trabajo realizado

Tarea	Horas empleadas	Fecha inicio	Fecha fin
5-1. Evaluación del funcionamiento del extractor	35	04/06/12	18/06/12
5-2. Evaluación de las aplicaciones MapReduce	50	25/06/12	16/07/12

Tabla 24: Calendario de planificación para el hito 5: Evaluación del trabajo realizado

5.2.2 Diagrama de Gantt

La siguiente figura muestra la información anterior de forma gráfica mediante un diagrama de

Gantt. Este diagrama nos muestra gran cantidad e información de un solo vistazo, como por ejemplo:

- Los periodos de tiempo en los que se solapan dos o más tareas.
- Las dependencias entre las tareas.
- El tamaño de cada subtarea con respecto a la tarea conjunto o hito.

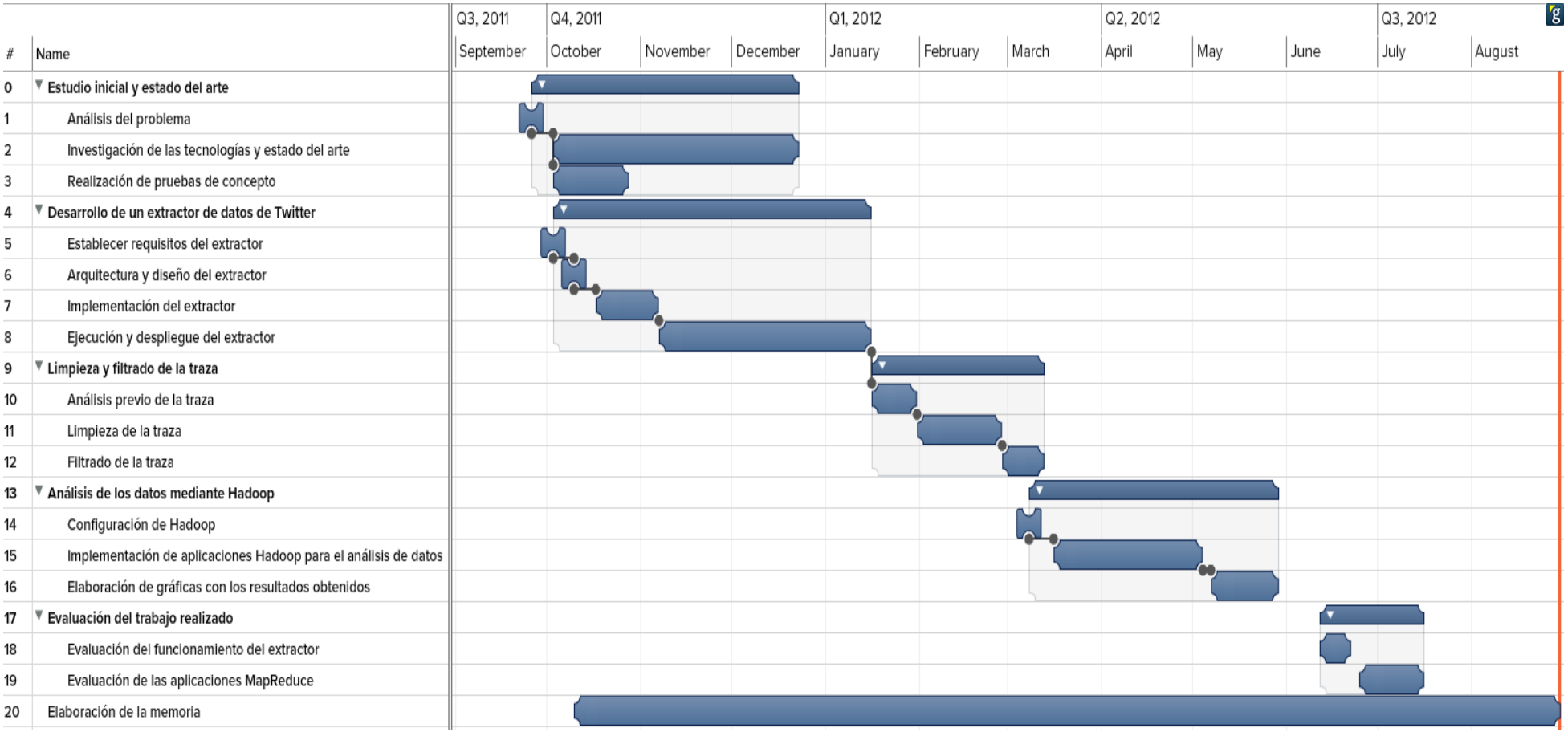

Figura 37: Diagrama de Gantt

Realizando un balance final del diagrama de *Gantt*, cabe destacar las tareas relacionadas con el análisis de datos utilizando *Hadoop*. En un principio, se estimó un tiempo bastante menor, ya que el aprendizaje de *Hadoop* se había obtenido en el hito anterior: limpieza y filtrado de la traza.

Sin embargo, esta tarea se complicó más de lo esperado con ciertos errores típicos de *Hadoop*. Aunque llevo tiempo solucionarlos, esto sirvió para reforzar el conocimiento sobre implementación de aplicaciones *Hadoop*, además de configuración y despliegue de estas tareas en un cluster.

5.3 Presupuesto

En este apartado se detalla el presupuesto del proyecto "*Extracción de información social desde Twitter y análisis mediante Hadoop*". Para calcularlo, se han dividido los costes del proyecto en costes personales y costes de material, siguiendo la plantilla de desglose presupuestario de la Escuela Politécnica superior de la Universidad Carlos III de Madrid. [78]

5.3.1 Costes de personal

A lo largo del desarrollo del proyecto, han participado las siguientes personas:

- **Cristian Caballero Montiel**: Con categoría de Ingeniero y con el rol de autor del proyecto.
- **Daniel Higuero Alonso-Mardones**: Con categoría de Ingeniero Senior y con el rol de tutor del proyecto.
- **Juan Manuel Tirado Martín**: Con categoría de Ingeniero Senior y con el rol de co-tutor del proyecto.

Siguiendo la plantilla presupuestaria [78]. Se indica que el sueldo de un ingeniero senior en la realización de 131.25 horas es de 4,289.54€, o lo que es lo mismo, 32.68€/hora, mientras que el sueldo de un ingeniero para el mismo número de horas es de 2,684.39€, es decir, 20.52€/hora. En la siguiente tabla se muestra el desglose de costes personales.

Apellidos y Nombre	Categoría	Dedicación (horas)	Coste (€/hora)	Coste total (€)
Caballero Montiel, Cristian	Ingeniero	900	20.52	18,468.00
Higuero Alonso-Mardones, Daniel	Ingeniero Senior	50	32.68	1,634.00
Tirado Martín, Juan Manuel	Ingeniero Senior	50	32.68	1,634.00
			SUBTOTAL	**21,706**
			TOTAL (+18% IVA)	**25,613.08**

Tabla 25: Costes de personal

Es decir, el total del coste de personal, con impuestos incluidos, asciende a un total de *veinticinco mil seiscientos trece con ocho céntimos* (*25,613.08*).

5.3.2 Costes de los materiales

Se han empleado los siguientes materiales durante la realización del proyecto:

- Un ordenador portátil modelo *HP Pavilion dv6* adquirido en el año 2012 con un importe de 600€ (IVA incluido).
- Un disco duro externo modelo *LG* de 1TB de capacidad adquirido en el año 2011 con un importe de 100€ (IVA incluido) y usado para tareas de backup.

La siguiente tabla muestra la imputación de costes de estos materiales:

Descripción	Coste (€)	% Uso dedicado al proyecto	Dedicación (meses)	Periodo de depreciación (meses)	Coste imputable (€)
Ordenador portátil HP Pavilion dv6	600	100	11	60	110
Disco duro externo LG 1TB	100	100	11	60	18.33
Cluster de cómputo 16 nodos	-	100	11	60	0*
Infraestructura de cloud privado	-	100	11	60	0*
				SUBTOTAL	**128.33**
				TOTAL (+18% IVA)	**151.42**

Tabla 26: Costes de los materiales

* Tanto el cluster de cómputo de 16 nodos como la infraestructura de cloud privado han sido puestas a disposición del proyecto de forma gratuita por parte de ARCOS, grupo de investigación de arquitectura de computadores y sistemas de la UC3M. [79]

Para el cálculo del coste imputable, se ha utilizado la siguiente fórmula:

$$\frac{A}{B} \times C \times D$$

donde:

- A = nº de meses desde la fecha de facturación en que el equipo es utilizado.
- B = periodo de depreciación (60 meses).
- C = coste del equipo
- D = % del uso que se dedica al proyecto (habitualmente 100%).

Es resumen, el total de los costes materiales asciende a *ciento cincuenta y uno con cuarenta y dos euros* (*151.42€*)

5.3.3 Presupuesto total

El calculo del presupuesto total viene dado por la suma de los costes más un porcentaje de

beneficio aplicado. En nuestro caso, se ha aplicado un porcentaje de un 15 % de beneficios sobre el total del coste del proyecto. La siguiente tabla muestra el cálculo del presupuesto total:

Tipo de coste	Coste (€)
Personal	25,613.08
Materiales	151.42
Subtotal	25,764.50
Beneficio (15%)	3,864.67
Total (Subtotal + Beneficio)	29,629.17

Tabla 27: Cálculo del presupuesto total

Por tanto, el presupuesto total del proyecto asciende a ***VEINITINUEVE MIL SEISCIENTOS VEINTINUEVE CON DIECISIETE EUROS (29,629.17 €)***. De los cuales, 3,864.67 se imputan en concepto de beneficio por el desarrollo del proyecto.

6. CONCLUSIONES

En este apartado se abordan las conclusiones que se han obtenido durante realizado el proyecto. Se abordan tanto las conclusiones generales como las conclusiones personales.

6.1 Conclusiones generales

En la propia definición del título del proyecto, ya vemos como lo dividíamos en dos objetivos claramente diferenciados: el diseño e implementación de un extractor masivo de información social con temática de la elecciones generales de 2011 sobre *Twitter* y su posterior análisis utilizando *Hadoop*. Por tanto, se dividen las conclusiones generales del proyecto en estos dos apartados:

6.1.1 Conclusiones de la extracción de datos

Lo primero que vamos a evaluar son las conclusiones obtenidas en cuanto a la extracción de datos. Esta etapa ha sido la más crítica en cuanto a todo el desarrollo del proyecto, ya que todo el análisis posterior venía determinado de lo obtenido en este primer hito.

La primera cuestión que se trató sobre la extracción de datos fue cual iba a ser la fuente de datos. Después de barajar varias opciones, rápidamente se decidió que *Twitter* era la apropiada para esto por dos razones principales. Por un lado, su contenido público cobra cada día mayor importancia. Instituciones y periodistas utilizan Twitter a diario como herramienta de trabajo, publicidad y difusión de información. Por otro lado, el API de *Twitter* está a disposición de cualquier desarrollador. Si bien tiene algunas limitaciones, estas no influyeron para el correcto avance del proyecto.

En cuanto a las decisiones de diseño para el desarrollo del extractor, la más decisiva fue la elección de una librería Java de las que implementa el API de *Twitter*. Finalmente, se decidió que la librería más apropiada era *Twitter4J*, por su sencillez de uso, facilidad de aprendizaje, fiabilidad y tolerancia a fallos.

A lo largo de la extracción de datos, aparecieron ciertas dificultades añadidas al correcto desarrollo del extractor. Una de las más destacable ocurría en el escenario en el que se almacenaban tweets con saltos de linea. Dado que la extracción de datos estaba pensada para que cada tweet se almacenara en una linea, no se barajó la posibilidad de que un usuario introdujera un tweet en el texto de este o tuviera un salto de linea en la descripción del usuario. Se soluciono de forma relativamente sencilla escribiendo un script en bash [80].

Otros impedimentos fueron los cortes de red que hubo en la infraestructura en la que se ejecutó el extractor. Esto provocó que algunos días las extracciones fueran erróneas, obteniendo menos tweets de los habituales o incluso ninguno para esos días concretos. Esto es un comportamiento

común para cualquier sistema de recuperación masiva de información que hay que tener en cuenta a la hora de realizar el posterior análisis de la traza. La solución parcial que se adoptó fue eliminar los datos atípicos y nulos de la traza. Si bien, fue necesario afinar el análisis posterior.

Dado que el extractor estuvo funcionando durante más de dos meses, fue necesario realizar un respaldo de datos para casos de desastre. En un primer momento, el respaldo consistía en copiar periódicamente los ficheros a otra máquina fuera del Cloud. Pronto se decidió que era necesario contar con algún otro backup, con lo que, además de seguir copiándolo a otra máquina, se empezaron a guardar los ficheros en un disco duro externo.

Como conclusión final del extractor de Twitter, en cuanto a los datos recuperados, cabe destacar la gran cantidad de datos recuperados: algo más de 89 GB, lo que hubiera sido algo inestable para una base de datos relacional. Una vez filtrados, se quedaron en 19 GB, es decir, únicamente un 20% de la traza. Teniendo en cuenta algunos de los términos que entraban dentro de la consulta al API de Twitter (*pp*, *cc*, etc.), no es de extrañar obtener mucho más tweets de los buscados a priori, ya que cualquier usuario utiliza esos términos en contextos diferentes al de las elecciones. Fue a raíz de esta conclusión por la que se decidió que era necesario realizar un filtrado de la traza para quedarnos únicamente con los tweets dentro del contexto de las elecciones.

6.1.2 Conclusiones del análisis de datos

Una vez obtenida y filtrada la traza, se realiza un análisis de datos para obtener tendencias o datos de interés sobre la información recuperada de *Twitter*. Esto se corresponde con el objetivo final del proyecto y pretende servir de ejemplo de como analizar gran cantidad de datos utilizando sistemas distribuidos y apoyándose en *Hadoop*.

Viendo los resultados, se puede observar las grandes ventajas que se obtienen al aplicar computación distribuida para el análisis masivo de datos, en especial en cuanto a tiempo de cómputo. Sin embargo, esto también puede tener ciertas desventajas:

- La programación distribuida suele ser conceptualmente complicada y difícil de implementar.
- El aprovechamiento total del HW disponible se complica con el número de nodos utilizados.
- No existe la escalabilidad perfecta en clusters de computo. A menudo, esta compite con con disponibilidad y rendimiento [81].

La primera la resuelve, sin lugar a duda, *MapReduce* y *Hadoop*, ya que, de un modo sencillo y con muy pocas lineas de código, podemos implementar programas completos con los que sacar conclusiones. En cuanto al aprovechamiento de HW y la escalabilidad, existen muchos factores a tener en cuenta ya que es muy difícil saber hasta qué punto utilizar más nodos significa obtener más rendimiento. Algunos de los factores más importantes para *Hadoop* son:

- El tamaño de bloque. Por defecto son 64 MB, sin embargo, es posible que para el caso del análisis de datos, en el que teníamos una entrada de 19 GB, hubiera sido más indicado disminuir el tamaño de bloque.

- El reparto de cada trozo a cada nodo de cómputo: A mayor número de nodos, mayor tiempo empleará el *jobtracker* o planificador de trabajos en “preparar” el entorno para la ejecución de la tarea. Además, el número de operaciones de E/S aumentará. Una posible mejora para esto puede ser optimizar la escritura de las máquinas que conforman el *HDFS* [82].

Hadoop tiene gran cantidad de parámetros ajustables a cada problema concreto, y a menudo buscar cual es el conjunto de parámetros más óptimo para cada problema puede llevar un largo tiempo de trabajo. Para el desarrollo del proyecto, no nos hemos volcado en este aspecto, aunque sin duda sería muy interesante probar distintos parámetros y analizar, por ejemplo el comportamiento del programa buscando patrones y monitorizando la utilización de los recursos de cada uno de los nodos.

Para el caso que nos ocupa podemos ver que somos capaces de obtener los resultados buscados procesando 19GB empleando 8 nodos y en algo más de 8 minutos (ver apartado 4.3.2). En este caso, buscar optimizar más los tiempos no tiene mucho sentido, ya que tenemos buenos resultados.

Como conclusión general sobre el análisis de datos, podemos decir que hemos obtenido todos los datos buscados utilizando un paradigma de programación distribuido y fácil de programar y/modificar en un periodo corto de tiempo. A modo de ejemplo, se han obtenido datos como el número de tweets por día, por usuario y por día/usuario, sin embargo, un pequeño cambio de pocas lineas en la función *Map* nos permitiría obtener mucha más información. Además, obtenemos esta información en muy poco tiempo. Sin embargo, hay que tener en cuenta que, si el tamaño de la traza fuera bastante mayor, nos encontraríamos con más frecuencia con los problemas de escalabilidad de los que hablábamos anteriormente, lo que necesitaría un mayor análisis de la solución adoptada. Por último, indicar que *MapReduce* en general y *Hadoop* en particular no es de propósito general. Es necesario un estudio previo para saber si implantar estas tecnologías en ciertos entornos tiene sentido para obtener los resultados que buscamos.

6.2 Conclusiones personales

Además de las conclusiones generales obtenidas a partir de la elaboración del proyecto sobre la extracción y el posterior análisis de datos, también se han obtenido ciertas conclusiones de carácter personal.

Por un lado, para todo el desarrollo del proyecto ha sido necesario adquirir ciertos conocimientos sobre las plataformas y tecnologías utilizadas. A grandes rasgos, los conocimientos más importantes adquiridos han sido los siguientes:

- Estado actual de las tecnologías de cloud-computing más utilizadas en la actualidad.
- Desarrollo de aplicaciones utilizando el API de *Twitter* con *Twitter4J*.
- Configuración y administración de *Hadoop*.
- Desarrollo de aplicaciones *MapReduce* utilizando *Hadoop*.

Otro de los factores que se han aprendido ha sido la gran importancia de la computación distribuida. Teniendo en cuenta que el volumen de datos aumenta exponencialmente, la computación distribuida es una de las mejores opciones para corresponder a esta gran demanda de datos.

Por último, desde un punto de vista mucho más personal, este TFG me ha ayudado entre otras cosas a decidir que rama dentro de la ingeniería informática me gusta más y me ha motivado para seguir trabajando y estudiando en la linea de investigación de la computación de altas prestaciones, distribuida y con procesamiento de gran cantidad de datos.

7. LINEAS FUTURAS

En este capítulo, se detallan posibles lineas de investigación y de trabajo a seguir como consecuencia del desarrollo de este proyecto.

A lo largo del desarrollo del proyecto, en especial en los momentos de redacción de la memoria, han ido surgiendo nuevas cuestiones a tratar que, por falta de tiempo, tendrán que quedar para medio-largo plazo. Si bien, es importante tenerlas en cuenta, ya que los resultados podrían ser de interés.

La mayoría de trabajo futuro a realizar estaría relacionado con la parte de análisis de datos con Hadoop, aunque se podría realizar cierto trabajo extra para la extracción de datos. Una primera aproximación nos lleva a las siguientes posibles lineas futuras para el extractor de *Twitter*:

- Diseñar el extractor con distintos lenguajes de programación y librerías para comparar el rendimiento de cada uno.
- Realizar una interfaz gráfica, para la cual un usuario cualquiera pudiera configurar de forma sencilla e intuitiva que usuarios y términos quiere seguir.
- Ejecutar el extractor en algún Cloud público.
- Extraer una nueva traza más grande con distinto contexto social y con campos distintos.
- Diseñar e implementar algún otro tipo de extractor de información para web. Algunos buenos ejemplos serían un extractor para otras redes sociales o un Crawler para construir índices inversos [83]. Esto además, se integraría perfectamente con el uso posterior de *Hadoop*.

En cuanto al análisis de datos, ya vimos en el apartado de las conclusiones cuales podían ser las tareas a realizar a corto plazo. Estas tareas vendrían dadas por refinar el análisis de la traza filtrada. En este contexto, se podrían realizar los siguientes estudios:

- Optimizar la ejecución de *Hadoop*. Para esto, ya hemos propuesto algunas alternativas, como podían ser ajustar el Hardware del cluster a nuestras necesidades o cambiar ciertos parámetros de la configuración de *Hadoop*, como el tamaño de bloque, el número de *maps* y *reduces* que ejecuta cada nodo en paralelo o el timeout de las tareas para caso de error.
- Realizar una interfaz gráfica para la configuración y despliegue de *Hadoop*. En la interfaz se podrían indicar los principales parámetros de *Hadoop*, las direcciones IP de las máquinas *master* y *slaves* y los directorios de entrada y salida de la aplicación *MapReduce*. Además, para el caso concreto de nuestra traza, se podría filtrar indicar cierta funcionalidad del *Map* únicamente indicando el campo del tweet a utilizar. Incluso se podría extender aun más la funcionalidad si, en lugar de indicar un solo campo, se pudiera escoger un conjunto de ellos.
- Analizar los tweets más retwitteados. Filtrando por el número de retweets, se podría obtener cuales son los tweets más retwitteados de la traza, lo cual nos proporcionaría un buen punto de partida para realizar un análisis de tendencias entre los usuarios analizados simplemente viendo el texto de estos tweets.

- Analizar el número de friends y de followers de los usuarios. En este punto, podríamos realizar dos tipos de estudios: por un lado, estudiar cuales son los usuarios con más friends y followers y por otro, se podrían estudiar las variaciones en número de friends y followers de ciertos usuarios. Para esto último, una vez realizada la tarea *MapReduce* en *Hadoop*, simplemente se programaría como una tarea periódica a repetir cada X días. Una buena opción es programar la ejecución de estas tareas utilizando *Cron* [84].
- Analizar el texto de los tweets para buscar modas. Por ejemplo, se podría ver cuales son los términos más repetidos dentro de los términos iniciales que utilizamos para la extracción de datos. Esto también puede ser un buen punto de partida para un análisis general de opinión.

Estos son solo algunos ejemplos, sin embargo, existen multitud de posibilidades para el análisis de la traza actual. La ventaja de todo esto es que, para analizar cualquier otra variable de la traza, sería suficiente con introducir un cambio mínimo en la implementación del *Mapper* de *Hadoop*.

En cuanto a medio-largo plazo, existen multitud de lineas futuras. En primer lugar, ya que existen multitud de soluciones de computación distribuida, una buena investigación sería utilizar cualquier otra y comparar los resultados con los obtenidos con Hadoop. El primer acercamiento en este aspecto es utilizar *GPUs* y *CUDA* [85] para este tipo de procesado paralelo.

Por último, uno de las lineas futuras más interesantes podría ser analizar las relaciones de usuarios. Mediante la traza, se podría elaborar un grafo dirigido, en el cual, cada nodo representaría un usuario y cada interconexión entre nodos representaría la relación semántica “usuario sigue a”. Una vez generado el mapa, se podrían realizar técnicas de análisis de redes sociales [86], y analizar métricas propias de la teoría de grafos.

8. REFERENCIAS

1. **Web 2.0**: http://ocw.uc3m.es/ingenieria-informatica/metodos-y-tecnicas-de-trabajo-corporativo/SoftwareSocial.pdf
2. **Facebook**: http://www.facebook.com/
3. **Hi5**: http://hi5.com/
4. **Google+**: https://plus.google.com
5. **Plurk**: http://www.plurk.com
6. **Tumblr**: http://www.tumblr.com
7. **Tim O'Reilly**: http://es.wikipedia.org/wiki/Tim_O'Reilly
8. **O'Reilly Media**: http://oreilly.com/
9. **Blogger**: www.blogger.com
10. **WordPress**: www.wordpress.org
11. **Wikipedia**: www.wikipedia.org
12. **Google Docs**: www.docs.google.com
13. **Issuu**: http://www.issuu.com/
14. **Calameo**: www.calameo.com
15. **Flickr**: www.flickr.com
16. **MetroFlog**: www.metroflog.com
17. **Slideshare**: www.slideshare.net
18. **Youtube**: www.youtube.com

19. **Vimeo:** www.vimeo.com

20. **Artículo Megaupload**: http://www.elmundo.es/elmundo/2012/01/19/navegante/1327002605.html

21. **MediaFire**: www.mediafire.com

22. **Rapidshare**: www.rapidshare.com

23. **MySpace**: www.myspace.com

24. **Tuenti**: www.tuenti.com

25. **Genius**: www.geniusnet.com

26. **Amazon**: www.amazon.com/

27. **Jack Dorsey**: www.wihe.net/2396/historia-jack-dorsey-twitter/

28. **Tráfico diario Twitter en Alexa.com**: http://www.alexa.com/siteinfo/twitter.com

29. **API REST**: https://dev.twitter.com/docs/api

30. **API Search**: https://dev.twitter.com/docs/using-search

31. **API Streaming**: https://dev.twitter.com/docs/streaming-apis

32. **Documentación oficial de Twitter developers**: https://dev.twitter.com

33. **Librerías del API de Twitter**: https://dev.twitter.com/docs/twitter-libraries

34. **Scribe**: https://github.com/fernandezpablo85/scribe-java

35. **TwitterAPIME**: http://kenai.com/projects/twitterapime/pages/Home

36. **Jtwitter**: http://www.winterwell.com/software/jtwitter.php

37. **Twitter4j**: http://twitter4j.org/en/index.html

38. **Capas de Cloud Computing:** http://putsub.com

39. **Google Apps**: http://www.google.com/apps/intl/es/business/

40. **ExchangeOnline:** http://www.microsoft.com/exchange/2010/es/xl/exchange-online.aspx

41. **SalesForce.com:** http://www.salesforce.com/es/

42. **Google App Engine**: https://developers.google.com/appengine/?hl=es

43. **Windows Azure:** http://www.windowsazure.com/es-es/

44. **Force.com:** http://www.force.com/

45. **IBM Blue Cloud:** http://www.ibm.com/cloud-computing/us/en/

46. **OpenStack**: http://www.openstack.org/

47. **Eucalyptus**: http://www.eucalyptus.com/

48. **OpenNebula**: http://opennebula.org/

49. **Amazon EC2**: http://aws.amazon.com/es/ec2/#instance

50. **Google Engine**: https://developers.google.com/

51. **BigTable en Linux Magazine**: http://www.linux-magazine.es/issue/39/047-050_PythonLM39.pdf

52. **Chang, F., Dean, J., Ghemawat, S., Hsieh, W.C., Wallach, D.A., Burrows, M., Chandra, T., Fikes, A., and Gruber**, *R.E. Bigtable: A distributed storage system for structured data. In Proceedings of the Seventh Symposium on Operating System Design and Implementation* (Seattle, WA, Nov. 6–8). Usenix Association, 2006

53. **Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Le- ung**. *The Google file system*. In 19th Symposium on Op- erating Systems Principles, pages 29–43, Lake George, New York, 2003.

54. **Presentación GFS:** http://www.cs.fsu.edu/~awang/courses/cis6935_s2004/google.ppt

55. **Dean, J. and Ghemawat, S**. *MapReduce: Simplified data processing on large clusters*. In Proceedings of the Sixth Symposium on Operating System Design and Implementation (San Francisco, CA, Dec. 6–8). Usenix Association, 2004

56. **Sitio oficial Hadoop**: http://hadoop.apache.org/

57. **White, Tom.** *Hadoop: The Definitive Guide*. s.l. : O'Reilly, 2009. ISBN: 978-0-596-52197-4

58. **Lista de instituciones que usan Hadoop**: http://wiki.apache.org/hadoop/PoweredBy

59. **Discusión sobre las restricciones del API de Streaming de Twitter:** https://dev.twitter.com/discussions/4120

60. **UML**: http://www.uml.org/

61. **Log4J**: http://logging.apache.org/log4j/1.2/

62. **Comandos básicos de Eucalyptus:** http://open.eucalyptus.com/wiki/Euca2oolsVmControl

63. **JAR:** http://docs.oracle.com/javase/1.4.2/docs/guide/jar/jar.html

64. **Limitación de peticiones al API de REST**: https://dev.twitter.com/docs/rate-limiting

65. **Lista de palabras que contienen "pp":** http://www.morewords.com/contains/pp/

66. **TORQUE:** http://www.clusterresources.com/torquedocs21/p.introduction.shtml

67. **Gestión de trabajos en TORQUE:** http://www.clusterresources.com/torquedocs21/2.1jobsubmission.shtml

68. **Configuración Hadoop en un cluster:** http://hadoop.apache.org/common/docs/r0.20.203.0/cluster_setup.html

69. **LangDetect:** http://code.google.com/p/language-detection/

70. **Desviación típica:** http://www.disfrutalasmatematicas.com/datos/desviacion-estandar.html

71. **Posibles causas pico de Tweets 13/01/12:** http://ccaa.elpais.com/ccaa/2012/01/13/madrid/1326492551_422151.html

72. **Debate electoral 7 de Noviembre de 2011:** http://es.wikipedia.org/wiki/Elecciones_generales_de_Espa%C3%B1a_de_2011#Debates_electorales

73. **JfreeChart:** http://www.jfree.org/jfreechart/

74. **Junit:** http://www.junit.org/

75. **Commodity Hardware:** http://silvertonconsulting.com/blog/2010/11/05/commodity-hardware-always-loses/

76. **WordCount Hadoop**: http://hadoop.apache.org/common/docs/r0.20.2/mapred_tutorial.html

77. **Diagrama de Gantt**: http://*es.wikipedia.org/wiki/Diagrama_de_Gantt*

78. **Plantilla presupuesto Proyecto Fin de Carrera y Trabajo Fin de Grado:** http://www.uc3m.es/portal/page/portal/administracion_campus_leganes_est_cg/proyecto_fin_carrera/Formulario_PresupuestoPFC-TFG%20(3)_2.xlsx

79. **ARCOS:** http://www.arcos.inf.uc3m.es/doku.php?id=inicio

80. **Sustituir saltos de linea por carácter:** http://www.lawebdelprogramador.com/foros/Linux_Unix_Shell_Scripting/1290197-Sustituir_una_cadena_por_un_salto_de_linea.html

81. **Escalabilidad vs disponibilidad vs rendimiento en clusters:** http://www.idg.es/computerworld/Disponibilidad,-escalabilidad-y-rendimiento,-tres-/seccion-/articulo-58781

82. **Optimizaciones para correr más rápido Hadoop:** http://natishalom.typepad.com/nati_shaloms_blog/2012/08/making-hadoop-run-faster.html

83. **Índices inversos:** http://ocw.uc3m.es/ingenieria-informatica/recuperacion-y-acceso-a-la-informacion/material-de-clase-1/01-IntroducciOn.pdf

84. **Cron:** http://www.adminschoice.com/crontab-quick-reference

85. **CUDA:** http://www.nvidia.es/object/cuda_home_new_es.html

86. **Análisis de redes sociales**: http://faculty.ucr.edu/~hanneman/nettext/

87. **Palacios Díaz-Zorita, Mª Carmen. Evaluación de la herramienta de código libre Apache Hadoop**: http://e-archivo.uc3m.es/bitstream/10016/13533/1/MemoriaPFC_MCarmenPalacios.pdf

88. **Cabañas Sánchez, Guillermo. Extracción y análisis de información del servicio de red social Twitter, a través de la plataforma de "cloud computing" Google App Engine:** http://e-archivo.uc3m.es/handle/10016/12973