Trabajo de Fin de Título

Análisis de sentimiento masivo con arquitectura de big data

TITULACIÓN: Ingeniería Informática

AUTOR: Jorge Vega Sánchez

TUTORIZADO POR:
Javier Jesús Sánchez Medina

Fecha 05/2024

Agradecimientos

A mis padres por su apoyo y confianza en mí, a mi tutor, Javier Jesús Sánchez Medina, por su orientación y asesoramiento durante este proyecto y a la Escuela de Ingeniería Informática de la ULPGC, por brindarme las herramientas y el entorno necesarios para mi formación académica.

Índice general

Índice de figuras

Índice de Algoritmos

Resumen

El desarrollo de un aplicativo de alto rendimiento para la descarga masiva y en línea de resúmenes de la plataforma *Crossref*, partiendo de un tópico de búsqueda, requiere el uso de técnicas avanzadas de computación paralela y *Big data [1]*. Este proyecto recopila datos de *Crossref [2]* utilizando DOI como identificadores únicos, mediante consultas de metadatos *web*. Los datos se almacenan en un sistema de archivos distribuido de *Hadoop* dispuesto en un clúster, lo que permite manejar grandes volúmenes de información de manera eficiente.

Para el procesamiento en tiempo real, se implementa un flujo con *Apache Spark Streaming* para la ingestión de datos. El *pipeline* de análisis de datos en línea incluye análisis de sentimiento y frecuencia de palabras repetidas, evaluando la actitud de los documentos por sus DOI. Los metadatos se integran en el análisis para proporcionar visualizaciones contextualizadas, facilitando la comprensión dinámica de los resultados.

Estas visualizaciones se despliegan en un servidor web de *Django*, permitiendo una interacción en tiempo real con los datos. Se construyen visualizaciones interactivas, como gráficos y word clouds, que permiten explorar eficazmente los datos. De este modo, el aplicativo ofrece una herramienta poderosa para analizar y visualizar resúmenes de documentos académicos, mejorando la accesibilidad y el entendimiento de la información.

Abstract

The development of a high-performance application for the massive and online download of abstracts from the Crossref platform, based on a search topic, requires the use of advanced parallel computing and big data techniques. This project collects data from Crossref using DOIs as unique identifiers through web metadata queries. The data is stored in a Hadoop distributed file system deployed on a cluster, allowing efficient handling of large volumes of information.

For real-time processing, a flow with Apache Spark Streaming is implemented for data ingestion. The online data analysis pipeline includes sentiment analysis and repeated word frequency, evaluating the sentiment of the documents by their DOIs. The metadata is integrated into the analysis to provide contextualized visualizations, facilitating the dynamic understanding of the results.

These visualizations are deployed on a Django web server, allowing real-time interaction with the data. Interactive visualizations, such as graphs and word clouds, are built to efficiently explore the data. Thus, the application offers a powerful tool for analyzing and visualizing academic document abstracts, improving the accessibility and understanding of information.

Capítulo 1

Introducción

> ”¡Los datos! ¡Los datos! Los datos!“ , gritó con impaciencia. “¡No puedo hacer ladrillos sin arcilla!”
>
> Sherlock Holmes (Arthur Conan Doyle)

1.1. Estado actual y objetivos iniciales

Según un informe de la empresa IDC en colaboración con *Seagate*, se espera que en 2025 se hayan generado 175 Zetabytes de información [5].

La ilustración 1.1 muestra el continuo incremento en Zetabytes de la cantidad de datos que se manejan cada año en todo el mundo.

Esto es base más que suficiente para deducir la suma importancia de poder tratar estos datos, generándolos, almacenándolos, procesándolos o visualizándolos. Por ello, han surgi-

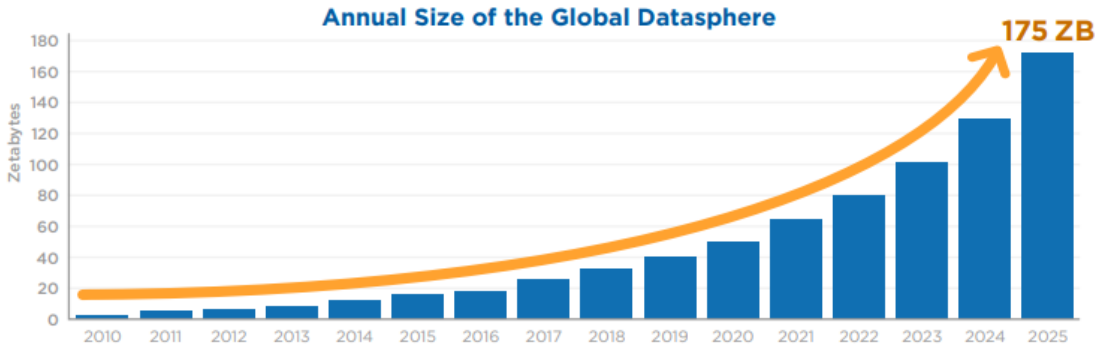

Ilustración 1.1: *Source: Data Age 2025, sponsored by Seagate with data from IDC Global DataSphere, Nov 2018*

do varias titulaciones a lo ancho del mundo para aprender cómo gestionarlos y numerosas empresas han aprendido la utilidad de esa información, por ejemplo, para recomendar a sus clientes de los productos que se adecuan más a sus intereses basándose en antiguas búsquedas en internet, su rango de edad o por sus amigos y familiares.

Dado a este consumo de datos, se busca la inmediatez para procesarlos y visualizarlos sin que se vean comprometidos por quedarse anticuados en cuestión de horas, por ello, también hay que destacar el papel del Procesamiento en tiempo real [3].

Crossref ingresa en su base de datos nueva información todos los días relacionados con artículos académicos. De esos artículos se obtiene sus metadatos, concretamente DOI [4], título, autores, resumen y fecha de publicación. Se emplea un procesador de texto para obtener información del resumen y por último se representa la información obtenida y procesada en tiempo real.

Este proyecto se propone como objetivo:

- ✔ Recopilar datos de la *web* de *Crossref* utilizando sus servicios de consulta de metadatos, centrándose en los DOI como identificadores únicos.
- ✔ Implementar un flujo de procesamiento en tiempo real utilizando *Apache Spark Streaming*, *Kafka* o equivalentes, para la ingestión de datos.
- ✔ Desarrollar un *pipeline* de análisis de datos en línea (minería de datos en *streaming*), que puede incluir análisis de sentimiento que permita evaluar la polaridad o actitud de los documentos representados por los DOI, u otros análisis textuales.
- ✔ Integrar los metadatos de los documentos en el análisis para proporcionar una visualización contextualizada de los resultados.
- ✔ Construir visualizaciones interactivas que permitan explorar y comprender los datos de manera efectiva y dinámica.

1.2. Motivación

La motivación principal de este proyecto radica en la creciente necesidad de herramientas que puedan procesar y analizar grandes volúmenes de datos textuales de manera eficiente. En el ámbito académico y de investigación, la capacidad de acceder y analizar rápidamente los resúmenes de publicaciones científicas puede acelerar significativamente el avance del conocimiento. La capacidad de analizar grandes volúmenes de datos textuales para extraer información valiosa sobre las opiniones y sentimientos de los usuarios está siendo muy solicitada por las empresas e instituciones. Este proyecto se motiva por la necesidad de contar con herramientas eficientes y escalables para procesar y analizar datos masivos en tiempo real, aprovechando tecnologías de *big data* y *machine learning*.

1.3. Estructura del documento

Capítulo 1: Introducción. Se presenta el contexto del proyecto, incluyendo el estado actual de la tecnología y las necesidades que motivan su desarrollo. También se detallan las motivaciones para llevar a cabo este trabajo, así como su relevancia en diferentes áreas. Finalmente, se enuncian los objetivos específicos del proyecto y la metodología general que se seguirá para alcanzarlos.

Capítulo 2: Competencias específicas y aportaciones del trabajo. Se describen las competencias técnicas y conocimientos adquiridos y aplicados durante el desarrollo del proyecto. Se detallan las habilidades en el uso de tecnologías de *Big Data*, computación paralela y análisis de datos. Además, se explican las principales aportaciones del trabajo, tales como:

- ✔ La creación de una arquitectura de *Big Data* escalable.
- ✔ El desarrollo de un sistema de análisis de sentimientos a gran escala.
- ✔ La automatización de *pipelines* de procesamiento de datos.
- ✔ La integración de metadatos para un análisis contextualizado.
- ✔ La implementación de visualizaciones interactivas.

Capítulo 3: Desarrollo. Se centra en la implementación del sistema, detallando cada una de las fases del desarrollo:

- ✔ **Arquitectura del clúster:** Descripción de la arquitectura general del sistema, incluyendo el uso de *Hadoop* para el almacenamiento distribuido y *Spark* para el procesamiento en memoria. Se explican las configuraciones del clúster y la integración de los componentes.
- ✔ **Obtención de datos:** Se define qué es *Crossref* y se describen los pasos y la librería necesaria para descargar sus metadatos y adecuarlos para posteriormente procesarlos, almacenándolos en el sistema de archivos distribuido de *Hadoop*.
- ✔ **Procesamiento de datos:** Se incluyen las técnicas de procesamiento en tiempo real con *Spark Streaming*. La descripción de los métodos utilizados para el análisis de sentimientos y la frecuencia de palabras. Todo ello para optimizar el procesamiento en concurrente de los datos.
- ✔ **Visualización de resultados:** Detalle de las herramientas y técnicas utilizadas para la visualización de los resultados. Se describen algunos ejemplos para explicar el comportamiento de los datos procesados en gráficos y nubes de palabras, y su integración en un servidor *web Django*.

Capítulo 4: Conclusiones y trabajos futuros. Se resumen las conclusiones principales del proyecto, evaluando el cumplimiento de los objetivos propuestos y la contribución al campo de estudio. Se proponen posibles direcciones para futuros trabajos, incluyendo:

- ✔ Mejoras en la arquitectura del sistema para incrementar la eficiencia.

- ✔ Extensión del análisis a otros tipos de datos y dominios.
- ✔ Desarrollo de nuevas funcionalidades de visualización y análisis.

Capítulo 2

Competencias específicas y aportaciones del trabajo

2.1. Competencias

- ✔ **CI12**: Conocimiento y aplicación de las características, funcionalidades y estructura de las bases de datos, que permitan su adecuado uso, y el diseño y el análisis e implementación de aplicaciones basadas en ellos.
- ✔ **CI13**: Conocimiento y aplicación de las herramientas necesarias para el almacenamiento, procesamiento y acceso a los sistemas de información, incluidos los basados en *web*.
- ✔ **CI14**: Conocimiento y aplicación de los principios fundamentales y técnicas básicas de la programación paralela, concurrente, distribuida y de tiempo real.

  Aunque durante en la elaboración del proyecto se han tratado también las siguientes competencias:

  CI7 Conocimiento, diseño y utilización de forma eficiente de los tipos y estructuras de datos más adecuados a la resolución de un problema.
- ✔ **CI8** Capacidad para analizar, diseñar, construir y mantener aplicaciones de forma robusta, segura y eficiente, eligiendo el paradigma y los lenguajes de programación más adecuados.
- ✔ **CI9** Capacidad de conocer, comprender y evaluar la estructura y arquitectura de los computadores, así como los componentes básicos que los conforman.
- ✔ **CI10** Conocimiento de las características, funcionalidades y estructura de los sistemas operativos y diseñar e implementar aplicaciones basadas en sus servicios.
- ✔ **CI11** Conocimiento y aplicación de las características, funcionalidades y estructura de los sistemas distribuidos, las redes de computadores e *Internet* y diseñar e implementar aplicaciones basadas en ellas.

2.2. Aportaciones

Este proyecto aporta varias contribuciones significativas:

✔ **Desarrollo de una arquitectura de *Big Data***

La implementación de una arquitectura robusta de *Big Data* utilizando tecnologías como *Apache Hadoop* y *Spark* es una contribución significativa de este proyecto. Esta arquitectura es capaz de manejar grandes volúmenes de datos y realizar análisis en tiempo real. Al aprovechar las capacidades de almacenamiento y procesamiento distribuidos de *Hadoop* y la computación en memoria de *Spark*, el sistema garantiza un procesamiento de datos eficiente y escalable. Esto permite obtener conocimientos e interpretaciones rápidas de grandes conjuntos de datos, lo cual es crítico para el análisis de sentimientos en tiempo real y otras aplicaciones que requieren una interpretación inmediata de los datos.

✔ **Análisis de sentimientos a gran escala**

La implementación de un sistema capaz de realizar análisis de sentimientos en grandes conjuntos de datos textuales es otra aportación vital. Este sistema puede procesar y analizar sentimientos de vastas cantidades de datos, proporcionando visualizaciones claras y útiles de los resultados. El análisis de sentimientos ayuda a comprender la actitud general o el tono emocional de los datos textuales, lo cual puede ser invaluable para diversas aplicaciones como, por ejemplo, el análisis de la evolución del lenguaje en la sociedad. La escalabilidad de este sistema asegura que pueda manejar cantidades crecientes de datos sin pérdida de rendimiento.

✔ ***Pipeline* de procesamiento automatizado**

La creación de un *pipeline* de procesamiento automatizado representa un avance significativo en el proyecto. Este pipeline facilita la ingestión, procesamiento y análisis continuo de datos, minimizando la necesidad de intervención manual. La automatización garantiza que los datos se procesen de manera consistente en tiempo real, manteniendo alta precisión y velocidad en el análisis. Esto contribuye a obtener conocimientos más fiables y oportunos, lo cual es esencial para la toma de decisiones en entornos dinámicos donde los datos están en constante cambio. La creación de un pipeline de procesamiento automatizado representa un avance significativo en el proyecto. Este pipeline facilita la ingestión, procesamiento y análisis continuo de datos, minimizando la necesidad de intervención manual. La automatización garantiza que los datos se procesen de manera consistente en tiempo real, manteniendo alta precisión y velocidad en el análisis. Esto contribuye a obtener conocimientos más fiables y oportunos, lo cual es esencial para la toma de decisiones en entornos dinámicos donde los datos están en constante cambio.

✔ **Integración de metadatos para análisis contextual**

La integración de metadatos en el análisis proporciona una comprensión más profunda del contexto de los datos procesados. Al incluir metadatos, el sistema puede ofrecer información más completa, como el origen de los datos y el momento en que fueron

creados. Esto enriquece el análisis al permitir que los usuarios interpreten los resultados en un contexto más amplio, haciendo que la información sea más relevante y útil para necesidades y escenarios específicos.

- ✔ **Visualización en tiempo real y exploración interactiva**

  El desarrollo de herramientas para visualización en tiempo real y exploración interactiva también es una gran contribución del proyecto. Las visualizaciones, como gráficos y nubes de palabras, permiten a los usuarios explorar y comprender los datos de manera más efectiva. La interactividad facilita que los usuarios profundicen en detalles específicos, descubriendo patrones y conocimientos que podrían no ser evidentes en informes estáticos. Esto mejora la experiencia del usuario y proporciona una comprensión más intuitiva de conjuntos de datos complejos.

Capítulo 3

Desarrollo

3.1. Organización

Este proyecto tiene varias fases por las que circulan los datos. Primeramente, se despliega un clúster con *Apache Hadoop* y como gestor de recursos *Apache Spark*. Los datos se obtienen de la *web* de *Crossref* hasta la fecha y luego se descargan los datos diariamente de la misma, almacenándose en el clúster de forma distribuida. Se procesan en un programa en *Scala* que estará continuamente en ejecución para proporcionar, a través de un *socket*, todos los datos que requiera el cliente sobre la media de sentimiento de una palabra entre dos fechas dadas de todos los resúmenes, o las cien palabras más frecuentes. Esto último se visualiza en un servidor *web* desarrollado con *Django* que lee del *socket* todos los datos procesados en tiempo real. 3.1

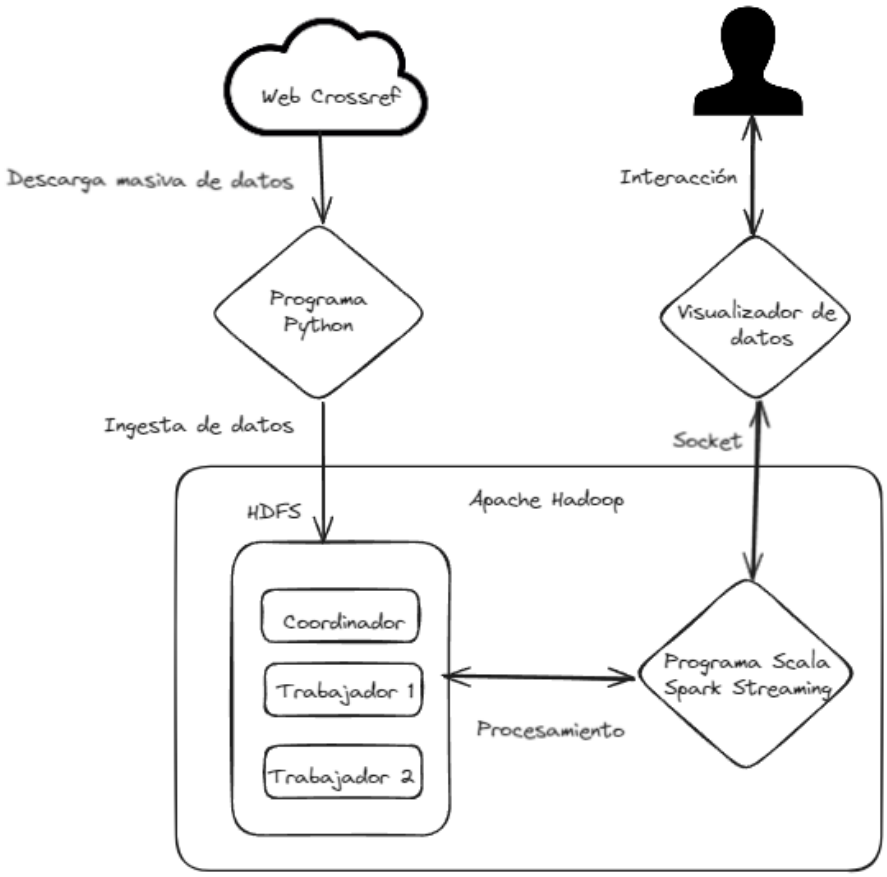

Ilustración 3.1: Recorrido de los datos durante la ejecución del proyecto.

3.2. Tecnologías

3.2.1. Apache Hadoop

Apache Hadoop es un *framework* que permite, a través de modelos de programa simples, establecer en varias máquinas dentro de un clúster una gran cantidad de datos, procesándolos de forma distribuida. [15]

Comparado con *Apache Flink*, esta tecnología no proporciona ningún sistema de ficheros distribuido, es necesario porque se guarda un histórico de anteriores ejecuciones para poder analizarse en un futuro y al ser distribuido es más tolerante al borrado de información.

La versión empleada en este proyecto es la 3.2.4.

3.2.2. HDFS

'*Hadoop Distributed File System*' (HDFS) es un sistema de ficheros distribuido entre varias máquinas que en este caso conforman un clúster. Es tolerante a fallos y diseñado para poder utilizarse en máquinas con pocos recursos. [13]

Comparado con *Amazon S3*, HDFS se integra de manera nativa con *Hadoop*, por lo que significa que no hay otra tecnología capaz de ofrecer mejor rendimiento.

3.2.3. YARN

'*Yet Another Resource Negotiator*'(YARN) es un *framework* que permite a Hadoop soportar varios motores de ejecución, proporcionando un planificador y un gestor de recursos. [14]

Comparado con otros gestores de recursos como *Kubernetes*, YARN está específicamente optimizado para trabajar con *Hadoop*, lo que lo hace ideal para gestionar trabajos de procesamiento de datos en entornos de Big Data.

3.2.4. Apache Spark

Apache Spark es un motor multilenguaje para ejecutar grandes cantidades de datos en clústeres. En este caso se usará para poder usar tecnologías de procesamiento de datos como *SparkSQL* y como '*Structured Streaming Programming*'. [24]

En comparación con *Apache Storm*, *Spark* tiene la capacidad de procesar tanto información en lotes como en tiempo real.

La versión empleada en este proyecto es la 3.2.4.

3.2.5. Streaming Programming

'*Streaming Programming*' es un paradigma de programación que permite el procesamiento de datos en tiempo real, gestionando y analizando flujos de datos a medida que se generan. Esto es esencial para aplicaciones que requieren un análisis continuo y en tiempo real. [19]

Otra tecnología posible es usar el procesamiento por lotes, esto es incompatible con el objetivo de dar respuestas en tiempo real de los datos.

La versión empleada en este proyecto es la 3.2.4.

3.2.6. SparkSQL

SparkSQL es un módulo de *Apache Spark* para el procesamiento de datos estructurados. Proporciona una interfaz de programación que permite trabajar con datos estructurados y semiestructurados usando SQL. [23]

En comparación con *Hive*, que también puede realizar operaciones SQL sobre *Hadoop*, *SparkSQL* ofrece una mayor velocidad y eficiencia al aprovechar el procesamiento en memoria de *Spark*.

La versión empleada en este proyecto es la 3.2.4.

3.2.7. Spark NLP

Spark NLP permite a las máquinas comprender y analizar el lenguaje humano. En este proyecto, se utiliza para realizar análisis de sentimientos en los resúmenes de los metadatos de *Crossref*, proporcionando valores de positivo o negativo.

Comparado con herramientas como NLTK, SpaCy y *Stanford* NLP, *Spark* NLP se destaca por su capacidad para manejar grandes volúmenes de datos de manera eficiente, gracias a su optimización para entornos de Big Data y su capacidad de procesamiento distribuido. Mientras que NLTK y SpaCy son potentes y ampliamente utilizadas para procesamiento de lenguaje natural, no están diseñadas para gestionar grandes cantidades de datos con la misma eficacia que *Spark* NLP, lo que hace que esta última sea ideal para proyectos de análisis de texto a gran escala.

La versión empleada en este proyecto es la 5.3.3.

3.2.8. RDDs

Los RDD (*Resilient Distributed Datasets*) son una de las características más distintivas de *Apache Spark*. Son colecciones distribuidas de datos, resistentes a fallos y altamente escalables. Un RDD es inmutable, lo que significa que una vez que se crea, no se pueden modificar sus elementos. [6]. Se usa en la búsqueda de actualizaciones de los datos en el HDFS.

En comparación con otras estructuras de datos distribuidas como los *DataFrames*, que también se utilizan en este proyecto, los RDD proporcionan un control más detallado sobre las operaciones de procesamiento.

3.2.9. Computación paralela

Consiste en el uso simultáneo de varios procesadores o núcleos que ejecutan una serie de instrucciones que conforman las distintas partes en las que se ha descompuesto un problema computacional. [25]. En este proyecto se ha logrado este propósito de la mano de *Apache Hadoop* y *Apache Spark* pudiendo utilizar los núcleos del clúster paralelamente.

La computación secuencia está totalmente descartada, ya que resulta muy lenta cuando se trata de grandes volúmenes de datos.

3.3. Herramientas

3.3.1. Visual Code Studio

Visual Studio Code es un editor de código fuente desarrollado por *Microsoft*. Es de gran utilidad, ya que se puede instalar extensiones para distintos lenguajes de programación, es muy versátil. [4]

Se ha elegido esta herramienta frente a otras por su capacidad de integración con múltiples herramientas y extensiones que hace que sea ideal para proyectos complejos.

3.3.2. Fedora

Fedora es una distribución de *Linux* utilizada en el entorno de desarrollo por su comodidad. [7]

Se ha elegido esta herramienta frente a otras por permitir una gestión eficiente de las dependencias y la configuración del entorno de desarrollo, facilitando la instalación y el uso de herramientas como *Apache Hadoop* y *Apache Spark*.

La versión empleada en este proyecto es la 40.

3.3.3. Mozilla Firefox

Mozilla Firefox es un navegador *web* de código abierto y libre, desarrollado por la Fundación Mozilla.

Se ha elegido esta herramienta frente a otras por su capacidad para soportar herramientas de desarrollo web integradas, como el inspector de elementos y la consola de *JavaScript* que facilita la depuración de aplicaciones *web*.

Instalado por defecto por *Fedora*. [10]

3.3.4. Máquinas CICEI

✔ **Nombre:** abaco.cicei.com

- **RAM:** 16 GB
- **CPU:** Intel(R) Xeon(R) E-2286G, 12 cpus
- **Arquitectura:** x86_64
- **S.O.:** Ubuntu 22.04.4 LTS
- **Almacenamiento:** 2 TB

✔ **Nombres:** liber.cicei.com y abaci.cicei.com

- **RAM:** 2 TB
- **CPU:** AMD EPYC 7662, 256 cpus
- **Arquitectura:** x86_64
- **S.O.:** Ubuntu 22.04.4 LTS
- **Almacenamiento:** 9 TB

3.3.5. Python

Python es un lenguaje de programación de alto nivel, interpretado y de propósito general, conocido por su simplicidad y legibilidad. Es ampliamente utilizado en análisis de datos, desarrollo web, automatización de tareas, y mucho más. [11]

La versión empleada en este proyecto es la 3.8.

3.3.6. Scala

Scala es un lenguaje de programación que combina características de programación funcional y orientada a objetos. Es conocido por su capacidad de escalar aplicaciones de manera eficiente, de ahí su nombre. [20]

Se ha elegido esta herramienta frente a otras por su utilidad para desarrollar las aplicaciones de procesamiento de datos, aprovechando su integración nativa con *Apache Spark* para manejar grandes volúmenes de datos de manera eficiente.

La versión empleada en este proyecto es la 2.12.15.

3.3.7. SBT

SBT (*Scala Build Tool*) es una herramienta de construcción para proyectos en *Scala*. Facilita la gestión de dependencias y la automatización de tareas de construcción, pruebas y despliegue. [3]

Se ha elegido esta herramienta frente a otras por la facilidad para gestionar el ciclo de vida del proyecto, desde la compilación hasta la generación de artefactos desplegables, como archivos .jar, esenciales para ejecutar aplicaciones Scala en el clúster.

La versión empleada en este proyecto es la 1.10.0

3.3.8. Git

Git es un sistema de control de versiones distribuido que permite a los desarrolladores rastrear cambios en el código fuente durante el desarrollo de *software*. Es especialmente útil para la colaboración en equipo y la gestión de versiones del proyecto. [12]

3.3.9. Java

Java es un lenguaje de programación de propósito general, concurrente, orientado a objetos y basado en clases. Es ampliamente utilizado en aplicaciones empresariales y móviles debido a su portabilidad y robustez. [16]

Esta herramienta ha sido elegida, ya que es requerida por *Hadoop* y por *Spark* para su ejecución.

La versión empleada en este proyecto es la java-1.8.0-411.

3.3.10. Django

Django es un *framework* de desarrollo *web* de alto nivel y de código abierto, escrito en *Python*. Está diseñado para facilitar la creación de sitios *web* complejos y basa su arquitectura en el patrón Modelo-Vista-Controlador (MVC).

Se ha elegido esta herramienta frente a otras por su facilidad para crear un servidor *web* y para poder visualizar los datos de forma dinámica.

La versión empleada en este proyecto es la Django 5.0.

3.3.11. Habanero

Habanero es una librería que proporciona una lista de comandos útiles para poder gestionar los metadatos que dispone *Crossref*. Esa librería se obtiene de un repositorio *Git* en donde se explican los pasos para descargarla. https://github.com/sckott/habanero

En este caso no ha habido ninguna preferencia frente las otras herramientas que ofrece *Crossref* para *Python*, ya que proporcionan una API para el manejo de sus datos de una forma clara y rápida.

La versión empleada en este proyecto es el habanero 1.2.6.

3.3.12. Spark Streaming

Spark Streaming es una funcionalidad esencial para la API de *Apache Spark* que proporciona flujos de datos en directo que son escalables y tolerantes a fallos. Esto ayuda al motor

de *Apache Spark* a soportar cargas de trabajo de forma más eficaz. [19]

Esta herramienta es de obligatorio uso si se quiere manejar datos que se generan continuamente, permitiendo análisis en tiempo real y soportando cargas de trabajo dinámicas, ya que es la proporcionada por *Apache Spark*, esto fomenta a que todas las funcionalidades estén totalmente optimizadas.

3.4. Arquitectura del clúster

Para la realización de este proyecto se ha usado un Clúster [5] 3.5 proporcionado por el CICEI que consta de tres máquinas 3.3.4 que comparten una red privada y es en las que se almacenan todos los datos que se procesan y donde se ejecutan paralelamente los datos. Es necesario configurar e instalar el clúster de acuerdo con los recursos disponibles para poder utilizar las herramientas de *Apache Hadoop* y *Apache Spark*.

Apache Hadoop 3.2 es un software de código abierto que facilita el procesamiento distribuido de enormes conjuntos de datos en clústeres. *Hadoop Distributed File System* se usa para almacenar datos de manera redundante y utiliza *MapReduce [6]* para el procesamiento paralelo de datos. Esta organización garantiza la accesibilidad de los datos y permite ejecutar tareas en múltiples máquinas.

Ilustración 3.2: Logo Hadoop, fuente Wikipedia.org

Para este proyecto, se ha optado por *Hadoop* debido a su capacidad para movilizar grandes cantidades de datos. Además, su capacidad de adaptación a diversos tipos de datos y su gran ventaja al poder ejecutarse en equipos con bajos recursos, lo convierten en una opción muy válida para el almacenamiento y procesamiento de datos masivos.

Apache Spark 3.3 es un motor de análisis de datos en memoria, elegido por su versatilidad y su rapidez de ejecución. Ofrece API en varios lenguajes de programación y soporta varios tipos de procesamiento de datos, incluyendo *batch*, *streaming*, *machine learning [7]* y SQL. Su capacidad de procesar datos en memoria lo hace significativamente más rápido que los modelos anteriores que dependen del disco, permitiendo análisis rápidos y eficientes.

Ilustración 3.3: Logo Spark, fuente Wikipedia.org

Spark ha sido elegido para este proyecto debido a su velocidad, simplicidad y versatilidad. Su integración con *Hadoop* y su capacidad para manejar diversos tipos de procesamiento de datos hacen de *Spark* una herramienta ideal para el análisis de grandes volúmenes de datos en tiempo real. Además, los módulos de *Spark* como *Spark* SQL, *Spark Streaming* y MLlib permiten tratar los volúmenes de distinta manera.

Big Data 3.4 se refiere a conjuntos de datos extremadamente grandes y complejos que no pueden ser manejados con herramientas y técnicas tradicionales de bases de datos. Implica la captura, almacenamiento, análisis y visualización de datos a gran escala, permitiendo obtener *insights* insights [8] de grandes volúmenes de información diversa y rápida. El objetivo principal de *Big Data* es permitir la toma de decisiones informadas a partir del análisis de datos masivos.

Ilustración 3.4: Imagen *BigData*, fuente Wikipedia.org

El enfoque de *Big Data* ha sido adoptado en este proyecto para manejar y analizar grandes volúmenes de datos de manera eficiente. La capacidad de procesar datos rápidamente y manejar una amplia variedad de tipos de datos asegura que se puedan obtener *insights*, esto se refiere a los datos que nos aportan una rica información para la optimización. Además, el uso de tecnologías como *Hadoop* y *Spark* convergen juntos para aprovechar todas las características que ofrece *Big Data*, proporcionando resultados clave para el análisis de datos en el entorno académico.

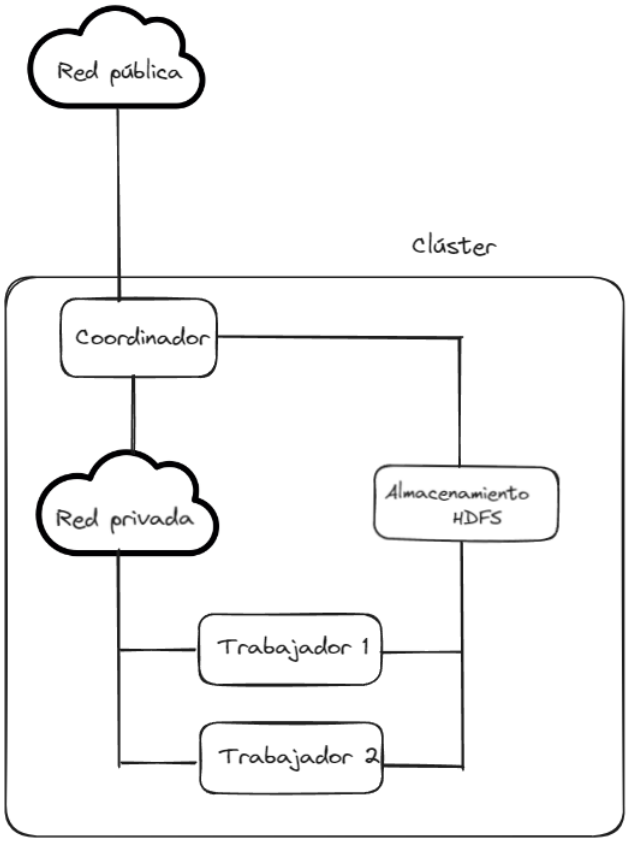

Ilustración 3.5: Clúster

Para que las máquinas puedan comunicarse, se debe añadir, como en 3.6, las IP de las máquinas que conforman el clúster y los nombres que se les quiera dar.

```
127.0.0.1       localhost
172.26.253.21 cero      abaco   abaco.cicei.com
172.26.253.22   liber.cicei.com uno
172.26.253.23   abaci.cicei.com dos


# The following lines are desirable for IPv6 capable hosts
::1     ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
```

Ilustración 3.6: Contenido del archivo /etc/hosts mostrando las asignaciones de direcciones IP y nombres de host del clúster.

Hay que tener en cuenta que las máquinas deben tener habilitado el servicio 'ssh', haber generado sus claves privadas y públicas y que todas posean las claves públicas de las otras máquinas alojadas en el fichero "~/.ssh/authorized_keys".

3.4.1. Instalación Fedora 40:

Para mayor comodidad se ha instalado en un equipo conectado a la red privada en CICEI el sistema operativo Fedora, del cual se ha estado usando para conectarse al clúster a través del comando `ssh` previa solicitud al administrador del mismo de usuario y contraseña.

3.4.2. Instalación Apache Hadoop:

Para descargar e instalar *Apache Hadoop* es necesario acceder a su web https://shorturl.at/tfVhs y descargar "hadoop-3.2.4.tar.gz".

```
tar -xvzf hadoop-3.2.4.tar.gz -C /home
sudo vi ~/.bashrc
- export JAVA_HOME=/usr/java/jdk1.8.0_411-amd64
- export HADOOP_HOME=/home/hadoop-3.2.4
- export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop
- export PATH=$PATH:$HADOOP_HOME/bin:$HADOOP_HOME/sbin
source ~/.bashrc
```

Para probar que se ha instalado correctamente se puede utilizar el comando:

```
hadoop version
```

3.4.3. Configuración Apache Hadoop HDFS:

Una vez instalado *Apache Hadoop* en el sistema, ha de modificarse dos ficheros para configurar el sistema de ficheros distribuido tanto en la máquina coordinadora como en las trabajadoras, denominados:

- ✔ core-site.xml: Este archivo se encuentra en el directorio etc/hadoop dentro del directorio de instalación de *Hadoop*. Se añaden las siguientes propiedades para configurar el nombre del nodo y la URI del sistema de archivos *Hadoop*.

Algoritmo 3.1: $HADOOP_HOME/etc/hadoop/core-site.xml (Todos)

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

✔ hdfs-site.xml: Este archivo también se encuentra en el directorio etc/hadoop. Se añaden las siguientes propiedades para especificar los directorios de almacenamiento de datos para los nodos *NameNode [9]* y *DataNode [10]*.

Algoritmo 3.2: $HADOOP_HOME/etc/hadoop/hdfs-site.xml (Coordinador)

```
<configuration>
    <property>
        <name>dfs.replication</name>
        <value>2</value>
    </property>
    <property>
        <name>dfs.namenode.name.dir</name>
        <value>/home/jorvesan/datos/namenode</value>
    </property>
</configuration>
```

Algoritmo 3.3: $HADOOP_HOME/etc/hadoop/hdfs-site.xml (Trabajadores)

```
<configuration>
    <property>
        <name>dfs.replication</name>
        <value>2</value>
    </property>
    <property>
        <name>dfs.namenode.name.dir</name>
        <value>/home/jorvesan/datos/namenode</value>
    </property>
    <property>
        <name>dfs.datanode.data.dir</name>
        <value>/home/jorvesan/datos/datanode</value>
    </property>
    <property>
        <name>dfs.datanode.hostname</name>
        <value>"nombre del host trabajador"</value>
    </property>
</configuration>
```

✔ workers.xml: Son los nodos de datos en el clúster. Estos nodos son responsables del almacenamiento y procesamiento de los datos.

```
jorvesan@cero:/usr/local/hadoop/etc/hadoop$ cat workers
liber.cicei.com
abaci.cicei.com
```

Ilustración 3.7: Nombre de los *host* de las máquinas trabajadoras. $HADOOP_HOME/etc/hadoop/workers

Se deberá crear un directorio, en este caso, en el directorio raíz de nombre 'datos' que deberá tener los permisos de lectura y escritura para el usuario, incluido su contenido. Por ello, y tal describe el código, también se debe crear tanto para el *NameNode* 3.8 como el *DataNode* 3.9.

```
jorvesan@cero:~/datos$ ll
total 12
drwxrwxr-x  3 jorvesan jorvesan 4096 jun  6 16:27 ./
drwxr-xr-x 17 jorvesan jorvesan 4096 jul 12 11:11 ../
drwxrwxr-x  3 jorvesan jorvesan 4096 jul  8 12:07 namenode/
```

Ilustración 3.8: Contenido del directorio datos del *namenode*

```
jorvesan@uno:~/datos$ ll
total 12
drwxrwxr-x 3 jorvesan jorvesan 4096 jun 21 19:43 ./
drwxr-xr-x 8 jorvesan jorvesan 4096 jun 20 12:59 ../
drwx------ 3 jorvesan jorvesan 4096 jul  8 12:08 datanode/
```

Ilustración 3.9: Contenido del directorio datos del *datanode*

Después de realizar estos cambios, se formatea el *NameNode* para inicializar el sistema de archivos HDFS:

```
1 hdfs namenode -format
```

Para iniciar el HDFS, se utiliza el siguiente comando:

```
1 start-dfs.sh
```

Para verificar que se ha iniciado correctamente se puede usar el comando 'jps' para comprobar que se han iniciado devolviendo como salida:

```
jorvesan@cero:~/datos$ jps
1730 NameNode
2019 SecondaryNameNode
386828 Jps
```

Ilustración 3.10: JPS del coordinador

```
jorvesan@uno:~/datos$ jps
2564734 Jps
706199 DataNode
```

Ilustración 3.11: JPS de los trabajadores

Hadoop proporciona una interfaz de usuario en la a través de la web en la dirección: http://abaco.cicei.com:9870.

3.4.4. Instalación Apache Spark:

Para descargar e instalar *Apache Hadoop* es necesario acceder a su web https://shorturl.at/tfVhs y descargar "spark-3.2.0.tar.gz". Se deben incluir las variables de entorno referentes a *Spark* como se hizo con *Hadoop*.

```
tar -xvzf spark-3.2.0.tar.gz -C /home
sudo vi ~/.bashrc
- export SPARK_HOME=/home/spark-3.2.0
- export PATH=$PATH:$HADOOP_HOME/bin:$HADOOP_HOME/sbin:$SPARK_HOME/
   sbin:$SPARK_HOME/bin
- export SPARK_DIST_CLASSPATH=$(hadoop classpath)
source ~/.bashrc
```

```
spark-shell
```

Spark se usa con YARN en *Hadoop* para gestionar eficientemente recursos y escalar aplicaciones de procesamiento de datos. YARN asigna dinámicamente memoria y capacidad de CPU a *Spark* y otras aplicaciones, garantizando un uso óptimo de recursos en clústeres compartidos. Esta integración permite a *Spark* ejecutar análisis distribuidos escalables, aprovechando la capacidad de almacenamiento de HDFS.

Por lo que se debe modificar los ficheros de configuración para que se ejecute sobre YARN.

✔ mapred-site.xml: Configura parámetros indispensables para el *framework MapReduce*, como la gestión de recursos y la optimización de tareas distribuidas.

Algoritmo 3.4: $HADOOP_HOME/etc/hadoop/mapreduce-site.xml (Todos)

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

Para poder permitir a *Spark* que distribuya automáticamente los recursos en función a la cantidad de trabajo requerida.

✔ yarn-site.xml: Configura y ajusta el *ResourceManager [11]* y *NodeManagers [12]* en un clúster YARN. Contiene configuraciones relacionadas con la asignación de recursos.

Algoritmo 3.5: $HADOOP_HOME/etc/hadoop/yarn-site.xml (Coordinador)

```
<configuration>
    <property>
        <name>yarn.resourcemanager.hostname</name>
        <value>abaco.cicei.com</value>
    </property>
    <property>
```

```
        <name>yarn.nodemanager.aux-services</name>
        <value>mapreduce_shuffle</value>
    </property>
    <property>
        <name>yarn.nodemanager.aux-services.mapreduce_shuffle.
           class</name>
        <value>org.apache.hadoop.mapred.ShuffleHandler</value>
    </property>
    <property>
        <name>yarn.scheduler.maximum-allocation-mb</name>
        <value>1900000</value>
    </property>
    <property>
        <name>yarn.scheduler.maximum-allocation-vcores</name>
        <value>256</value>
    </property>
    <property>
        <name>yarn.nodemanager.resource.memory-mb</name>
        <value>1950000</value>
    </property>
    <property>
        <name>yarn.application.classpath</name>
        <value>
            /usr/local/hadoop/etc/hadoop,
            /usr/local/hadoop/share/hadoop/common/*,
            /usr/local/hadoop/share/hadoop/hdfs/*,
            /usr/local/hadoop/share/hadoop/common/lib/*,
            /usr/local/hadoop/share/hadoop/hdfs/lib/*, /usr/
               local/hadoop/share/hadoop/mapreduce/*,
            /usr/local/hadoop/share/hadoop/mapreduce/lib/*,
            /usr/local/hadoop/share/hadoop/yarn/*,
            /usr/local/hadoop/share/hadoop/yarn/lib/*
        </value>
    </property>
</configuration>
```

Algoritmo 3.6: $HADOOP_HOME/etc/hadoop/yarn-site.xml (Trabajadores)

```
<configuration>
    <property>
        <name>yarn.resourcemanager.hostname</name>
        <value>abaco.cicei.com</value>
    </property>
    <property>
        <name>yarn.nodemanager.aux-services</name>
        <value>spark_shuffle</value>
    </property>
    <property>
```

```
11          <name>yarn.nodemanager.aux-services.spark_shuffle.class<
               /name> <value>org.apache.spark.network. yarn.
               YarnShuffleService</value>
12      </property>
13      <property>
14          <name>yarn.nodemanager.aux-services.spark_shuffle.
               classpath</name>
15          <value>/usr/local/spark/yarn/*</value>
16      </property>
17      <property>
18          <name>yarn.application.classpath</name>
19          <value>
20              /usr/local/hadoop/etc/hadoop,
21              /usr/local/hadoop/share/hadoop/common/*,
22              /usr/local/hadoop/share/hadoop/common/lib/*,
23              /usr/local/hadoop/share/hadoop/hdfs/*,
24              /usr/local/hadoop/share/hadoop/hdfs/lib/*, /usr/
                   local/hadoop/share/hadoop/mapreduce/*,
25              /usr/local/hadoop/share/hadoop/mapreduce/lib/*,
26              /usr/local/hadoop/share/hadoop/yarn/*,
27              /usr/local/hadoop/share/hadoop/yarn/lib/*,
28              /usr/local/spark/jars
29          </value>
30      </property>
31      <property>
32          <name>yarn.nodemanager.resource.cpu-vcores</name>
33          <value>256</value>
34      </property>
35      <property>
36          <name>yarn.nodemanager.resource.memory-mb</name>
37          <value>1950000</value>
38      </property>
39      <property>
40          <name>yarn.scheduler.maximum-allocation-mb</name>
41          <value>1900000</value>
42      </property>
43  </configuration>
```

✔ slaves.xml: Son los nodos de ejecución en el clúster. Estos nodos son responsables de ejecutar las tareas de procesamiento de datos.

```
jorvesan@cero:/usr/local/spark/conf$ cat slaves
liber.cicei.com
abaci.cicei.com
```

Ilustración 3.12: Nombre de los hosts de las máquinas trabajadoras.$SPARK_HOME/conf/slaves

De haber realizado estos pasos, se puede arrancar el servicio YARN iniciando el *ResourceManager* y el *NodeManager* con el comando:

```
1 start-yarn.sh
```

Para verificar que se ha iniciado correctamente podremos usar el comando '`jps`' para comprobar que se han iniciado devolviendo como salida:

```
jorvesan@cero:~/datos$ jps
1730 NameNode
2019 SecondaryNameNode
2243 ResourceManager
386828 Jps
303086 SparkSubmit
```

Ilustración 3.13: JPS del coordinador

```
jorvesan@uno:~/datos$ jps
2080491 ApplicationMaster
2564734 Jps
706199 DataNode
706743 NodeManager
```

Ilustración 3.14: JPS de los trabajadores

Spark proporciona una interfaz de usuario en la a través de la web en la dirección: http://abaco.cicei.com:8088.

Con todo esto ya está el clúster totalmente configurado para poder procesar los datos de forma paralela con todos los recursos disponibles.

3.5. Obtención de datos

En esta sección, se describe el proceso de obtención de datos. Para obtener datos suficientes para poder realizar este proyecto de *BigData* ha sido necesario descargarse de *Crossref* los metadatos de los DOI publicados hasta 2021 https://shorturl.at/ZD54D, que han sido procesados por un script 3.7 para obtener los metadatos más relevantes como: doi, título, autores, resumen y fecha de creación del documento en *Crossref*. Es necesario haber guardado previamente esos metadatos en un directorio para luego crear otro donde se guarden los metadatos procesados.

Crossref es el mayor registro de identificadores de objetos digitales (*Digital Object Identifiers*) (DOI) y metadatos para la comunidad de investigación académica. A diferencia de otras agencias DOI, ellos proveen a todos los interesados en investigaciones en todo el mundo. Facilitan un promedio de 1.1 mil millones de resoluciones de DOI cada mes. Y sus API reciben miles de millones de consultas de sus metadatos cada mes. Para que eso sea posible, *Crossref* facilita la interoperabilidad entre diferentes sistemas de gestión de información y bases de datos académicas, mejorando la eficiencia en la gestión de referencias bibliográficas y citaciones.

Los miembros registran cualquier tipo de estudio como pueden ser artículos, libros, informes de datos, software, etc. Se almacenan metadatos asociados a los DOI, como títulos, autores, fechas de publicación, resúmenes y enlaces a contenido adicional.

Por todo lo anterior, se ha decidido usar esta fuente datos sobre otras, ya que garantiza que los documentos académicos sean accesibles en el tiempo, independientemente de cambios en la ubicación del contenido, ayuda a mejorar la visibilidad y el descubrimiento de la investigación académica, fomenta la colaboración entre editores, investigadores y otras partes interesadas en el ámbito académico a nivel mundial por lo que resulta en una comunidad viva y una continua fuente de información.

Algoritmo 3.7: Procesamiento de metadatos de artículos académicos hasta 2021

```
for filename in os.listdir(directorio_entrada):
    if filename.endswith('.json'):
        nombre_sin_extension = os.path.splitext(filename)[0]
        filepath = os.path.join(directorio_entrada, filename)
        with open(filepath, 'r', encoding='utf-8') as file:
            data = json.load(file)
            datos_procesados = []
            for item in data['items']:
                doi = item.get('DOI', '')
                title_list = item.get('title', [])
                title = title_list[0] if title_list else ''
                authors = ', '.join([f"{author.get('given', '')} {
                    author.get('family', '')}" for author in item.get
                    ('author', [])])
                abstract_clean = extract_paragraphs(item.get('
                    abstract', ''))
```

```
                fecha_creacion_completa = item['created']['date-time'] if 'created' in item else ''
                fecha_creacion = obtener_fecha_yyyymmdd(fecha_creacion_completa)

                datos_procesados.append({
                    'doi': doi,
                    'title': title,
                    'authors': authors,
                    'abstract': abstract_clean,
                    'creation_date': fecha_creacion
                })

            df = pd.DataFrame(datos_procesados)

            output_filepath = os.path.join(directorio_salida, f'{nombre_sin_extension}.parquet')
            df.to_parquet(output_filepath, index=False)

            print(f'Datos procesados de {filename} guardados en {output_filepath}')
```

Tras haber procesado esos metadatos se deben guardar en el HDFS 3.8.

Algoritmo 3.8: Almacenado de datos de hasta 2021 en el HDFS

```
hdfs dfs -put datosProcesados2021/* /datos/input_data
```

También se ha desarrollado otro script para descargar, procesar y almacenar en el HDFS los datos subidos a la *web* de *Crossref* desde 2021 hasta la fecha actual 3.9. Estos datos se suben directamente al HDFS.

Algoritmo 3.9: Descarga y subida de metadatos de artículos académico desde 2021 hasta día actual

```
def descargar_y_procesar_datos():
    fecha_desde = datetime(2021, 7, 1).date()
    fecha_hasta = datetime.today().date()
    cursor = '*'

    while cursor is not None:
        nuevos_registros = []

        resultados = consulta_crossref(fecha_desde, fecha_hasta, cursor=cursor)

        if not resultados:
            break

```

```
        if 'message' in resultados and 'next-cursor' in resultados['message']:
            cursor = resultados['message']['next-cursor']
        else:
            cursor = None

        if 'message' in resultados and 'items' in resultados['message']:
            nuevos_registros = resultados['message']['items']

        if nuevos_registros:
            datos_procesados = []

            for item in nuevos_registros:
                doi = item['DOI']
                titulo = item.get('title', [''])[0]
                autores = [f"{autor.get('given', '')} {autor.get('family', '')}" for autor in item.get('author', [])]
                autores = ', '.join(autores) if autores else ''
                resumen = extract_paragraphs(item.get('abstract', ''))

                fecha_creacion = format_date(item.get('created', ''))

                datos_procesados.append((doi, titulo, autores, resumen, fecha_creacion))

            spark = SparkSession.builder \
                .appName("CrossRef Data Processing") \
                .getOrCreate()

            schema = StructType([
                StructField("doi", StringType(), True),
                StructField("title", StringType(), True),
                StructField("authors", StringType(), True),
                StructField("abstract", StringType(), True),
                StructField("creation_date", StringType(), True)
            ])

            df = spark.createDataFrame(datos_procesados, schema=schema)

            df.show()

            timestamp = datetime.now().strftime('%Y-%m-%d_%H-%M-%S')
            output_parquet_path = f'{HDFS_URL}{HDFS_PATH_PROCESSED}'
```

```
54
55              df.write.mode('append').option("header", "true").parquet
                    (output_parquet_path)
56
57              spark.stop()
58
59              print(f'Se han añadido {len(nuevos_registros)} nuevos
                    registros y se han guardado en el archivo Parquet {
                    output_parquet_path}.')
60          else:
61              print('No hay nuevos registros publicados dentro del
                    rango de fechas especificado.')
```

Y por último, como *Crossref* añade nuevos DOI todos los días, se ha desarrollado otro *script* que se ejecuta diariamente para almacenar en el HDFS los nuevos datos de ese día 3.11. Se utiliza una librería proporcionada por *Crossref* llamada '*habanero*' la cual proporciona una API para gestionar de una mejor manera la descarga de metadatos.

Este *script* se ejecuta gracias a la utilidad de *Linux* 'crontab' que permite a los usuarios programar la ejecución de tareas, en este caso 3.10 se programa este *script* para que se ejecute a las 4:00 am.

Algoritmo 3.10: Fichero crontab

```
1 SHELL=/bin/bash
2 PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin
3
4 0 4 * * * /home/jorvesan/TFTProyecto/script_python_crossref.sh > /
    home/jorvesan/TFTProyecto/error_script_python_crossref.log 2>&1
```

Algoritmo 3.11: Descarga y subida de metadatos de artículos académicos diario

```
1  def descargar_y_procesar_datos():
2      fecha_actual = datetime.now().date()
3      cursor = '*'
4
5      while cursor is not None:
6          nuevos_registros = []
7
8          resultados = consulta_crossref(fecha_actual, cursor=cursor)
9
10         if not resultados:
11             break
12
13         if 'message' in resultados and 'next-cursor' in resultados['
               message']:
14             cursor = resultados['message']['next-cursor']
15         else:
16             cursor = None
```

```
17
18          if 'message' in resultados and 'items' in resultados['
               message']:
19              nuevos_registros = resultados['message']['items']
20
21          if nuevos_registros:
22              datos_procesados = []
23
24              for item in nuevos_registros:
25                  doi = item['DOI']
26                  titulo = item.get('title', [''])[0]
27                  autores = [f"{autor.get('given', '')} {autor.get('
                       family', '')}" for autor in item.get('author',
                       [])]
28                  autores = ', '.join(autores) if autores else ''
29                  resumen = extract_paragraphs(item.get('abstract', ''
                       ))
30
31                  fecha_creacion = format_date(item.get('created', '')
                       )
32
33                  datos_procesados.append((doi, titulo, autores,
                       resumen, fecha_creacion))
34
35              spark = SparkSession.builder \
36                  .appName("CrossRef Data Processing") \
37                  .getOrCreate()
38
39              schema = StructType([
40                  StructField("doi", StringType(), True),
41                  StructField("title", StringType(), True),
42                  StructField("authors", StringType(), True),
43                  StructField("abstract", StringType(), True),
44                  StructField("creation_date", StringType(), True)
45              ])
46
47              df = spark.createDataFrame(datos_procesados, schema=
                   schema)
48
49              df.show()
50
51              timestamp = datetime.now().strftime('%Y-%m-%d_%H-%M-%S')
52              output_parquet_path = f'{HDFS_URL}{HDFS_PATH_PROCESSED}'
53
54              df.write.mode('append').option("header", "true").parquet
                   (output_parquet_path)
55
56              spark.stop()
```

```
57
58              print(f'Se han añadido {len(nuevos_registros)} nuevos
                    registros y se han guardado en el archivo Parquet {
                    output_parquet_path}.')
59          else:
60              print('No hay nuevos registros publicados hoy.')
```

Todos estos metadatos se almacenan en el directorio `/datos/input_data` del HDFS en formato *Parquet*, el cual, es un formato ideal para el procesamiento de datos en paralelo y de forma masiva, ya que proporciona unos metadatos útiles para su posterior procesamiento.

Todo esto proporciona a nuestro proyecto de una ingesta robusta y cuantiosa de información para posteriormente ser visualizada.

3.6. Procesamiento de datos

Esta parte del proyecto está desarrollado en *Scala*, este lenguaje es bastante utilizado en sistemas distribuidos y dado que el programa tiene un tamaño considerable en parte por las librerías, se utiliza SBT que genera un fichero llamado *build.sbt* 3.15 en el que se incluyen esas librerías y se especifica que se genere un fichero .jar que se utiliza para arrancar y montar la aplicación de *Spark*.

```
jorvesan@cero:~/TFTProyecto/CrossrefTFT$ cat build.sbt
import Dependencies._
import sbtassembly.AssemblyPlugin.autoImport._

ThisBuild / scalaVersion     := "2.12.15"
ThisBuild / version          := "1.0.0-SNAPSHOT"
ThisBuild / organization     := "ulpgc.jorge.vega109"
ThisBuild / organizationName := "Universidad de Las Palmas de Gran Canaria"

lazy val root = (project in file("."))
  .settings(
    name := "CrossrefDataProcessing",
    libraryDependencies ++= Seq(
      "org.apache.spark" %% "spark-core" % "3.2.4",
      "org.apache.spark" %% "spark-streaming" % "3.2.4",
      "org.apache.spark" %% "spark-sql" % "3.2.4",
      "com.johnsnowlabs.nlp" %% "spark-nlp" % "5.3.3",
      "org.apache.hadoop" % "hadoop-common" % "3.2.4",
      "org.apache.hadoop" % "hadoop-hdfs" % "3.2.4",
    ),
    assemblyMergeStrategy :=
    {
      case PathList("META-INF", xs @ _*) => MergeStrategy.discard
      case x => MergeStrategy.first
    },
    assemblyJarName := "CrossrefDataProcessing.jar",
    mainClass := Some("CrossrefDataProcessing"),
  )
```

Ilustración 3.15: Fichero build.sbt

El programa se centra en diferenciar dos tipos de procesamiento de datos a elección del usuario que se reciben por el programa a través de un *socket*. En caso de recibir un '0', el usuario ha de acompañar ese dato especificando palabra a analizar, fecha de inicio y fecha de fin para que el programa inicie la búsqueda sobre todos los datos de los ficheros en el HDFS, al encontrar esas coincidencias realiza el análisis de sentimiento y en caso de recibir un '1', el programa realiza una búsqueda de las 100 palabras más frecuentes en el HDFS 3.16.

```
command match {
  case "1" =>
    println("Ejecutando conteo de las 100 palabras más frecuentes...")
    val parquetDF = spark.read.parquet(hdfsInputDir)
    val clientRequest = ClientRequest(1, "", "", "")
    lastClientRequest = Some(clientRequest)
    lastClientOutput = Some(output)
    countTop100WordCount(spark, parquetDF, output)
  case "0" =>
    val palabra = input.readLine()
    val fechaInicio = input.readLine()
    val fechaFin = input.readLine()
    val clientRequest = ClientRequest(0, palabra, fechaInicio, fechaFin)
    lastClientRequest = Some(clientRequest)
    lastClientOutput = Some(output)
    println(s"Ejecutando procesamiento de archivos con palabra $palabra entre
    $fechaInicio y $fechaFin")
    val parquetDF = spark.read.parquet(hdfsInputDir)
    processFilesWordDates(spark, broadcastPipeline, parquetDF, clientRequest.palabra,
    clientRequest.fechaInicio, clientRequest.fechaFin, output)
  case _ =>
    println("Comando no reconocido.")
    output.println("Error: Comando no reconocido.")
    output.flush()
}
```

Ilustración 3.16: Modos de procesamiento de datos

3.6.1. Media del análisis de sentimiento

El procesamiento del lenguaje natural (NLP (*Natural Language Processing*)) se utiliza en proyectos para permitir que las máquinas comprendan y generen lenguaje humano, lo cual es esencial para una variedad de aplicaciones. Por ejemplo, en el análisis de sentimientos, el NLP puede evaluar emociones en textos como reseñas y comentarios, ayudando a las empresas a entender la opinión pública. Además, facilita la extracción de información relevante de grandes volúmenes de texto, útil para investigaciones y análisis de datos.

Dentro del NLP se encuentra la técnica de análisis de sentimiento, que busca detectar opiniones o sentimientos en los textos procesados. Esto se realiza mediante la identificación de palabras y frases expresan emociones o juicios, como 'bueno', 'nefasto', 'indispuesto' o 'apoteósico'. Gracias a esta identificación, es posible clasificar esas palabras o frases como positivas, negativas o neutrales e incluso, hay algunos bastante avanzados que pueden identificar la ironía o el sarcasmo.

El análisis de sentimiento se usa en nuestra sociedad de diversas maneras. Las empresas, a través de las redes sociales, lo emplean para averiguar la opinión de sus clientes sobre sus

productos y, en caso necesario, ajustar su estrategia de *marketing*. Además, en el ámbito político, se utiliza para determinar el grado de simpatía hacia un candidato basándose en la opinión de los votantes. También puede ser útil para detectar a tiempo los fallos en el mercado o en la sociedad, permitiendo una respuesta más rápida y detallada para mitigar problemas mayores.

En el archivo build.sbt 3.15, es importante destacar que la librería añadida es fundamental 3.12 para el procesamiento de lenguaje natural.

Algoritmo 3.12: Librería Spark-NLP añadida al fichero build.scala

```
libraryDependencies += "com.johnsnowlabs.nlp" %% "spark-nlp" % "
    5.4.0"
```

Se crea una instancia de *PretrainedPipeline* para el análisis de sentimientos. *PretrainedPipeline* es parte de la librería NLP de *John Snow Labs*, conocida como *Spark NLP*. Esta librería ofrece modelos preentrenados para diversas tareas de procesamiento de lenguaje natural, y en este caso, se está cargando un *pipeline* que analiza el sentimiento de textos en inglés. Se utiliza, posteriormente, el contexto de *Spark*(***spark.sparkContext***) para crear 3.17 una variable *broadcast* de la *pipeline*. Las variables *broadcast* en *Spark* permiten distribuir eficientemente una copia de un objeto a todos los nodos del clúster.

Spark NLP de *John Snow Labs* se ha elegido sobre otras herramientas de procesamiento de lenguaje natural debido a su perfecta integración con *Apache Spark*, lo que permite aprovechar las capacidades de procesamiento en memoria y distribuido de *Spark*. Además, *Spark* NLP ofrece un rendimiento superior y escalabilidad, modelos preentrenados personalizables, y compatibilidad con múltiples lenguajes de programación.

El modelo utilizado en este código es un modelo preentrenado de análisis de sentimientos proporcionado por *Spark* NLP. Este modelo de *deep learning* ha sido entrenado previamente en un gran conjunto de datos y es capaz de identificar, como se comenta anteriormente, sentimientos positivos, negativos y neutros en los textos procesados.

```
val pipeline = PretrainedPipeline("analyze_sentiment", lang = "en")
val broadcastPipeline = spark.sparkContext.broadcast(pipeline)
```

Ilustración 3.17: Creación *PretrainedPipeline* y del *broadcastPipeline*

La función *analyzeSentiment* 3.18 toma una descripción de texto y el *pipeline* preentrenada de análisis de sentimientos, distribuida mediante una variable *broadcast* de *Spark*, para evaluar el sentimiento del texto. Utiliza la *pipeline* para anotar el sentimiento de la descripción y luego mapea el resultado a un valor numérico: ’1’ para positivo, ’-1’ para negativo y ’0’ desconocido.

```scala
def analyzeSentiment(description: String, broadcastPipeline: Broadcast[PretrainedPipeline]): Double = {
  val pipeline = broadcastPipeline.value
  val result = pipeline.annotate(description)
  val sentiment = result("sentiment").headOption.getOrElse("")
  sentiment match {
    case "positive" => 1.0
    case "negative" => -1.0
    case _ => 0.0
  }
}
```

Ilustración 3.18: Método *analyzeSentiment* que atribuye valores a las etiquetas de 'positivo' y 'negativo'

Se define una función UDF 3.19 (*User Defined Function*) llamada *analyzeSentimentUDF* que aplica la función *analyzeSentiment* a descripciones de texto. Luego, agrupa los datos filtrados por la fecha de creación y calcula la media de los sentimientos analizados de los resúmenes, agregando el resultado en una nueva columna llamada media. Finalmente, los resultados agregados se recogen en el controlador.

```scala
val analyzeSentimentUDF = udf((description: String) => analyzeSentiment(description, broadcastPipeline))

val mediaPorDia = filteredData
  .groupBy(col("creation_date"))
  .agg(avg(analyzeSentimentUDF(col("abstract"))).alias("media"))
  .collect()

println("Creando DataFrame con resultados...")
val schema = StructType(Seq(
  StructField("fecha", StringType, nullable = false),
  StructField("media", DoubleType, nullable = false)
))
```

Ilustración 3.19: Creación del *dataframe* de resumen y media

3.6.2. 100 palabras más frecuentes

La función se inicia leyendo los datos almacenados en formato *Parquet*. Una vez que los datos son leídos, se procede a la transformación de los textos en palabras individuales. Este paso involucra varias operaciones como convertir todo el texto a minúsculas para asegurar la uniformidad y dividir los resúmenes en palabras utilizando una expresión regular que reconoce cualquier carácter no alfanumérico como delimitador. 3.20

```
val wordCounts = parquetDF
  .select(explode(split(lower(col("abstract")), "\\W+")).as("word"))
  .groupBy(col("word"))
  .count()
  .orderBy(col("count").desc)
  .limit(100)
  .collect()

val schema = StructType(Seq(
  StructField("word", StringType, nullable = false),
  StructField("count", LongType, nullable = false)
))
```

Ilustración 3.20: Lectura y procesado de datos para el conteo de las 100 palabras más frecuentes

Con las palabras ya aisladas, se agrupan y se cuenta la frecuencia de aparición de cada una. El resultado de este conteo es una lista de palabras junto con su respectiva frecuencia. Una vez que se tiene el conteo de todas las palabras, el siguiente paso es ordenar estas palabras por su frecuencia en orden descendente. De esta lista ordenada, se seleccionan las 100 palabras más frecuentes. Para estructurar los resultados de una manera organizada, se define un esquema y se crea un nuevo *DataFrame* que contiene las 100 palabras más frecuentes junto con sus conteos. 3.21

```
val wordCountsDF = spark.createDataFrame(wordCounts.toList.asJava, schema).toDF("word", "coun

val timestamp = LocalDateTime.now().format(DateTimeFormatter.ofPattern("yyyyMMdd_HHmmss"))
val outputPath = s"$hdfsOutputDir/top100_word_counts_$timestamp"

println(s"Guardando resultados en HDFS: $outputPath")

wordCountsDF.write
  .mode("overwrite")
  .parquet(outputPath)

println("Enviando resultados al cliente...")
```

Ilustración 3.21: Conteo y selección de las 100 palabras más frecuentes

Finalmente, el *DataFrame* resultante se guarda en el sistema de archivos distribuidos de *Hadoop* y simultáneamente, esos mismos datos, se envían al cliente conectado mediante el servidor de *socket*. 3.22

```
wordCounts.foreach { row =>
  output.println(s"${row.getString(0)}, ${row.getLong(1)}")
}
output.flush()
println("Conteo de palabras completado.")
```

Ilustración 3.22: Almacenamiento en el HDFS y transferencia de datos a través del *socket* de las 100 palabras más frecuentes

3.6.3. Procesamiento en tiempo real

Se encarga de procesar nuevos RDD provenientes de un flujo de datos. Si el RDD no está vacío, se procede a leer los datos en formato *Parquet* desde el directorio de entrada en HDFS y a procesar la última solicitud del cliente utilizando la *pipeline* de análisis de sentimientos. En caso de que el RDD esté vacío, se imprime un mensaje indicando que no hay nada que procesar.

```
  fileStream.foreachRDD { rdd =>
    if (!rdd.isEmpty()) {
      println("Procesando nuevo RDD...")
      println("RDD no está vacío. Leyendo datos Parquet...")
withRetry {
val parquetDF = spark.read.parquet(hdfsInputDir)

      lastClientRequest.foreach { request =>
        println(s"Usando última solicitud del cliente para procesar datos automáticamente: $request")
        lastClientOutput.foreach { output =>
          processClientRequest(spark, broadcastPipeline, parquetDF, request, output)
        }
      }
    }
    } else {
      println("RDD está vacío. Nada que procesar.")
    }
  }
```

Ilustración 3.23: Búsqueda de nuevos datos en el HDFS

3.6.4. Ejecución de la aplicación

Después de haber realizado todas las modificaciones y añadido las librerías necesarias, se proporciona un *script* 3.24 llamado *run_processing.sh* el cual ejecuta tres instrucciones propias de SBT.

```
jorvesan@cero:~/TFTProyecto/CrossrefTFT$ cat run_processing.sh
#!/bin/bash

# Limpiar el proyecto con sbt clean
sbt clean

# Compilar el proyecto con sbt compile
sbt compile

# Crear un ensamblado JAR con sbt assembly
sbt assembly
```

Ilustración 3.24: *Script* de limpieza, compilación y ensamblado

- ✔ **`sbt clean`**: Elimina todos los archivos generados en compilaciones previas del proyecto. Es útil para asegurarse de que la próxima compilación comience desde un estado limpio, sin archivos antiguos que puedan causar conflictos.
- ✔ **`sbt compile`**: Compila el código fuente del proyecto. Traduce el código escrito en *Scala* a *bytecode*. Es un paso necesario antes empaquetar la aplicación.
- ✔ **`sbt assembly`**: Empaqueta el proyecto y todas sus dependencias en un único archivo JAR. Este JAR se utiliza para ejecutar la aplicación a través de *Spark*.

Seguidamente, se puede ejecutar el programa de *Scala* a través de *Spark* con la siguiente instrucción 3.13.

Algoritmo 3.13: Ejecución del programa de scala en Spark

```
spark-submit \
  --class CrossrefDataProcessing \
  --master yarn \
  --deploy-mode cluster \
  --conf spark.sql.parquet.filterPushdown=true \
  --conf spark.sql.parquet.cacheMetadata=true \
  --conf spark.driver.memory=8g \
  --conf spark.driver.memoryOverhead=1g \
  --conf spark.dynamicAllocation.enabled=true \
  --conf spark.shuffle.service.enabled=true \
  --conf spark.files.io.connectionTimeout=1000s \
  --conf spark.memory.fraction=0.8 \
  --conf spark.memory.storageFraction=0.2 \
  --conf spark.executor.extraJavaOptions="-XX:+UseG1GC -XX:
     InitiatingHeapOccupancyPercent=35" \
  target/scala-2.12/CrossrefDataProcessing.jar
```

- ✔ **`--class CrossrefDataProcessing`**: Especifica la clase principal de la aplicación que contiene el método `main`.
- ✔ **`--master yarn`**: Indica que la aplicación se ejecutará en un clúster YARN.

- ✔ **`--deploy-mode cluster`**: Define que el modo de despliegue será en el clúster, donde el coordinador se ejecutará en una de las máquinas trabajadoras del clúster.
- ✔ **`--conf spark.sql.parquet.filterPushdown=true`**: Activa el empuje de filtros en la lectura de archivos *Parquet* para mejorar el rendimiento.
- ✔ **`--conf spark.sql.parquet.cacheMetadata=true`**: Habilita la caché de metadatos de *Parquet* para acelerar el acceso a los datos.
- ✔ **`--conf spark.driver.memory=8g`**: Asigna 8 GB de memoria para el *driver* (Coordinador).
- ✔ **`--conf spark.driver.memoryOverhead=1g`**: Reserva 1 GB de memoria adicional para el *driver* para sobrecarga de ejecución.
- ✔ **`--conf spark.dynamicAllocation.enabled=true`**: Activa la asignación dinámica de recursos, permitiendo ajustar automáticamente el número de ejecutores.
- ✔ **`--conf spark.shuffle.service.enabled=true`**: Habilita el servicio de *shuffle* externo, necesario para la asignación dinámica de recursos.
- ✔ **`--conf spark.files.io.connectionTimeout=1000s`**: Configura el tiempo de espera para las conexiones de IO a 1000 segundos.
- ✔ **`--conf spark.memory.fraction=0.8`**: Establece la fracción de la memoria que se utilizará para almacenamiento interno y ejecución a 80 %.
- ✔ **`--conf spark.memory.storageFraction=0.2`**: Establece la fracción de la memoria que se utilizará para almacenamiento de datos a 20 %.
- ✔ **`--conf spark.executor.extraJavaOptions=XX:+UseG1GC -XX:InitiatingHeapOccupancyPerc`** Configura opciones adicionales para los ejecutores, como el uso del recolector de basura G1GC y el umbral de ocupación del *heap* al 35 %.
- ✔ **`target/scala-2.12/CrossrefDataProcessing.jar`**: Especifica el archivo JAR que contiene la aplicación *Spark*.

3.6.5. Interpretación del procesamiento en la IU de Hadoop y Spark

Una vez se haya iniciado el HDFS, *Hadoop* proporciona a través de la url http://abaco.cicei.com:9870 una visión de los recursos disponibles y usados.

La IU que proporciona *Hadoop* 3.25 muestra un resumen del estado y uso del sistema de archivos distribuido *Hadoop* (HDFS).

- ✔ Se informa del número total de archivos y directorios, bloques replicados y objetos del sistema de archivos. En este caso:
  - Se tienen, 498226 archivos y directorios.

- Se tienen, 497223 bloques replicados.
- El total de objetos del sistema de archivos es 995449.

✔ Se detallan las estadísticas de la memoria *heap* y *non-heap usadas* y asignadas:

- Se usa 1.81 GB de 2.8 GB de memoria *heap*.
- La memoria *heap* máxima es 3.88 GB.
- Se usa 96.98 MB de 100.38 MB de memoria *non-heap* asignada.
- La memoria *non-heap* máxima es ilimitada.

✔ Se presenta un resumen de la capacidad y uso del sistema de archivos distribuidos (DFS):

- Capacidad configurada: 13.86 TB.
- Capacidad remota configurada: 0 B.
- DFS usado: 129.91 GB (0.92 %).
- No DFS usado: 1.43 TB.
- DFS restante: 11.6 TB (83.73 %).
- *Pool* de bloques usado: 129.91 GB (0.92 %).

✔ Se muestra el uso de los *DataNodes* en términos de porcentaje mínimo, mediano, máximo y desviación estándar.

✔ Se detalla el estado de los nodos:

- Máquinas vivas: 2 (ninguno descomisionado o en mantenimiento).
- Máquinas muertas: 0.
- Máquinas en descomisión: 0.
- Máquinas en mantenimiento: 0.

✔ Se muestra el número total de fallas en los volúmenes de *DataNodes*, bloques sub-replicados y bloques pendientes de eliminación:

- Total de fallas en los volúmenes de *DataNodes*: 0 (0 B).
- Número de bloques sub-replicados: 37188.
- Número de bloques pendientes de eliminación: 0.

498.226 files and directories, 497.223 blocks (497.223 replicated blocks, 0 erasure coded block groups) = 995.449 total filesystem object(s).

Heap Memory used 1.81 GB of 2.8 GB Heap Memory. Max Heap Memory is 3.88 GB.

Non Heap Memory used 96.98 MB of 100.38 MB Commited Non Heap Memory. Max Non Heap Memory is <unbounded>.

Configured Capacity:	13.86 TB
Configured Remote Capacity:	0 B
DFS Used:	129.91 GB (0.92%)
Non DFS Used:	1.43 TB
DFS Remaining:	11.6 TB (83.73%)
Block Pool Used:	129.91 GB (0.92%)
DataNodes usages% (Min/Median/Max/stdDev):	0.92% / 0.92% / 0.92% / 0.00%
Live Nodes	2 (Decommissioned: 0, In Maintenance: 0)
Dead Nodes	0 (Decommissioned: 0, In Maintenance: 0)
Decommissioning Nodes	0
Entering Maintenance Nodes	0
Total Datanode Volume Failures	0 (0 B)
Number of Under-Replicated Blocks	37188
Number of Blocks Pending Deletion (including replicas)	0
Block Deletion Start Time	Mon Jul 08 12:07:57 +0100 2024
Last Checkpoint Time	Fri Jul 12 10:12:07 +0100 2024
Enabled Erasure Coding Policies	RS-6-3-1024k

Ilustración 3.25: Recursos de almacenamiento y estado de las máquinas en la IU de *Hadoop*

En otra pestaña 3.26 se pueden ver las máquinas trabajadoras (*workers*). En esta vista se detectan correctamente las dos máquinas trabajadoras, proporcionando información detallada sobre cada una. Esta información incluye el identificador de la máquina en la red, un enlace donde se especifican detalles adicionales sobre cada máquina, como su rendimiento, estado y configuración específica e indica cuánto del espacio de almacenamiento del HDFS está siendo utilizado en cada una de las máquinas trabajadoras.

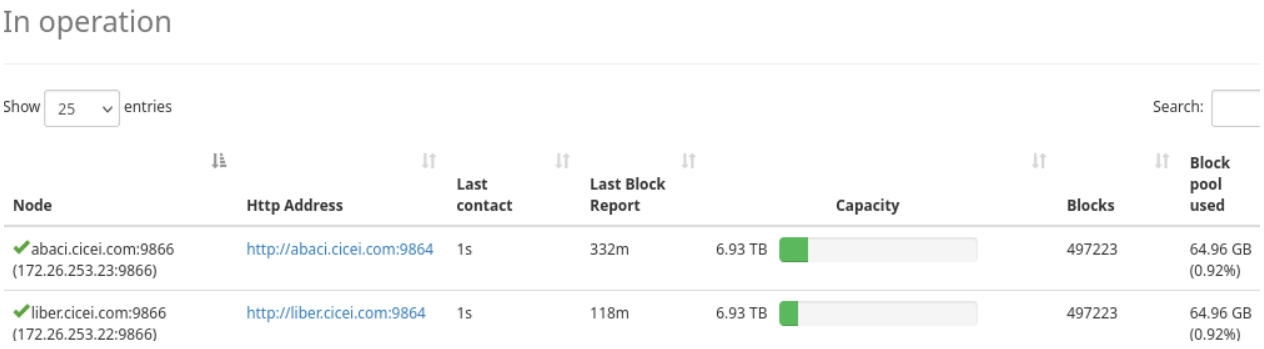

Ilustración 3.26: Información máquinas trabajadoras IU Hadoop

En otra pestaña se puede visualizar 3.27 el cómo está dispuesto el HDFS, con sus directorios y ficheros, en el mismo formato en el que se muestran los sistemas de ficheros en *Linux*. Esta vista es útil para navegar y gestionar los archivos almacenados en el HDFS, permitiendo a los usuarios realizar operaciones como explorar directorios y archivos, identificar y organizar los datos almacenados en el clúster y realizar operaciones típicos de los archivos como mover, copiar o eliminarlos, y verificar su integridad y replicación.

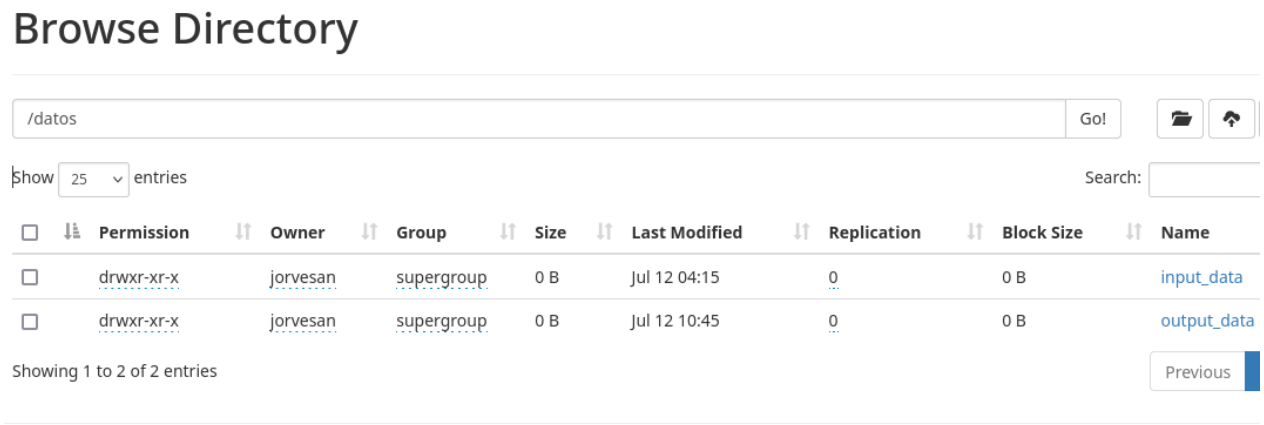

Ilustración 3.27: Visualización del HDFS a través de IU *Hadoop*

Aun así, estos ficheros también se pueden consultar desde la terminal con un comando 3.14 que proporciona Hadoop.

Algoritmo 3.14: Consulta desde terminal de los datos del HDFS

```
hdfs dfs -ls /datos/input_data/*
```

Y una vez ejecutado la aplicación de *Scala* en *Spark* se podrá acceder a la url http://abaco.cicei.com:8088, esto muestra una IU 3.28 donde se detallan datos importantes como:

- ✔ La cantidad total de memoria RAM disponible en el clúster.
- ✔ La cantidad de memoria actualmente utilizada por las aplicaciones en ejecución.

- ✔ La memoria que ha sido reservada para usos específicos, pero que aún no ha sido utilizada.
- ✔ El número total de unidades de procesamiento virtual disponibles en el clúster.
- ✔ El número de VCPUs actualmente en uso por todas las aplicaciones.
- ✔ El número de contenedores (entornos de ejecución aislados para cada tarea) que están actualmente activos.
- ✔ Se muestra un resumen del estado de cada nodo en el clúster, incluyendo si están activos, inactivos o en modo de mantenimiento.

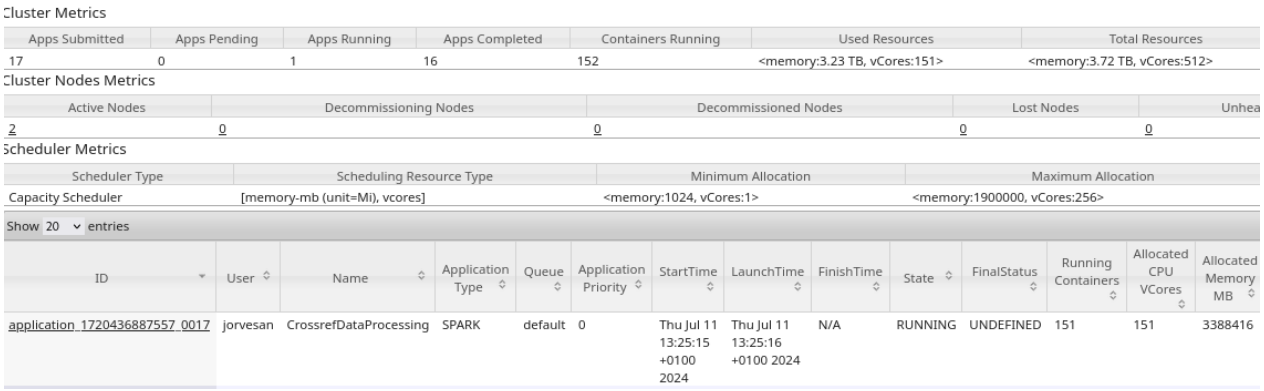

Ilustración 3.28: Visualización de los recursos usados y disponibles para las aplicaciones en IU *Hadoop*

Al pulsar sobre los detalles de la aplicación que se ejecuta en ese momento, se muestra una vista 3.29 detallada de los recursos, como CPU, memoria y almacenamiento de dicha ejecución. Proporciona información sobre los ejecutores (*executors*), que son los componentes que realizan el trabajo de procesamiento de datos y detalla el estado de las tareas individuales que componen la aplicación, incluyendo tareas activas, fallidas y completadas.

	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)
Active(92)	0	13.5 MiB / 1.4 TiB	0.0 B	91	123	0	6043	6166	1.1 h (21 s)
Dead(92)	0	89.6 MiB / 1.4 TiB	0.0 B	92	0	0	12563	12563	1.7 h (32 s)
Total(184)	0	103.1 MiB / 2.8 TiB	0.0 B	183	123	0	18606	18729	2.8 h (53 s)

xecutors

Show 20 entries

Executor ID	Address	Status	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)
2323	dos:40011	Active	0	73.1 KiB / 15.8 GiB	0.0 B	1	2	0	91	93	1.6 min (0.4 s)
2324	dos:34483	Active	0	73.1 KiB / 15.8 GiB	0.0 B	1	1	0	89	90	1.5 min (0.4 s)
2325	uno:46089	Active	0	73.1 KiB / 15.8 GiB	0.0 B	1	2	0	165	167	1.5 min (0.5 s)
2326	dos:33499	Active	0	73.1 KiB / 15.8 GiB	0.0 B	1	1	0	85	86	1.4 min (0.3 s)
2327	uno:41941	Active	0	73.1 KiB / 15.8 GiB	0.0 B	1	2	0	195	197	1.4 min (0.3 s)
2328	dos:37943	Active	0	73.1 KiB / 15.8 GiB	0.0 B	1	2	0	88	90	1.4 min (0.4 s)
2329	dos:40151	Active	0	73.1 KiB / 15.8 GiB	0.0 B	1	1	0	92	93	1.4 min (0.5 s)
2330	dos:45581	Active	0	73.1 KiB / 15.8 GiB	0.0 B	1	2	0	89	91	1.4 min (0.4 s)
2331	dos:46279	Active	0	73.1 KiB / 15.8 GiB	0.0 B	1	1	0	90	91	1.4 min (0.4 s)
2332	dos:33687	Active	0	73.1 KiB / 15.8 GiB	0.0 B	1	2	0	92	94	1.4 min (0.4 s)

Ilustración 3.29: Ejecución de una aplicación en IU de *Spark*

- ✔ En la pestaña *Executors*, se muestra un resumen y detalles de los ejecutores que están activos o en total. En este caso:
  - Se tienen 92 ejecutores activos.
  - En total, se tienen 184 ejecutores.
- ✔ Se detalla la memoria de almacenamiento utilizada, el espacio en disco usado, el número de núcleos (*cores*) asignados, las tareas activas, fallidas y completadas, así como el tiempo de ejecución y el I/O de entrada y salida (*Input* y *Shuffle Read/Write*) para cada ejecutor.
- ✔ Para cada ejecutor activo, se muestra información específica, como el ID del ejecutor, la dirección, el estado, los bloques de RDD, la memoria de almacenamiento, el espacio en disco utilizado, los núcleos asignados, las tareas activas, fallidas y completadas, el tiempo de las tareas, el *input*, el *shuffle read/write* y enlaces a los registros (*logs*) y *dumps* de hilo (*thread dumps*).
- ✔ Se observan las métricas detalladas para cada uno de los ejecutores activos. Por ejemplo, el ejecutor con ID 2323 está ejecutando 2 tareas activas y ha completado 91 tareas con un tiempo total de GC (*garbage collection*) de 1.6 minutos.

Con todo lo expuesto se observa que realmente la ejecución de la aplicación se ha realizado de manera paralela entre todos los recursos del clúster, usando, según la necesidad del YARN de *Spark*, las CPU y espacio de RAM de ambas máquinas trabajadoras.

3.7. Visualización de datos

3.7.1. Instalación e iniciación del servidor web

La visualización de los datos se realiza gracias a un servidor *web* desarrollado con *Django*, para ello ha sido necesario descargarse las librerías de *Django*.

```
pip install django
```

Una vez descargado se inicia el proyecto y la aplicación en la ubicación actual.

```
django-admin startproject servidorWeb
python manage.py startapp visualizador
```

Django genera automáticamente toda la estructura necesaria para iniciarse y proporciona un html base para comprobar que todo se ha instalado correctamente. Se indica cuál va a ser la IP y puerto del que *Django* realice todas las escuchas.

```
python manage.py runserver 172.26.253.21:8000
```

Para configurar el servidor *web* se debe indicar la estructura del mismo mediante la modificación de los ficheros urls, tanto del proyecto *web* 3.15 como de la aplicación 3.16.

Algoritmo 3.15: Configuración urls del proyecto web

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('visualizador.urls')),
]
```

Algoritmo 3.16: Configuración urls de la aplicación

```
from django.urls import path
from . import views

urlpatterns = [
    path('', views.index, name='index'),
    path('process/', views.process_request, name='process_request'),
    path('results/<str:palabra>/<str:fecha_inicio>/<str:fecha_fin>/'
        , views.show_results, name='results'),
    path('word_cloud/', views.show_word_cloud_results, name='
       word_cloud_results'),
    path('polling_endpoint/results/', views.polling_endpoint_results
        , name='polling_endpoint_results'),
```

```
10      path('polling_endpoint/word_cloud/', views.
            polling_endpoint_word_cloud, name='
            polling_endpoint_word_cloud'),
11  ]
```

La manera en la que el servidor *web* obtiene los datos depende de un método 3.30 en el fichero *views.py* el cual se conecta a un *socket*, si ese *socket* está disponible y emite datos en el formato esperado, este los lee y los envía a las páginas html dependiendo de la función seleccionada.

```python
def send_and_receive_data(flag, palabra=None, fecha_inicio=None, fecha_fin=None):
    global last_used_ip_index
    result = []
    connected = False

    for attempt in range(len(ip_addresses)):
        ip = ip_addresses[last_used_ip_index]
        try:
            client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            client_socket.settimeout(5)
            print(f"Attempting to connect to {ip}...")
            client_socket.connect((ip, 9999))
            print("Socket connected to", ip)
            print("Sending data to the socket...")
            client_socket.sendall(f"{flag}\n".encode())

            if flag == 0:
                client_socket.sendall(f"{palabra}\n".encode())
                client_socket.sendall(f"{fecha_inicio}\n".encode())
                client_socket.sendall(f"{fecha_fin}\n".encode())
            elif flag == 1:
                pass
            result = receive_data(client_socket)
            connected = True
            break
        except Exception as e:
            print(f"Error connecting to {ip}: {str(e)}")
            last_used_ip_index = (last_used_ip_index + 1) % len(ip_addresses)

    if not connected:
        print("No se pudo conectar a ninguna de las direcciones IP especificadas.")

    return result

def receive_data(client_socket):
    result = []
    buffer = ''
    try:
        print("Receiving data from the socket...")
        while True:
            data = client_socket.recv(1024)
            if not data:
                break
            buffer += data.decode()
            while '\n' in buffer:
                line, buffer = buffer.split('\n', 1)
                result.append(line.strip())
        print("Data received.")
    except Exception as e:
        print(f"Error al recibir datos del socket: {str(e)}")
    return result
```

Ilustración 3.30: Métodos de conexión al *socket*, selección de funcionalidad y lectura

Para el caso concreto de la actualización de datos en tiempo real, se ha decidido porque el código *JavaScript*, sea el que cada 20 minutos solicite a estos métodos una lectura del *socket* y así obtener los datos.

El servidor *web* está preparado para leer los datos de un *socket* en formato *JSON*. Para comprobar los datos que se reciben correctamente, se imprimen en una lista, agrupándolos, en este caso, por fecha y su media del análisis de sentimiento 3.31.

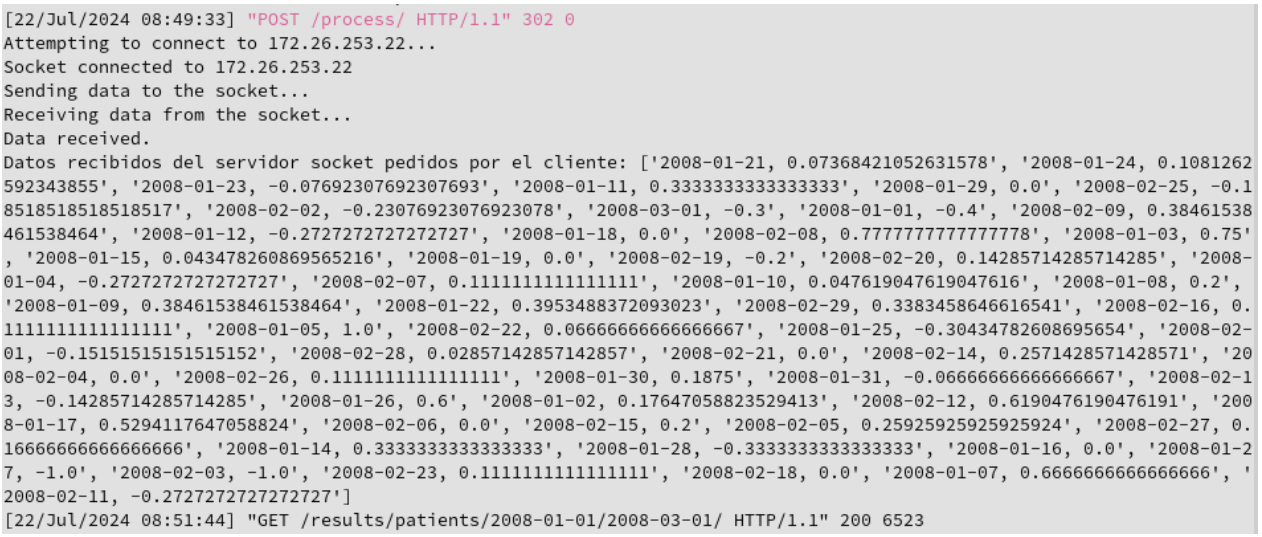

Ilustración 3.31: Datos transferidos por el *socket*

En este ejemplo 3.32, se puede observar en el rectángulo azul un aumento de la frecuencia de palabras debido a la nueva ingesta diaria de datos obtenido de la consola de la página *web*.

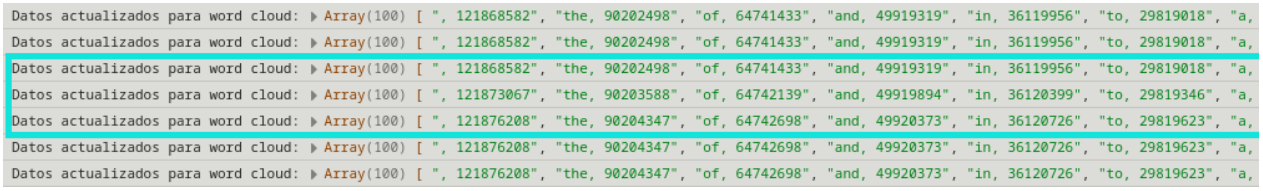

Ilustración 3.32: Visualización de datos que actualizan el gráfico

La primera página que muestra el servidor *web* 3.33 al iniciarse es un formulario con dos botones, donde el formulario es necesario para realizar el análisis de sentimiento y, por otro lado, está el botón de 'Ver *Word Cloud*' el cual mostrará la nube de palabras de las 100 palabras más frecuentes.

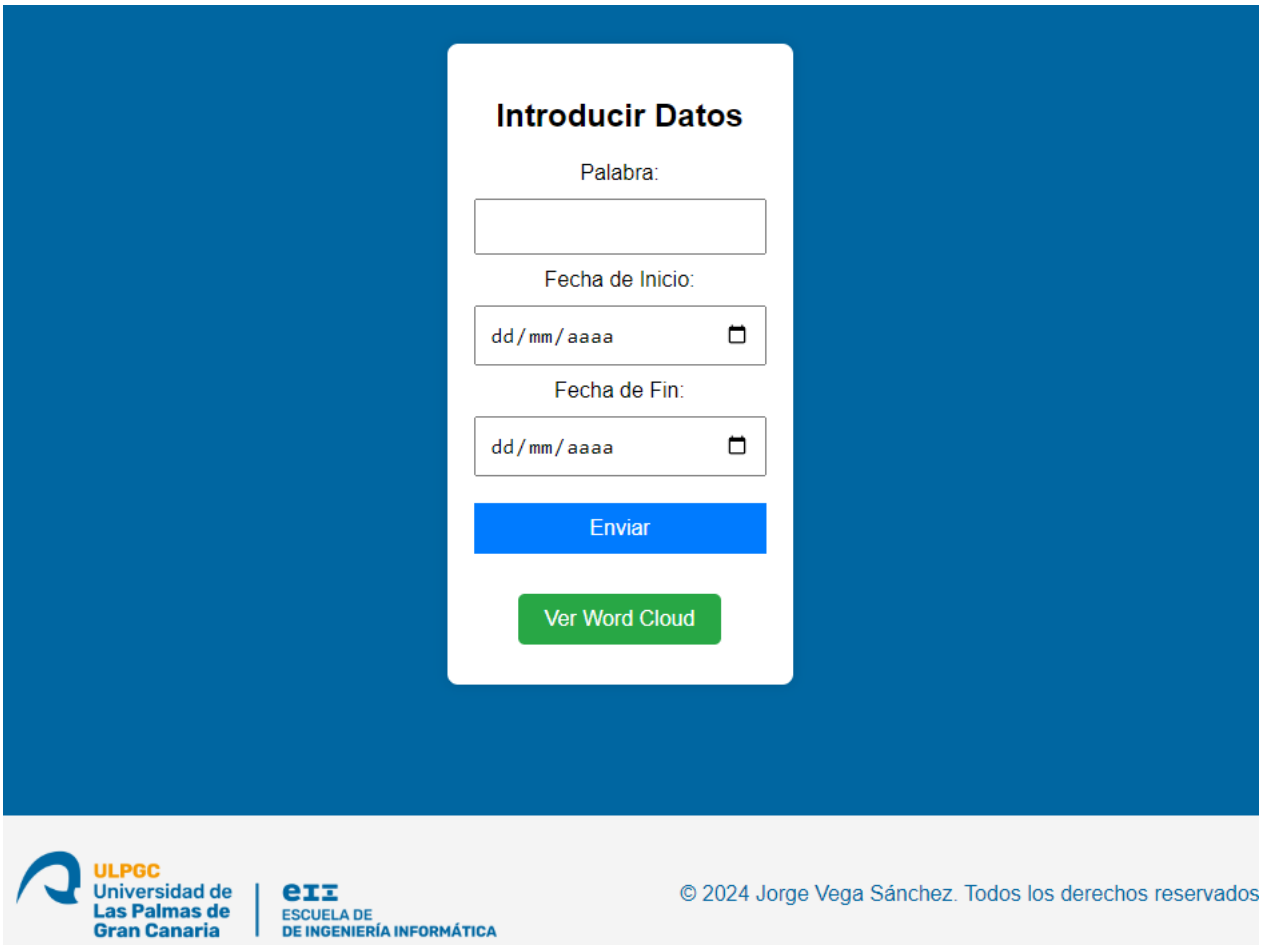

Ilustración 3.33: Página principal del servidor *web*

3.7.2. Media del análisis de sentimiento

El análisis de sentimiento de palabras específicas a lo largo de distintos periodos proporciona una visión preliminar de cómo ciertos temas son percibidos en el entorno académico. Aunque hay limitaciones significativas que impiden extraer conclusiones más detalladas de estos datos.

La precisión del análisis de sentimiento depende de la capacidad del analizador de texto para comprender el contexto y la semántica de los textos. Los analizadores actuales pueden tener limitaciones en la detección de ironías y sarcasmos, lo cual puede llevar a interpretaciones incompletas de los sentimientos.

Mejorar la precisión del analizador de texto mediante el uso de modelos más avanzados de procesamiento de lenguaje natural, como aquellos basados en aprendizaje profundo, seguramente proporcionaría resultados más fiables.

Aun así, se ha decidido usar un enfoque preliminar para obtener una primera aproximación de los sentimientos en la literatura académica, con la intención de identificar patrones

generales. Dando la posibilidad de ampliar este proyecto con nuevas tecnologías que puedan asumir esas faltas sobre equipos más preparados en su desarrollo.

A continuación, se muestran distintos casos de uso con diferentes palabras clave para mostrar sus resultados.

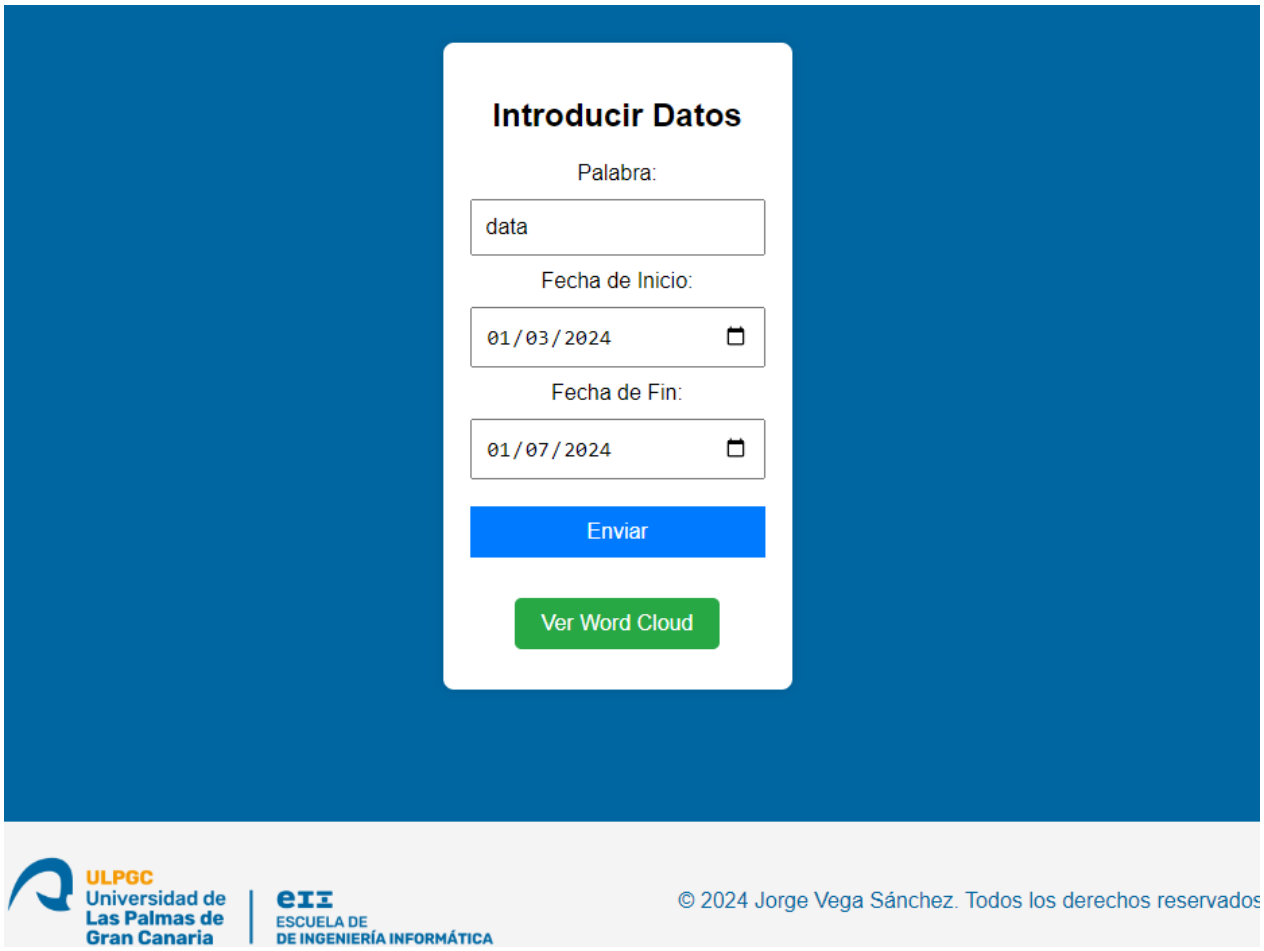

Ilustración 3.34: Caso de uso de la palabra '*data*'

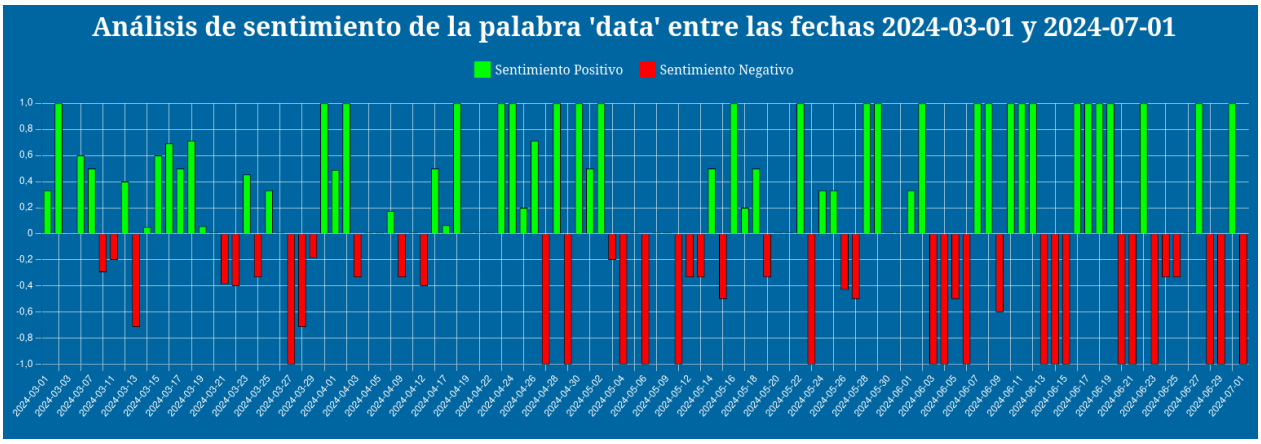

Ilustración 3.35: Análisis de sentimiento palabra '*data*'

Se presenta 3.35 el análisis de sentimiento de la palabra '*data*' en artículos académicos entre las fechas 1 de marzo de 2024 y 1 de julio de 2024.

Se pueden observar picos de sentimiento positivo marcados en verde, alcanzando valores cercanos a 1.0 en varias fechas, lo que indica un alto nivel de comentarios positivos asociados con la palabra '*data*'.

Los segmentos rojos representan sentimientos negativos, con valores que llegan hasta -1.0, indicando momentos en los que la palabra '*data*' se asocia con contextos negativos.

El análisis revela una fluctuación constante entre sentimientos positivos y negativos, lo cual puede reflejar la naturaleza variable de las discusiones en torno a '*data*' en el periodo estudiado.

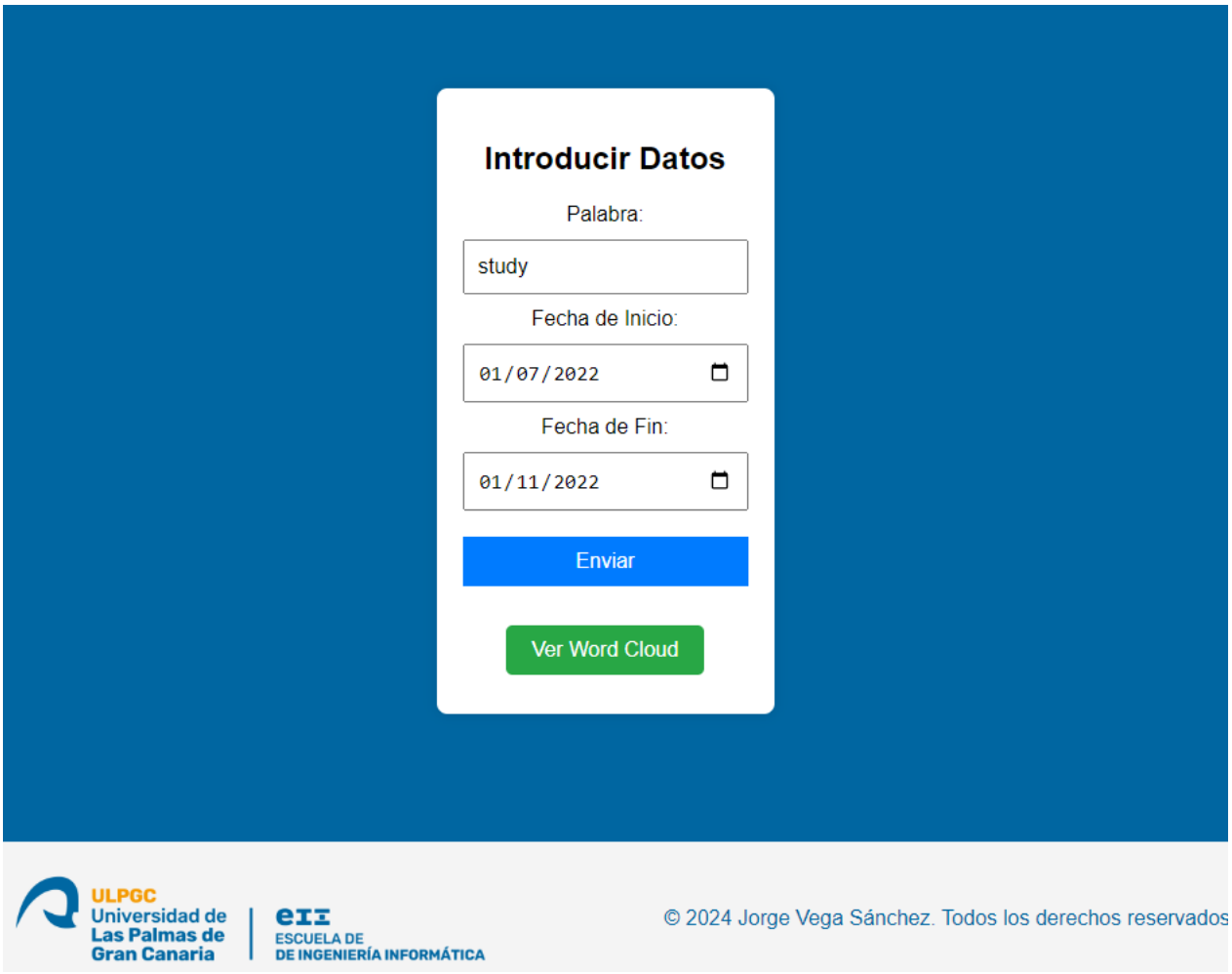

Ilustración 3.36: Caso de uso de la palabra '*study*'

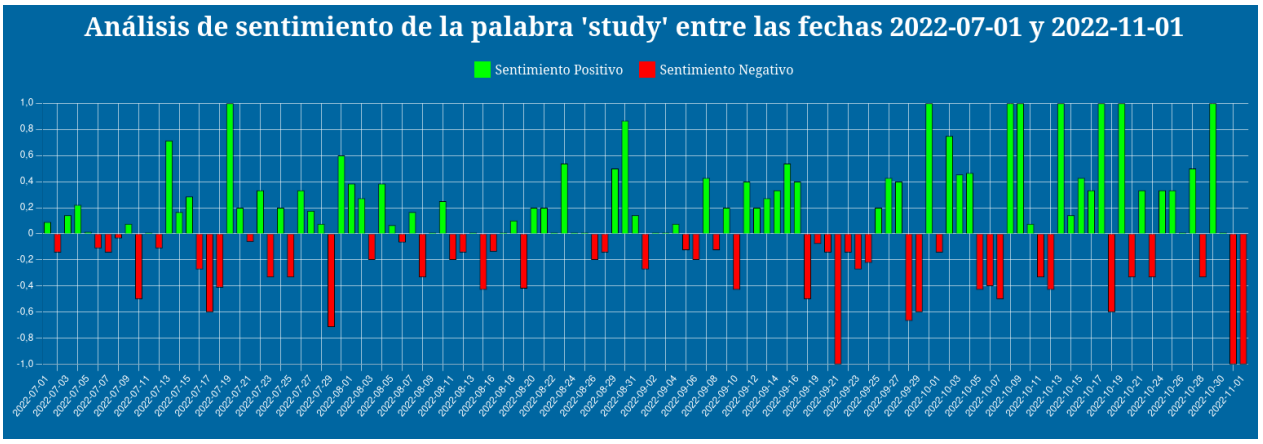

Ilustración 3.37: Análisis de sentimiento palabra '*study*'

Se muestra 3.37 el análisis de sentimiento de la palabra '*study*' en artículos académicos desde el 1 de julio de 2022 hasta el 1 de noviembre de 2022.

Se refleja un patrón mixto de sentimientos positivos y negativos a lo largo del periodo.

Se observan picos significativos de sentimiento positivo alrededor de julio y agosto de 2022, sugiriendo un contexto favorable en esos meses.

Los sentimientos negativos son más prominentes en ciertas fechas de septiembre y octubre, lo que podría estar relacionado con publicaciones críticas.

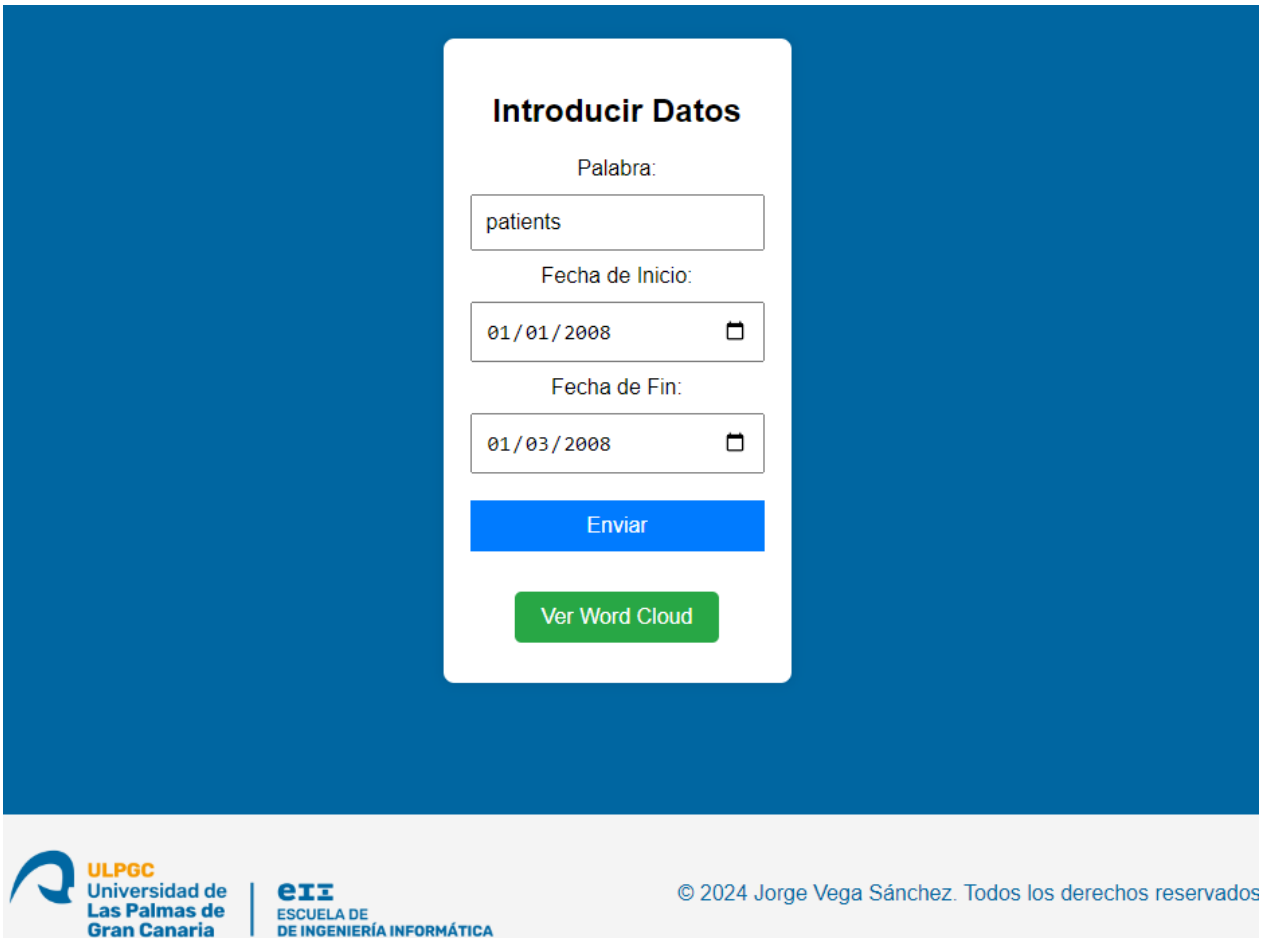

Ilustración 3.38: Caso de uso de la palabra '*patients*'

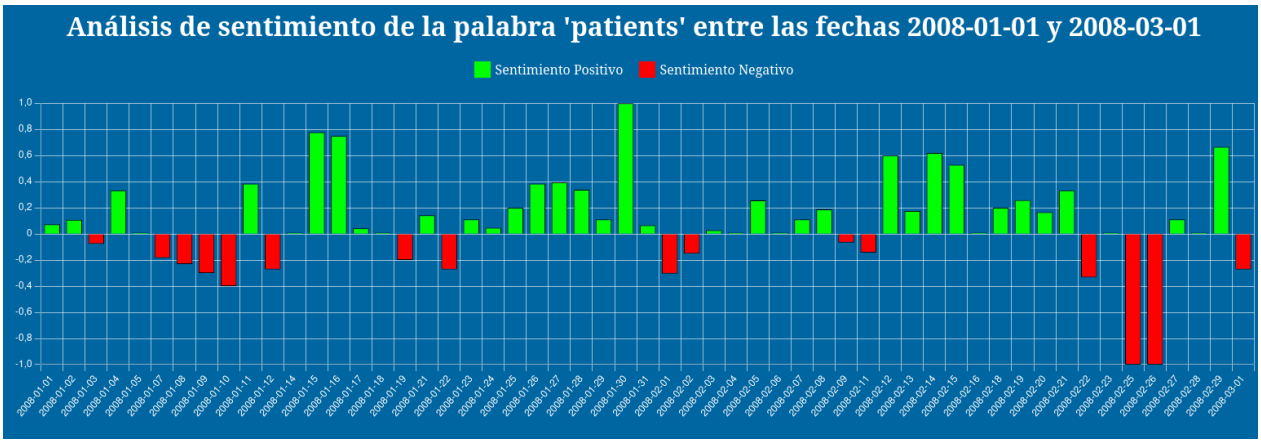

Ilustración 3.39: Análisis de sentimiento palabra '*patients*'

Se visualiza 3.39 el análisis de sentimiento de la palabra '*patients*' en artículos académicos desde el 1 de enero de 2008 hasta el 1 de marzo de 2008.

Se destaca un sentimiento positivo en las primeras semanas de enero, alcanzando picos altos en los días 15 y 16 de enero de 2008.

Los sentimientos negativos son menos frecuentes pero notables alrededor de finales de enero y mediados de febrero.

La variabilidad en los sentimientos puede reflejar diferentes contextos en los que se discute sobre '*patients*', pudiendo asumir desde estudios clínicos positivos hasta críticas en ciertos tratamientos, por ejemplo.

Nota: Aunque se han realizado más análisis de palabras, se han seleccionado estos tres ejemplos como representación de las capacidades del sistema. Estos análisis específicos de las palabras '*data*', '*study*', y '*patients*' son solo una muestra del conjunto más amplio de palabras procesadas en el proyecto. Los resultados presentados son indicativos de la metodología utilizada y de los patrones generales observados.

3.7.3. 100 palabras más frecuentes

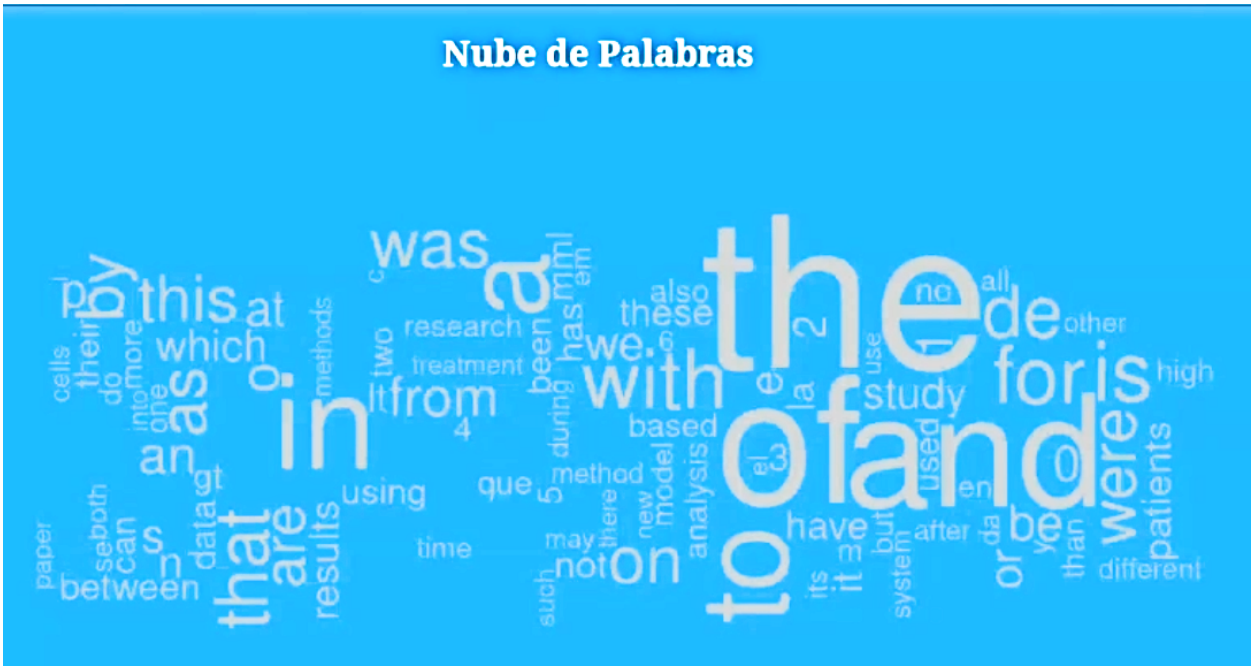

Ilustración 3.40: Visualización 100 palabras más frecuentes

En el conjunto de palabras 3.40 se representa la frecuencia de aparición de las palabras en los resúmenes de los metadatos por el tamaño de esa palabra.

Aunque la visualización de los datos es exhaustiva, hay ciertas limitaciones que impiden la inclusión de más ejemplos en la subsección de las 100 palabras más frecuentes. La principal razón es la insuficiencia de datos procesados en tiempo real. A continuación, se detallan las causas de esta limitación:

- ✔ Frecuencia de actualización de datos:

  Los datos en tiempo real se actualizan con una frecuencia definida, lo que significa que entre las actualizaciones no hay suficientes datos nuevos que alteren significativamente las palabras más frecuentes ya almacenadas en el HDFS. Esto limita la variabilidad y la cantidad de nuevos ejemplos que se pueden generar.

- ✔ Volumen de datos en tiempo real:

  El volumen de datos procesados en tiempo real es menor comparado con los datos almacenados. Esta disparidad significa que cualquier nuevo conjunto de datos tiene una menor probabilidad de influir en las palabras más frecuentes de manera significativa.

Capítulo 4

Conclusiones y trabajo futuro

4.1. Conclusiones

El presente proyecto ha abordado la implementación y optimización de un sistema distribuido para el análisis de sentimientos y procesamiento de grandes volúmenes de datos utilizando herramientas como *Hadoop* y *Spark*. A través de este trabajo, se han logrado varios objetivos clave que demuestran la eficacia y escalabilidad de las tecnologías empleadas.

- ✔ **Eficiencia en el procesamiento de datos**. Uno de los logros más destacados es la mejora en la eficiencia del procesamiento de datos. La utilización de *Apache Spark* en combinación con *Scala* ha permitido manejar grandes conjuntos de datos de manera eficiente y rápida. *Spark*, con su capacidad para procesar datos en memoria, ha demostrado ser una herramienta potente para tareas de análisis de datos intensivas, reduciendo significativamente los tiempos de procesamiento en comparación con enfoques tradicionales.
- ✔ **Análisis de sentimientos con NLP**. La integración de la biblioteca *Spark* NLP de *John Snow Labs* ha sido crucial para realizar análisis de sentimientos de manera precisa. La capacidad de utilizar modelos preentrenados ha facilitado la implementación, permitiendo obtener resultados fiables sin necesidad de entrenar modelos desde cero. Este enfoque ha demostrado ser efectivo para evaluar emociones en textos, proporcionando *insights* valiosos sobre la opinión pública y permitiendo a las empresas tomar decisiones informadas basadas en los sentimientos de los usuarios.
- ✔ **Escalabilidad y flexibilidad**. El uso de tecnologías como *Hadoop* HDFS y YARN ha garantizado que el sistema sea altamente escalable. La capacidad de distribuir y replicar datos en múltiples nodos asegura que el sistema pueda manejar crecientes volúmenes de datos sin comprometer el rendimiento. Además, la flexibilidad de *Spark* para trabajar en diferentes modos de despliegue (local, clúster, etc.) proporciona un espacio adaptable que puede ajustarse a diversas necesidades.
- ✔ **Automatización y facilidad de uso**. La automatización de procesos, como la búsque-

da de nuevos archivos de datos y la ejecución periódica del análisis en tiempo real, ha sido un avance significativo. Esto no solo reduce la intervención manual, sino que también asegura que el sistema se mantenga actualizado y operativo de manera continua. La implementación de scripts automatizados para la compilación y despliegue del código ha simplificado considerablemente el proceso de mantenimiento y actualización del sistema.

- ✔ **Visualización y monitoreo**. La integración de interfaces de usuario proporcionadas por *Hadoop* y *Spark* ha facilitado el monitoreo y la gestión del sistema. Estas interfaces permiten a los usuarios visualizar el estado de los recursos, el progreso de las tareas y el rendimiento general del sistema en tiempo real. Este nivel de transparencia es esencial para la administración eficiente y la rápida resolución de problemas.

4.2. Trabajo futuro

Esto ha demostrado la viabilidad y eficacia de utilizar arquitecturas de *big data* para el análisis de sentimientos masivo. Los objetivos iniciales se han cumplido satisfactoriamente, destacando la capacidad del sistema para manejar grandes volúmenes de datos y proporcionar análisis en tiempo real.

Para trabajos futuros, se sugiere explorar las siguientes extensiones:

- ✔ **Mejora de modelos de *machine learning***: Investigar y aplicar modelos de *machine learning* más avanzados para mejorar la precisión del análisis de sentimientos.
- ✔ **Análisis multilingüe**: Ampliar las capacidades del sistema para soportar múltiples idiomas, utilizando modelos de NLP adaptados para cada idioma, lo que permitiría un análisis de sentimientos más global y diverso.
- ✔ **Optimización de la arquitectura**: Continuar optimizando la arquitectura para mejorar la eficiencia y reducir el tiempo de procesamiento, explorando nuevas tecnologías y técnicas de *bigdata*.

Bibliografía

[1] Bestard, D. (2024a). Primeros pasos para personalizar apache yarn, [en línea]. Disponible en: https://bit.ly/3WkQAkK. Accedido: 2024-07-11.

[2] Bestard, D. (2024b). Primeros pasos para personalizar apache yarn, [en línea]. Disponible en: https://bit.ly/3WkQAkK. Accedido: 2024-07-11.

[3] Center, S. (2024). Sbt, [en línea]. Disponible en: https://www.scala-sbt.org/. [Accedido: 27-mayo-2024].

[4] Code, V. S. (2024). Visual studio code, [en línea]. Disponible en: https://code.visualstudio.com/. Accedido: 2024-05-27.

[5] David Reinsel, John Gantz, J. R. (November 2018). The digitization of the world, [en línea]. Disponible en: https://bit.ly/3zU8Qu1. [Accedido: 26-mayo-2024].

[6] elmundodelosdatos (2024). Dominando apache spark (iii): Explorando rdd (resilient distributed datasets) y su poder en el procesamiento de datos, [en línea]. Disponible en: https://bit.ly/3LC7QNx. Accedido: 2024-07-17.

[7] Fedora (2024). Fedora project, [en línea]. Disponible en: https://getfedora.org/. Accedido: 2024-05-27.

[8] Fernandez, O. (2024a). ¿qué es hdfs? introducción, [en línea]. Disponible en: https://bit.ly/3YzAFCb. Accedido: 2024-07-11.

[9] Fernandez, O. (2024b). ¿qué es hdfs? introducción, [en línea]. Disponible en: https://bit.ly/3YzAFCb. Accedido: 2024-07-11.

[10] Firefox, M. (2024). Mozilla firefox, [en línea]. Disponible en: https://www.mozilla.org/firefox/. Accedido: 2024-05-27.

[11] Foundation, P. S. (2024). Python 3.8.19 documentation, [en línea]. Disponible en: https://docs.python.org/3.8/. [Accedido: 27-mayo-2024].

[12] Git (2024). Git, [en línea]. Disponible en: https://git-scm.com/. [Accedido: 27-mayo-2024].

[13] Hadoop, A. (2022). Hdfs architecture guide, [en línea]. Disponible en: https://bit.ly/3yddG4W. [Accedido: 27-mayo-2024].

[14] Hadoop, A. (2023). Apache hadoop yarn, [en línea]. Disponible en: https://bit.ly/4bYosK0. [Accedido: 27-mayo-2024].

[15] Hadoop, A. (2024). Apache hadoop, [en línea]. Disponible en: https://hadoop.apache.org/. [Accedido: 27-mayo-2024].

[16] Oracle (2024a). Java downloads, [en línea]. Disponible en: https://www.oracle.com/java/technologies/downloads/. [Accedido: 27-mayo-2024].

[17] Oracle (2024b). ¿qué es big data?, [en línea]. Disponible en: https://www.oracle.com/es/big-data/what-is-big-data/. [Accedido: 26-mayo-2024].

[18] PoliScience (2022). ¿qué es el doi?, [en línea]. Disponible en: https://bit.ly/3WAzfWs. [Accedido: 26-mayo-2024].

[19] Rouse, M. (2024). Spark streaming, [en línea]. Disponible en: https://bit.ly/3Sj0Cla. [Accedido: 27-mayo-2024].

[20] Scala (2024). Scala, [en línea]. Disponible en: https://www.scala-lang.org/. [Accedido: 27-mayo-2024].

[21] SmartPanel (2024a). Machine learning, insight y otros términos de big data que deberías conocer, [en línea]. Disponible en: https://bit.ly/3WgttrM. Accedido: 2024-07-24.

[22] SmartPanel (2024b). Machine learning, insight y otros términos de big data que deberías conocer, [en línea]. Disponible en: https://bit.ly/3WgttrM. Accedido: 2024-07-24.

[23] Spark, A. (2024a). Apache spark sql, [en línea]. Disponible en: https://spark.apache.org/sql/. [Accedido: 27-mayo-2024].

[24] Spark, A. (2024b). What is apache spark™?, [en línea]. Disponible en: https://bit.ly/4bYosK0. [Accedido: 27-mayo-2024].

[25] U.N.I.R. (2024). La computación paralela: características, tipos y usos, [en línea]. Disponible en: https://bit.ly/46zfe6d. [Accedido: 26-mayo-2024].

[26] Wikipedia (2024a). Crossref, [en línea]. Disponible en: https://es.wikipedia.org/wiki/Crossref. [Accedido: 26-mayo-2024].

[27] Wikipedia (2024b). Mapreduce, [en línea]. Disponible en: https://es.wikipedia.org/wiki/MapReduce. Accedido: 2024-07-17.

Glosario

Big data El término '*big data*' abarca datos que contienen una mayor variedad y que se presentan en volúmenes crecientes y a una velocidad superior. Bibl.: [17] Pag. 1

Clúster Se refiere a un conjunto de computadoras interconectadas que trabajan juntas como si fueran una sola entidad. Pag. 19

Crossref Es una organización sin fines de lucro que provee infraestructura digital para la comunidad académica y de investigación global. *Crossref* registra y conecta el conocimiento de forma única y persistente a través de metadatos e identificadores abiertos para todos los objetos de investigación, como subvenciones y artículos. Bibl.: [26] Pag. 1

DataNode Se corresponden con los nodos del clúster que almacenan los datos. Se encarga de gestionar el almacenamiento del nodo. Generalmente usan hardware básico con varios discos y una gran capacidad..[8] Pag. 24

DOI DOI (*Digital Object Identifier*) es un identificador único y permanente para las publicaciones electrónicas. El DOI proporciona información sobre la descripción, a través de metadatos, del objeto digital (autor, título, datos de publicación,...) y su localización en internet, utilizando el sistema *handle*. Bibl.:[18] Pag. 4

insights Hace referencia a los datos concretos que nos aportan una información valiosa para la optimización general, un ejemplo de insight puede ser detectar un patrón de comportamiento en el consumidor, habitualmente estos puntos concretos solo pueden ser vistos por expertos en Big Data. Bibl.:[21] Pag. 20

machine learning Es el aprendizaje automático de las máquinas, los sistemas con machine learning aprenden por sí solos mediante la repetición, sin necesidad de que el programador introduzca nuevos datos o códigos. Es una tecnología muy utilizada por ejemplo en el uso de chats bot. Bibl.:[22] Pag. 19

MapReduce es un modelo de programación para dar soporte a la computación paralela sobre grandes colecciones de datos en grupos de computadoras y al commodity computing. El nombre del framework está inspirado en los nombres de dos importantes métodos, macros o funciones en programación funcional: Map y Reduce. Bibl.:[27] Pag. 19

NameNode Es el maestro o nodo principal del sistema. No se encarga de almacenar los datos en sí, sino de gestionar su acceso y almacenar sus metadatos. Se asemeja a una tabla de contenidos, en la que se asignan bloques de datos a *DataNodes*. Bibl.:[9] Pag. 24

NodeManagers Hay uno por nodo del clúster y se encarga de gestionar el uso de los recursos en el nodo. Bibl.:[1] Pag. 26

Procesamiento en tiempo real Es un método que proporciona datos a una velocidad casi instantánea, busca que se introduzcan datos constantemente y se consuman de la misma manera. Pag. 4

ResourceManager Sólo hay uno por clúster y se encarga de gestionar el uso de los recursos en todo el clúster. Bibl.:[2] Pag. 26