

Xelhua

Sistema agnóstico en la nube para la construcción de soluciones de big data basada en el diseño de servicios de ciencia de datos de alta disponibilidad y tolerante a fallos.



Convocatoria FORDECYT 2019-06 CONACyT

Reporte Técnico

Proyecto **41756**

Responsable Técnico:

Dr. José Luis González Compeán

jose Luis.gonzalez@cinvestav.mx

Profesor-Investigador

Cinvestav Tamaulipas



Autorizaciones

Fecha: 8/11/2022	Publicación:	Versión: 2.0
Seguimiento	Nombre completo	Fecha
Elaboró	Julio Noe Hernandez Torres	12/12/2022
Revisó	Juan Armando Barrón Lugo	
Autorizó	Dr. José Luis González Compeán	

Control de cambios

Versión	Fecha	Bitácora
1.0	25/11/2022	Primera versión
2.0	10/12/2022	Segunda versión
3.0	25/1/2023	Tercera versión

Resumen

Xelhua es un sistema de big data agnóstica para la construcción, asistida por el diseño, de servicios de ciencia de datos de alta disponibilidad para la toma de decisiones basada en datos. El sistema Xelhua consta de cuatro componentes principales: (i) un marco de diseño de alto nivel para la selección de herramientas analíticas y de aprendizaje automático, a través de una malla de servicios acoplados basada en pipelines de procesamiento, (ii) un nuevo modelo de procesamiento basado en el modelo de Extracción-Transformación-Carga (ETL, por sus siglas en inglés) recursivo para convertir automáticamente los diseños de pipelines en estructuras de software agnósticas a la infraestructura, (iii) un modelo novedoso de orquestación para gestionar, de forma transparente, la entrega de datos a lo largo de cada etapa ETL de los pipelines de procesamiento utilizados en los sistemas de ciencia de datos, y (iv) un modelo descentralizado de datos para enmascarar de forma transparente la indisponibilidad de algún servicio debido a, por ejemplo, las interrupciones en la nube y la indisponibilidad de las aplicaciones o los datos. En la fase de experimentación, se generaron, a través del sistema Xelhua, servicios de ciencia de datos para el análisis de publicaciones científicas, el análisis de sentimientos a partir de una colección de publicaciones, en redes sociales, relacionadas a la pandemia provocada por el virus de covid-19 y el agrupamiento de críticas de películas. Estos servicios fueron evaluados como casos de estudio revelando la eficacia del sistema Xelhua para el diseño de soluciones en múltiples tipos de problemas de ciencia de datos basado en el diseño. Igualmente, se evaluó el enmascaramiento automático de fallas en el sistema por la falta de disponibilidad de recursos y datos en la nube. Actualmente, el sistema Xelhua es utilizado para la creación de un observatorio nacional del cáncer y de un sistema de ciencia de datos para fusionar conjuntos de datos relacionados al suicidio, salud mental y consumo de drogas, además de un conjunto de datos macroeconómicos para encontrar patrones espaciotemporales.

Índice

RESUMEN	3
ÍNDICE DE FIGURAS	4
PRINCIPIOS DE DISEÑO DEL MODELO DE CONSTRUCCIÓN DE XELHUA PARA CONSTRUIR SISTEMAS DE CIENCIA DE DATOS.....	13
CONSTRUCCIÓN DE UNA MALLA DE SERVICIOS INDEPENDIENTE DE LA PLATAFORMA E INFRAESTRUCTURA DE CÓMPUTO	14
<i>Conectividad</i>	14
<i>Manejabilidad</i>	15
<i>Portabilidad y elasticidad</i>	15
<i>Usabilidad</i>	17
<i>Disponibilidad y Fiabilidad</i>	17
IMPLEMENTACIÓN DEL PROTOTIPO DEL SISTEMA XELHUA	19
<i>Monitoreo e indexación</i>	20
<i>Definición de aplicaciones para estructuras ABox</i>	20
<i>Despliegue de servicios mediante el uso de estructuras ABox</i>	21
EVALUACIÓN EXPERIMENTAL Y RESULTADOS	23
<i>Métricas</i>	23
<i>Infraestructura</i>	24
<i>Evaluación del rendimiento y la disponibilidad del modelo descentralizado de Xelhua</i>	24
CASOS DE ESTUDIO.....	26
CASO DE ESTUDIO 1: PROCESAMIENTO DE DOCUMENTOS	26
CASO DE ESTUDIO 2: ANÁLISIS DE SENTIMIENTOS	28
CASO DE ESTUDIO 3: AGRUPACIÓN DE SINOPSIS DE PELÍCULAS	33
CASO DE ESTUDIO 4: ESTUDIO EXPLORATORIO SOBRE EL CÁNCER	37
MANUAL DE USUARIO DEL SISTEMA XELHUA	41
DESPLIEGUE LOCAL DEL SISTEMA XELHUA.....	41
<i>Configuración local del sistema Xelhua</i>	41
<i>La interfaz gráfica del sistema Xelhua</i>	46
<i>Ejemplo de despliegue de un grafo utilizando la interfaz grafica</i>	47
CONFIGURACIONES AVANZADAS DEL SISTEMA.....	49
<i>Cambiar la ruta de almacenamiento de resultados</i>	49
<i>Limitar recursos de cómputo de los contenedores</i>	Error! Bookmark not defined.
<i>Bloquear registro de nuevos usuarios</i>	50
<i>Administrar base de datos de usuarios</i>	51
<i>API KEY de servicios</i>	52
<i>Añadir nuevas aplicaciones a Xelhua</i>	53
<i>Añadir nuevas aplicaciones a la interfaz gráfica de Xelhua</i>	57
CONCLUSIONES Y ESTATUS DEL SISTEMA XELHUA.....	59
REFERENCIAS	60

Índice de figuras

Figura 1. Representación gráfica de la metodología diseñada	10
Figura 2. Representación gráfica de los componentes del sistema Xelhua	16

Figura 3. fiabilidad y tolerancia a los fallos del sistema Xelhua a través de un modelo descentralizado	19
Figura 4. Representación gráfica de los componentes desarrollados para el prototipo funcional.	20
Figura 5. Ejemplo de un archivo de configuración para una aplicación de una estructura ABox.	21
Figura 6. Representación gráfica del modelo de componentes implementado en una infraestructura.	23
Figura 7. Comparativa del tiempo de procesamiento entre el desempeño de los servicios de aprendizaje profundo al añadir nuevos servicios a la red descentralizada y el desempeño de paso de mensajes al elegir servicios para procesar la carga de trabajo.	25
Figura 8. Comparativa del tiempo de procesamiento de las solicitudes de carga de trabajo y el tiempo de paso de mensajes producido por las tareas de preprocesamiento y aprendizaje profundo.	25
Figura 9. Tiempo de procesamiento del modelo descentralizado del sistema Xelhua al utilizar Paxos, Chord y Bully.	26
Figura 10. Estructura y configuración BDServ	27
Figura 11. Tiempos de respuesta de GROBID para la infraestructura A y C.	28
Figura 12. Representación gráfica del modelo de análisis de sentimientos.	29
Figura 13. Tiempo de respuesta del modelo de análisis de sentimientos.	31
Figura 14. Costo de la malla al incrementar el tiempo de procesamiento de la LSTM.	32
Figura 15. Evaluación del tiempo de intercambio de mensajes cuando se detectan fallos.	33
Figura 16. Representación gráfica de las aplicaciones para el agrupamiento de sinopsis de películas.	33
Figura 17. Tiempo de respuesta obtenido.	36
Figura 18. Costo de la malla al incrementar el tiempo de procesamiento	36
Figura 19. Diseño gráfico del BDServ con validación de índices y resultado obtenido.	37
Figura 20. Representación gráfica del BDServ con índices de validación y resultado obtenido.	39
Figura 21. Tiempos de servicio escalando el número de trabajadores	39
Figura 22. Tiempos de servicio cuando los trabajadores fallan	40
Figura 23. Interfaz gráfica de usuario del sistema Xelhua para el diseño de soluciones de problemas de análisis de datos. En el lado izquierdo se muestra la lista de servicios (cajas). Las cajas pueden arrastrarse al área de diseño (parte derecha).	47

Introducción

Los sistemas de expediente clínico electrónico (SECE) han sido herramientas clave para mejorar los procesos de atención de pacientes. Sin embargo, aún existen áreas de mejora divididas en dos vertientes:

a) el cumplimiento de requisitos de interoperabilidad, eficiencia, seguridad, resistencia a fallas y trazabilidad de transacciones, los cuales no son provistos, en su conjunto, por los SECES;

b) la posibilidad de que tanto los SECE, sus fuentes de datos (personal médico, sensores y dispositivos médicos) y la información producida por los SECE (datos históricos) pueda convertirse en una fuente de datos para algoritmos de analítica con soporte de procesos de toma de decisiones.

Ambas áreas de oportunidad representan un desafío, ya que los datos de salud son de alta sensibilidad y su manejo deben sujetarse a las normas oficiales de protección de datos personales, las cuales no son cubiertas, en su totalidad, por los SECE disponibles en México.

Muyal-llac es un proyecto de carácter multidisciplinario donde colaborarán investigadores y especialistas en telecomunicaciones, ciencias de datos, informática médica y tecnologías de la información para el diseño y desarrollo de esta plataforma para la gestión, aseguramiento, intercambio y preservación de grandes volúmenes de datos en salud (big data) que proveerá servicios para facilitar a los SECE el cumplimiento de normas oficiales y estándares internacionales de operatividad y seguridad. Esta plataforma contempla tres grupos de servicios:

1. Integración, gestión y compartición de información de pacientes, generada a partir de la práctica clínica, así como la interoperabilidad en el intercambio de datos entre los SECE existentes, sin modificarlos.
2. Esquemas de criptografía de siguiente generación, sistemas de almacenamiento digital, sistemas distribuidos de transmisión/descarga de imágenes radiológicas (PACS) y entrega de contenidos basados en flujos de trabajo de publicación/suscripción creados con tecnologías de perímetro, la nube e internet de las cosas, y
3. Servicios de análisis/estadística de datos que podrán acceder automáticamente a los datos publicados por instituciones a través de investigadores autorizados.

Particularmente, en este reporte se presenta el sistema Xelhua como solución al tercer grupo de servicios de la plataforma Muyal-llac con el fin de realizar estudios espaciales temporales con mapas de riesgo basados en bases de datos de egresos de algunas enfermedades crónicas.

Los procesos de toma de decisiones basados en datos (DDDM¹, por sus siglas en inglés) [1] [2] se han convertido en una solución común, entre las organizaciones, para el análisis del contenido de un conjunto de datos [3], transformándolo en información útil para alcanzar objetivos concretos.

Las herramientas de análisis de datos, como el aprendizaje automático y el aprendizaje profundo² [4] [5] [6], se han utilizado ampliamente para apoyar los procesos de DDDM [7] [8] [9]. Los proveedores de servicios en la nube³ ofrecen estas herramientas de análisis de datos (un concepto conocido como *Analytic as a Service* o *AaaS*) [10] [11] para que las organizaciones construyan, operen y consuman sistemas de big data⁴. Un proveedor puede ofrecer un modelo denominado "all-in-one" donde la infraestructura [12] [13], la plataforma [14], las herramientas analíticas de

¹ Data-driven decision-making processes

² También denominado Deep learning en idioma inglés.

³ Proveedores, de aquí en adelante.

⁴ Término en idioma inglés utilizado comúnmente para referirse a grandes volúmenes de datos.

software [15] [16] [8] [9], así como los conjuntos de datos [17] [12] son proporcionados por el proveedor para que las organizaciones puedan crear los servicios de big data que den solución a sus necesidades de procesamiento y análisis de datos.

Los proveedores ofrecen el modelo de negocios AaaS utilizando una técnica denominada orquestación de datos [18] [19] [16]; se encarga de gestionar el intercambio de datos entre los servicios de análisis de datos y la ejecución de aplicaciones utilizando los recursos y la infraestructura de la nube. Esta técnica también considera la gestión de las fuentes de datos (ejemplo, *data lakes* o *data warehouses*) y la entrega de contenidos de información (resultado de los servicios de big data) en cualquier momento a través de diversos dispositivos [20]. Los proveedores ofrecen una amplia variedad de servicios de análisis de datos, así como también múltiples fuentes de datos (estructurados y/o no estructurados), sin embargo, el uso de estos recursos es limitado [21]. Lo anterior presenta un reto en escenarios donde los servicios ofrecidos por los proveedores no contemplan todas las necesidades de procesamiento y análisis de datos que una organización o usuario puedan requerir. Por ejemplo, los repositorios online de datos y los servicios de análisis de datos [22] que siguen los principios FAIR (*Findability, Accessibility, Interoperability, and Reusability*, por sus siglas en inglés) [23] que están actualmente disponibles con el fin de fomentar la reutilización de servicios, compartiendo datos/información [24]. Otro ejemplo es el procesamiento de datos sensibles donde las organizaciones prefieren mantener los datos dentro de la empresa. Los usuarios de servicios en la nube delegan el control sobre los componentes de los servicios de big data (conjuntos de datos, herramientas de análisis de datos, resultados, etc.) a los proveedores. Además, los usuarios deben enfrentarse a un escenario de bloqueo del proveedor [25] [26] en el que deben hacer frente a los efectos secundarios de incidentes como cortes en la nube [27] [28], el acceso no autorizado [29], así como la falta de disponibilidad de los recursos en cualquier momento.

La acumulación de datos [30] en los servicios de la nube pública es otro efecto secundario de los escenarios de bloqueo de proveedores [31]. En un contexto de big data, la acumulación de datos produce una migración masiva de datos entre servicios en la nube cuando las organizaciones intentan cambiar de un proveedor a otro. La migración resulta en un proceso largo y costoso [32] que produce tiempos de inactividad, lo que también podría afectar a la continuidad del negocio [33].

Este reporte técnico presenta el diseño, desarrollo e implementación del sistema Xelhua⁵, una plataforma agnóstica que proporciona soluciones de big data en la nube basada en la construcción guiada por el diseño de servicios de ciencia de datos de alta disponibilidad y tolerante a fallos como herramienta de apoyo a los procesos de DDDM.

El diseño e implementación del sistema Xelhua está enfocado en cuatro componentes principales:

1. Un marco de diseño de alto nivel. Permite seleccionar diferentes herramientas de análisis de datos y de aprendizaje automático a partir

⁵ El nombre Xelhua está basado en la mitología Azteca, haciendo referencia al famoso arquitecto encargado de la construcción de la pirámide de Cholula, Puebla, México.

de una malla de servicios acoplados en pipelines⁶ de procesamiento. El sistema Xelhua implementa un servicio de diseño impulsado por datos (figura 1, desarrollo), permitiendo crear pipelines de big data de alto nivel, lo que produce grafos acíclicos dirigidos (DAG⁷, por sus siglas en inglés).

2. Un nuevo modelo de procesamiento de Extracción-Transformación-Carga (ETL⁸, por sus siglas en inglés) recursivo. Permite al sistema Xelhua convertir automáticamente los diseños de pipelines en estructuras de software independientes a la infraestructura, basándose en el DAG producido en la fase de diseño. ETL, es un proceso de integración de datos que extrae, transforma y carga datos de múltiples fuentes a un almacén de datos o a otro repositorio de datos unificado [34]. Este modelo se encarga de encapsular las aplicaciones analíticas de datos, en imágenes de software genéricas agnósticas a la infraestructura, denominadas ABox, que incluyen las dependencias, bibliotecas y sistemas operativos requeridos por las aplicaciones analíticas para ser ejecutadas en una plataforma de contenedores virtuales (figura 1, desarrollo). Estas imágenes también incluyen interfaces de entrada/salida para interconectar diferentes estructuras ABox, creando los pipelines.
3. Un modelo de orquestación para gestionar de forma transparente la entrega y recuperación de datos a lo largo de cada fase de los pipelines de procesamiento. Este modelo garantiza que el intercambio de datos se orqueste siguiendo la dirección de las aristas del DAG (figura 1, ejecución)).
4. Un modelo descentralizado que enmascara automáticamente los incidentes de indisponibilidad de los servicios para reducir los efectos secundarios del bloqueo del proveedor relacionados con los cortes o la indisponibilidad de los datos y la infraestructura. Este modelo se basa en esquemas de gestión de datos y eventos, implementados en un software que se incrusta en las estructuras ABox. Estos esquemas de gestión de eventos cumplen con los requisitos no funcionales (NFR⁹, por sus siglas en inglés) para garantizar el funcionamiento continuo de los servicios de big data mediante la creación de redes P2P¹⁰ (figura 1, operación).

⁶ El término pipeline es utilizado para referirse a un conjunto de servicios de procesamiento conectados uno tras otro y sin bifurcaciones. Los datos de entrada son transformados por cada servicio, siendo el último servicio quien entregue el resultado final.

⁷ Direct Acyclic Graph

⁸ Extraction-Transformation-Load

⁹ Nonfunctional requirements

¹⁰ Peer to Peer

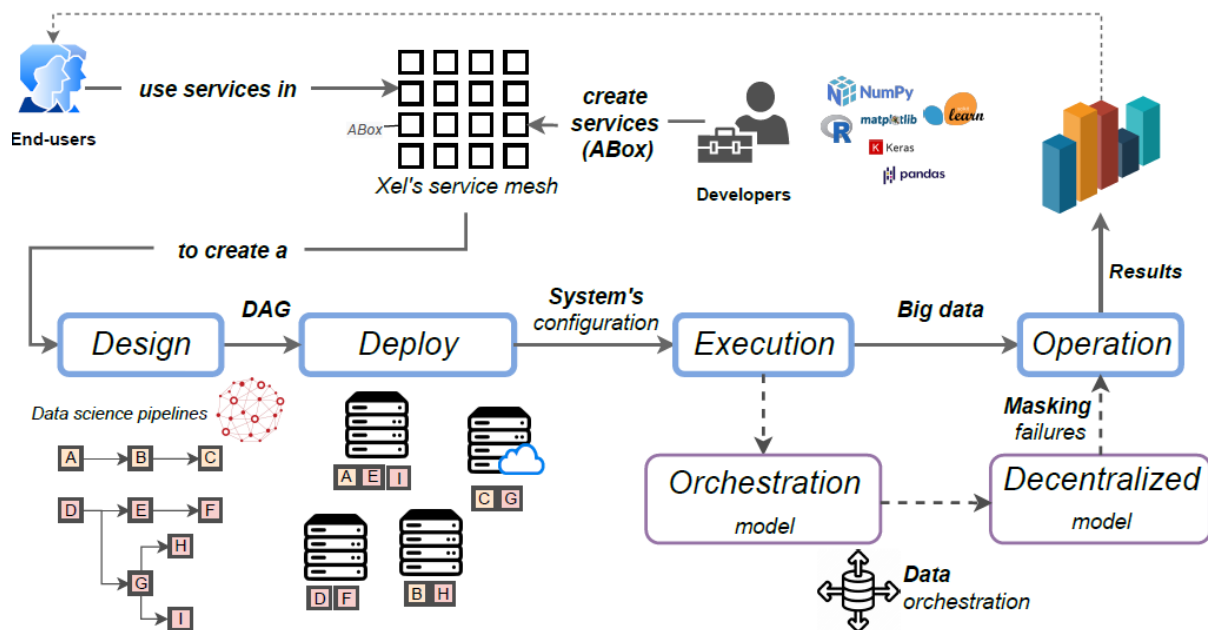


Figura 1. Representación gráfica de la metodología diseñada

El sistema Xelhua permite construir soluciones de alto nivel a través de un esquema impulsado por el diseño, convirtiendo automáticamente los diseños de pipelines en servicios de ciencia de datos agnósticos y de alta disponibilidad en la nube, desplegados en múltiples infraestructuras para hacer frente a los efectos secundarios del bloqueo del proveedor. En tiempo de ejecución, un motor de orquestación crea flujos de datos continuos, mientras que un modelo descentralizado garantiza las operaciones continuas de estos servicios de big data enmascarando los fallos detectados, como la indisponibilidad de aplicaciones y datos.

Para evaluar el sistema Xelhua, se llevó a cabo una evaluación cualitativa entre las funcionalidades del sistema Xelhua y los requisitos no funcionales con las propuestas disponibles en el estado del arte. También se realizó una evaluación experimental utilizando el prototipo del sistema Xelhua, creando servicios analíticos de datos en la nube para DDDM, tales como servicios de aprendizaje profundo de publicaciones científicas, análisis de sentimientos de publicaciones en redes sociales relacionadas a la pandemia provocada por el virus de Covid-19 y el agrupamiento de películas en función de la sinopsis de estas.

La evaluación experimental reveló la eficacia y flexibilidad de las características genéricas y agnósticas del sistema Xelhua para crear múltiples tipos de servicios analíticos de big data fiables en diferentes infraestructuras. La evaluación también reveló la eficacia de los servicios construidos mediante el uso del sistema Xelhua para mantener los servicios de big data disponibles durante fallos como cortes en la nube, así como la indisponibilidad de aplicaciones y datos, produciendo una sobrecarga reducida en el tiempo de respuesta. Esta evaluación cuantitativa demostró que la sobrecarga producida por el sistema Xelhua para mantener el análisis continuo de datos no es significativa e incluso podría eliminarse en algunos escenarios utilizando patrones paralelos.

Trabajo relacionado

En la actualidad existen múltiples herramientas para el análisis de datos, desde tareas de preprocesamiento de datos hasta algoritmos de aprendizaje profundo para la búsqueda de patrones. Algunos ejemplos de estas herramientas son Weka [35], KNIME, RapidMiner, Dislib [9], TensorFlow [8], y MLflow [36]

- Weka es una herramienta de código abierto que proporciona diferentes algoritmos para el preprocesamiento de un conjunto de datos y algoritmos de aprendizaje automático para el análisis de un conjunto de datos. Weka proporciona una interfaz de usuario para seleccionar la fuente de datos de entrada y salida, así como la selección de los algoritmos de limpieza, análisis y de aprendizaje automático. Existen variantes de Weka como Weka4WS [37] y Weka-Parallel [15], que extienden las funcionalidades de Weka para la ejecución de algoritmos de aprendizaje automático como un servicio y la ejecución paralela y distribuida de los algoritmos, respectivamente.
- KNIME y RapidMiner son herramientas de análisis de datos y aprendizaje automático con un enfoque orientado al diseño. Estas herramientas permiten crear flujos de procesamiento de datos sin necesidad de tener conocimientos de programación.
- Dislib es una biblioteca de aprendizaje automático distribuido escrita en Python sobre PyCOMPS¹¹. PyCOMPS ofrece a Dislib una capa transparente para ejecutarse en un entorno distribuido agnóstico a la infraestructura. Dislib integra algoritmos de aprendizaje automático y herramientas para el procesamiento de datos en un entorno distribuido, siendo capaz de procesar datos en paralelo basándose en un enfoque de divide y vencerás, lo que permite procesar grandes conjuntos de datos.
- TensorFlow es una plataforma que opera a gran escala y en un entorno heterogéneo. TensorFlow soporta una gran variedad de aplicaciones, con un enfoque en el entrenamiento y la inferencia en redes neuronales profundas. TensorFlow proporciona una abstracción de programación basada en el flujo de datos que permite a los usuarios desplegar aplicaciones desde clústeres distribuidos hasta estaciones de trabajo locales. TensorFlow hace uso de Apache Beam, un modelo unificado de código abierto para definir pipelines de procesamiento de datos en paralelo y por lotes. En este contexto, las aplicaciones de TensorFlow pueden ejecutarse en entornos distribuidos utilizando *Apache Flink*, *Apache Spark* o *Google Cloud Dataflow*.
- MLflow es una plataforma para gestionar el ciclo de vida de las aplicaciones de aprendizaje automático, incluyendo la experimentación, la reproducibilidad y el despliegue. Esta plataforma permite a los usuarios encapsular aplicaciones, datos y parámetros y dependencias en elementos llamados "proyectos" que son portables y permiten la reproducibilidad de los experimentos en múltiples plataformas. Además, MLflow permite publicar aplicaciones de

¹¹ PyCOMPS es un modelo de programación basado en aplicaciones científicas [53]

aprendizaje automático como servicio, ofreciendo capacidades de procesamiento en tiempo real. Mediante el uso de bibliotecas en múltiples lenguajes de programación (Python, R y Java), MLflow convierte las aplicaciones de aprendizaje automático en proyectos portátiles, que pueden desplegarse en diferentes infraestructuras (por ejemplo, *Kubernetes*, *Google Cloud*, *Azure*, *Amazon*, etc.) y utilizar varias herramientas de aprendizaje automático (por ejemplo, *TensorFlow*, *Keras*, etc.).

En el caso de las herramientas de diseño basadas en datos, el sistema *Xelhua* considera una herramienta de diseño similar a *Weka*, *KNIME* y *RapidMiner*, pero integrada a un esquema de gestión de recursos informáticos sin servidor. Esto significa que, en lugar de estas herramientas, el sistema *Xelhua* no sólo crea sistemas de aprendizaje automático o analíticos, sino que también opera y gestiona esas soluciones en tiempo de ejecución y de forma agnóstica a la infraestructura. Esto permite diseñar pipelines de aplicaciones de análisis de datos, convertir ese diseño en servicios en línea y bajo demanda.

En el caso del procesamiento descentralizado, el sistema *Xelhua* funciona de forma similar a *Dislib* y *TensorFlow*, ya que el objetivo de estas tres soluciones es producir un servicio distribuido y paralelizado que se ejecute en infraestructuras informáticas como servidores, HPC o clusters de ordenadores. Sin embargo, *Dislib* y *TensorFlow* se centran en el procesamiento de alto rendimiento y no tanto en la flexibilidad y fiabilidad de las aplicaciones, ya que los servicios creados por estas soluciones se despliegan, principalmente, en un único entorno, lo que produce escenarios de bloqueo de proveedores. Por su parte, el sistema *Xelhua* genera servicios de big data mediante el uso de patrones de diseño, que realiza y despliega, de forma implícita y transparente, réplicas del servicio en infraestructuras de computación en la nube. Un modelo descentralizado implementado e integrado en los servicios gestiona la distribución de las tareas de análisis/procesamiento a las réplicas de un servicio. En este contexto, el sistema *Xelhua* produce un esquema de gestión de datos y eventos, que no sólo gestiona la distribución y el paralelismo de las tareas, sino que también monitoriza los eventos de fallo para enmascararlos de forma transparente y garantizar un análisis/procesamiento continuo de los datos. Esto significa que *Dislib* y *TensorFlow* pueden utilizar el sistema *Xelhua* para crear servicios de alta disponibilidad, o el sistema *Xelhua* puede utilizar *Dislib* para escenarios HPC y *TensorFlow* para gestionar los requisitos de la GPU.

El sistema *Xelhua* es similar a *MLflow* ya que ambas se centran en la gestión de todo el ciclo de vida de los algoritmos de aprendizaje automático, ofreciendo características para la replicación de experimentos y la publicación de servicios, así como la creación de flujos de trabajo encadenando diferentes aplicaciones a través de sus entradas y salidas. La tabla 1 presenta un resumen comparativo entre el sistema *Xelhua* y los enfoques del estado del arte.

Trabajo	Infraestructura agnóstica	Disponibilidad	Tolerancia a fallos	Análisis de datos	Orientado al diseño	Múltiples infraestructuras	Ciclo de vida de ciencia de datos	Análisis como servicio	Malla de servicio
Weka				X	X				
Weka4WS		X	X	X				X	
KNIME				X	X		X		
RapidMiner				X	X	X	X		
CDAP			X		X	X	X		
MLFlow			X	X		X	X		
DisLib	X		X	X		X		X	
AWS		X	X	X	X		X	X	
Xelhua	X	X	X	X	X	X	X	X	X

Tabla 1. Tabla comparativa de trabajos del estado del arte relacionados con el sistema Xelhua.

Principios de diseño del modelo de construcción de Xelhua para construir sistemas de ciencia de datos

El sistema Xelhua define y está basado en la estructura lógica denominada **Agnostic Box** (ABox, de aquí en adelante), inspirada en la tecnología de contenedores de software, considerados el estándar para el despliegue de microservicios y aplicaciones en la nube [38]. Una estructura ABox representa una aplicación analítica a través de la metainformación descriptiva del servicio proporcionado por dicha aplicación. La estructura ABox, al ser un contenedor de software, proporciona los componentes de software necesarios para la ejecución de la aplicación analítica, así como la configuración necesaria para el correcto despliegue del servicio proporcionado.

Las características principales del sistema Xelhua son las siguientes:

- **Agnosticismo.** El sistema Xelhua proporciona una serie de aplicaciones analíticas que integran servicios de big data, los cuales son independientes de la plataforma de software y la infraestructura de cómputo donde se ejecute¹². Para cumplir con lo anterior, una estructura ABox cuentan con las siguientes características:
 - Conectividad: Una estructura ABox puede interconectarse con otras estructuras ABox a través de interfaces de entrada y/o salida siguiendo un esquema de acoplamiento para crear cadenas de aplicaciones analíticas.
 - Manejabilidad: Un grupo de estructuras ABox pueden ser gestionadas por un administrador descentralizado basado en un **API Gateway** (AG). Una AG provee funciones de acceso a servicios publicados por otras estructuras ABox.
 - Portabilidad: Una estructura ABox es un elemento autónomo con la capacidad de ser ejecutada en cualquier infraestructura con soporte para plataformas de contenedores.
 - Usabilidad: El sistema Xelhua proporciona una interfaz de usuario intuitiva, así como un proceso de implementación y administración guiados.

¹² En el dominio de cómputo en la nube, esta característica se le conoce como *cloud-agnostic*.

- **Disponibilidad.** Una estructura ABox de alta demanda¹³ puede ser replicada o clonada para hacer más eficiente el servicio proporcionado por la estructura ABox, evitando la saturación de este y por tanto el tiempo de espera de cada usuario. Lo anterior provee una mejor experiencia de servicio para usuario final.
- **Confiabilidad.** El sistema Xelhua incluye una estrategia para el control de los fallos, basada en un esquema de consenso descentralizado el cual coordina la gestión y el enmascaramiento de los fallos. De esta forma, si el servicio proporcionado por una estructura ABox presenta un fallo, las peticiones a ese servicio son redireccionadas a una instancia réplica del servicio.

Construcción de una malla de servicios independiente de la plataforma e infraestructura de cómputo

El sistema Xelhua proporciona diferentes aplicaciones analíticas para big data a través de la publicación de diferentes servicios. Los servicios proporcionados están contenidos en unas estructuras ABox, disponibles en un repositorio (R), basada en la tecnología de contenedores de software con las características de conectividad, manejabilidad, portabilidad y usabilidad. A continuación, se detalla cada una de estas características, utilizadas para la construcción de la malla de servicios.

Conectividad

El sistema Xelhua agrupa las estructuras ABox en una malla de servicios, denominada *Big data service* (BDServ). La BDServ está basada en una arquitectura de software para facilitar la comunicación servicio a servicio entre las estructuras ABox e incluso entre otros tipos de servicios o microservicios.

La construcción de un BDServ está basada en el acoplamiento de estructuras ABox, formando un pipeline. El BDServ integra una red de big data P2P (BD-P2P) para la gestión de la comunicación entre los elementos del pipeline a través de un entorno de malla de servicios (SME¹⁴, por sus siglas en inglés) utilizando conexiones punto a punto (P2P).

La BD-P2P representa un DAG de estructuras ABox interconectadas a través de conexiones P2P. Las interconexiones son representadas como tuplas (x,y) donde ambos elementos de la tupla son estructuras ABox, mientras que las aristas (A) del grafo representan las conexiones P2P o *Internal Gateways* (IG) entre estructuras ABox. Las IG representa una puerta trasera para conectar estructuras ABox del mismo BD-P2P o de una estructura ABox perteneciente a otro BD-P2P.

¹³ Una estructura ABox es de alta demanda si muchos usuarios hacen uso del servicio que proporciona.

¹⁴ Service Mesh Environment

Manejabilidad

Para la gestión, ejecución y acoplamiento de aplicaciones, las estructuras ABox dependen de un conjunto de entidades denominadas *Black Box* (BBox). Una entidad BBox representa una abstracción de una aplicación analítica, por ejemplo, el servicio de una aplicación analítica (ABox) de clasificadores que ofrece diferentes opciones de clasificación de acuerdo con el tipo de clasificador deseado: supervisado (BBox_a), no supervisado (BBox_b), por mencionar un ejemplo. En este sentido, una estructura ABox se puede definir como sigue:

$$ABox = (server, \{BBox\}, client)$$

donde:

- BBox, representa un conjunto de entidades BBox, donde cada entidad BBox es una abstracción de una aplicación analítica con parámetros específicos, esto se representa como: $BBox = \{BBox_1, \dots, BBox_2, \dots, BBox_m\}$
- server, representa la fuente de datos de entrada para la estructura ABox
- client, representa la salida de datos de la estructura ABox.

Derivado de la definición anterior, y considerando que una entidad BBox es una abstracción de una estructura ABox, una entidad BBox se define como:

$$BBox = \{server, client\}$$

donde:

- server, representa la fuente de datos de entrada para la entidad BBox.
- client, representa la salida de datos de la entidad BBox

Portabilidad y elasticidad

La portabilidad de una estructura ABox se consigue a través de la metainformación referente a la aplicación analítica contenida en dicha estructura. Los metadatos integran información sobre las dependencias de software relacionadas a la aplicación analítica asociada a la estructura ABox (por ejemplo, sistema operativo, bibliotecas y variables de entorno), dando lugar a una estructura ABox genérica autocontenida.

Una estructura ABox, así como las entidades BBox asociadas, sigue un modelo de Extracción-Transformación-Carga (ETL). ETL, es un proceso de integración de datos que extrae, transforma y carga datos de múltiples fuentes a un almacén de datos o a otro repositorio de datos unificado [39]. El procesamiento realizado por una aplicación analítica, encapsulada en una estructura ABox, sigue las etapas de ETL, donde el primer paso es la recolección de los datos de entrada (E) para la aplicación analítica, después el servicio proporcionado por la estructura ABox (la aplicación analítica) transforma (T) los datos y carga (L) los resultados para almacenarlos y, si es el caso, enviar los resultados a otra aplicación analítica para repetir el proceso hasta alcanzar el resultado final. La figura 2 ilustra el proceso de ETL, donde S representa la etapa de extracción (E), la estructura $Abox_i$ representa la etapa de transformación (T) y C representa la etapa de carga (L).

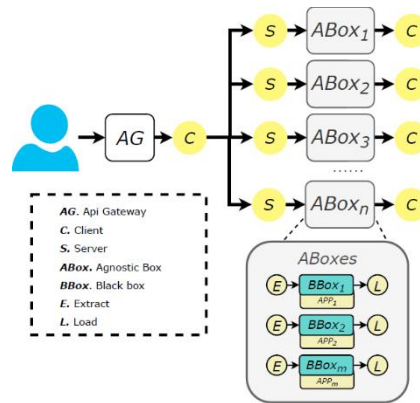


Figura 2. Representación gráfica de los componentes del sistema Xelhua

En la figura 2, la estructura $ABox_n$ ilustra el concepto de una estructura ABox con diferentes entidades BBox, en otras palabras, la estructura $ABox_n$ define diferentes entidades BBox de acuerdo con los parámetros de la aplicación analítica que representa. Las etapas de extracción (E) y carga (L) de cada entidad BBox corresponden con las etapas de extracción (S) y carga (C) de la estructura ABox que la contiene. La entidad BBox para la etapa de transformación (T) dependerá de los parámetros definidos por el usuario con respecto al servicio proporcionados por la estructura ABox.

En este sentido, la etapa de transformación (T) puede estar asociada a un sólo servicio o a un conjunto de servicios definidos por las entidades BBox.

En función de la relación entre los datos de entrada y de salida en una entidad BBox, se definen dos tipos de relaciones:

- Relación paralela (PR). Las entidades BBox aceptan los mismos datos de entrada, realizan una transformación completa de los mismos y producen resultados distintos. Por ejemplo, supongamos que la entidad $BBox_a$ y la entidad $BBox_b$ ofrecen un servicio de clasificación con algoritmos diferentes; tanto la entidad $BBox_a$ como la entidad $BBox_b$ reciben el mismo conjunto de datos como entrada, sin embargo, cada servicio genera resultados distintos. La relación PR se representa de la siguiente manera:

$$PR = (E_j = E_h) \wedge (L_j \neq L_h) \mid (j \neq h) \wedge (BBox_j, BBox_h \in BBoxes_i)$$

- Relación secuencial (SR). Las entidades BBox pueden o no aceptar los mismos datos de entrada, realizan una transformación completa de los mismos y el resultado puede (o no) ser utilizado por otra entidad BBox, o por la misma entidad BBox, con un servicio complementario. Por ejemplo, supongamos que la entidad $BBox_j$ y la entidad $BBox_h$ contienen algoritmos de eliminación de valores atípicos. En este contexto, los datos de entrada son transformados por el proceso de limpieza del servicio proporcionado por la entidad $BBox_j$ y el resultado puede pasar (o no) como dato de entrada para el servicio proporcionado por la entidad $BBox_h$. De esta manera, es posible encadenar diferentes entidades BBox, creando pipelines de procesamiento dentro de una estructura ABox. La relación SR se representa de la siguiente manera:

$$SR = (L_j = E_h) \mid (j \neq h) \wedge (BBox_j, BBox_h \in BBoxes_i)$$

Usabilidad

El enfoque ETL es clave en el sistema Xelhua para el desarrollo de soluciones de big data. Un servicio de big data se construye mediante un conjunto de servicios de aplicaciones analíticas que se ejecutan en una determinada secuencia, siguiendo los pasos de ETL para la extracción de datos, su transformación y la carga del resultado en una fuente de datos específica.

El uso del enfoque ETL provee al sistema Xelhua de la característica de generalidad, ya que las etapas de extracción, transformación y carga son independientes entre cada estructura ABox o entidad BBox, siendo posible acoplar los resultados a través de la malla de servicios (BDServ) para dar solución a un problema de big data particular. La característica de generalidad también permite expandir una malla de servicios añadiendo nuevos servicios (estructuras ABox) a la malla utilizando los componentes de extracción y carga.

A partir de lo anterior, una BDServ puede ser considerada como una serie de transformaciones (TR) sobre una determinada secuencia de datos, que son extraídos y/o almacenados en una fuente de datos (source) específica a través de tuberías, entonces:

$$BDServ = \{pipe_1, pipe_2, pipe_3, \dots, pipe_v\}$$

$$pipe = source \rightarrow TR_1 \rightarrow TR_2 \rightarrow \dots \rightarrow TR_s \rightarrow sink \mid TR_i \in R$$

Los pipelines comienzan el procesamiento a partir de una fuente de datos (source) común. El proceso de transformación se realiza a través de cada una de las estructuras ABox hasta producir un resultado final (sink). Los pipelines de un BDServ pueden utilizar una estructura ABox la cantidad de veces necesarias, ya sea dentro de un mismo pipeline, o entre diferentes pipelines. Por ejemplo:

$$\begin{aligned} &source_1 \rightarrow TR_1 \rightarrow TR_2 \rightarrow TR_3 \rightarrow TR_4 \rightarrow sink_1 \\ &= \\ &source_1 \rightarrow ABox_1 \rightarrow ABox_3 \rightarrow ABox_1 \rightarrow ABox_2 \rightarrow sink_1 \end{aligned}$$

El flujo de datos comienza desde la fuente 1 ($source_1$), donde los datos son transformados por la estructura $ABox_1$, cargando el resultado en la estructura $ABox_3$. La estructura $ABox_3$ carga su resultado en la estructura $ABox_1$, y, finalmente, la estructura $ABox_1$ carga el resultado en la estructura $ABox_2$. Debido a las características de las estructuras ABox, éstas les permiten conectarse entre sí, construyendo así estructuras complejas para dar solución a un problema de big data específico.

Disponibilidad y Fiabilidad

En los escenarios del mundo real, varios usuarios pueden hacer peticiones a un conjunto de servicios para procesar datos. En este contexto, si uno de los servicios falla, los usuarios no pueden acceder a él. Una solución a este problema es utilizar la redundancia de servicios [40], por ejemplo, hacer copias de un servicio. El sistema Xelhua, considerando las características de las estructuras ABox, permite la implementación de redundancia de servicios debido a que las estructuras ABox pueden ser "clonadas" y desplegadas en múltiples recursos informáticos. Si una copia de la estructura ABox falla, las demás siguen estando disponibles para procesar los datos.

En escenarios donde no se implementa la redundancia de servicios para el control sobre los fallos, como es el caso de las infraestructuras/plataformas externalizadas (por ejemplo, la nube y *serverless*) comunes en los escenarios de big data. En estos escenarios, es necesario recopilar información sobre el estado actual de los servicios proporcionados y gestionar la carga de trabajo distribuida a las copias de tales servicios. Para lograr este tipo de gestión compleja, se diseñó una nueva versión *serverless* para la tolerancia a fallos y la alta disponibilidad del protocolo Paxos [41] en combinación con un nuevo balanceo de carga y la gestión de la carga de trabajo basada en tablas hash distribuidas y un balanceo de carga probabilístico inspirado en el enfoque de dos-opciones (*two-choices*, por su nombre en inglés).

El acoplamiento de las estructuras ABox en el sistema Xelhua se realiza a través de redes P2P descentralizadas. El esquema descentralizado no sólo permite a las estructuras ABox establecer un consenso entre las entidades participantes de una red de servicios P2P, sino que también mantiene un grado de tolerancia a los fallos a través de la redundancia del servicio, así como el control del intercambio de datos entre las estructuras ABox.

En el esquema descentralizado, se utilizan entidades con diferentes roles como nodos, proponentes, aceptantes y aprendices. El rol de nodo realiza tareas de equilibrio de carga, asignación y distribución de la carga de trabajo. Los proponentes procesan cada solicitud que llega a un servicio (SV). Los aceptadores reciben y procesan cada solicitud y finalmente responden afirmativa o negativamente a los proponentes. Cuando los aceptadores reciben una solicitud, la envían a los aprendices, que, por consenso, deciden reenviar la solicitud a la estructura ABox correspondiente. El protocolo permite que, aunque fallen algunas entidades, las peticiones puedan ser atendidas por el resto de las entidades disponibles sin interrumpir la funcionalidad del servicio de big data.

Siguiendo este mismo proceso, los nodos permiten establecer controles sobre las réplicas de las estructuras ABox. Esto significa que en escenarios de fallos de una estructura ABox, los nodos realizan interacciones entre las estructuras ABox disponibles. La figura 3 ilustra este proceso.

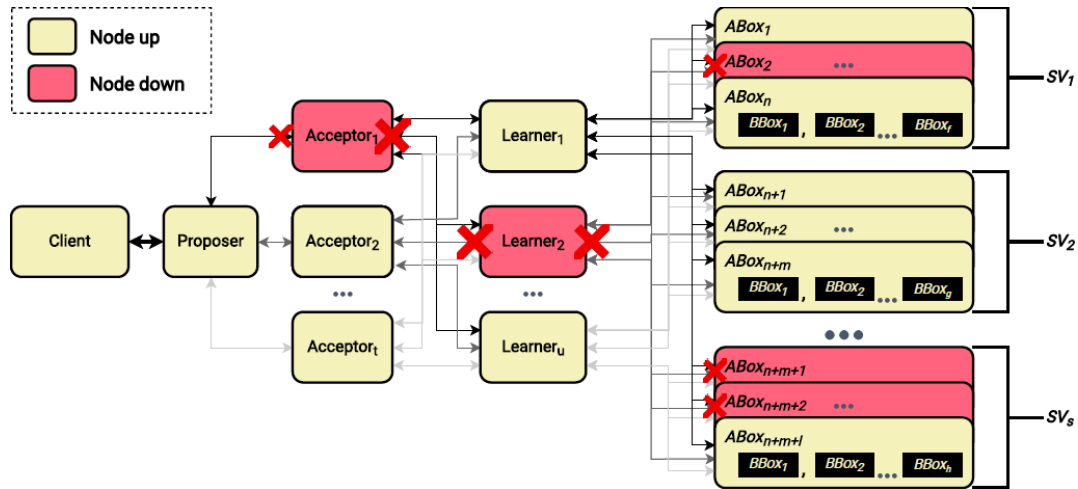


Figura 3. fiabilidad y tolerancia a los fallos del sistema Xelhua a través de un modelo descentralizado

Las tablas hash distribuidas permiten determinar las solicitudes enviadas a las estructuras ABox a través de la red descentralizada (P2P). El uso de este tipo de tablas permite llevar un registro del balance de la carga enviada. Utilizando esta información, los nodos invocan algoritmos de balanceo de carga para distribuir las peticiones de los usuarios a las réplicas de una estructura ABox para compensar los costes de comunicación producidos por la gestión descentralizada.

En este contexto, el sistema Xelhua permite la creación de múltiples réplicas de una estructura ABox con el fin de crear un solo servicio ($SV \subset R$), donde cada estructura ABox en este subconjunto tiene las mismas entidades BBox, que se representa como sigue:

$$ABox_i, ABox_j \in SV_k \mid BBoxes_i = BBoxes_j$$

Implementación del prototipo del sistema Xelhua

El prototipo del sistema Xelhua se implementó mediante el uso de contenedores virtuales basado en la plataforma Docker¹⁵.

Una estructura ABox expone el servicio proporcionado por la aplicación analítica a través de su capa de datos. Esta capa está controlada por un *Building-Block-Middleware* (BB-M), encargado de gestionar y redirigir los datos a las aplicaciones. El envío y recepción de datos se realiza a través de sockets¹⁶, generando conexiones P2P entre los diferentes servicios construidos con estructuras ABox.

El módulo BB-M se comunica con cada aplicación a través de una capa de acceso llamada *Agnostic Black Box Middleware* (BBOX-M). Este módulo se encarga de la ejecución y monitorización de las aplicaciones, gestionando el proceso de ETL en cada entidad BBox. El sistema Xelhua cuenta con una capa de acceso para el usuario, que tiene la función de puerta de acceso a los servicios existentes en la malla, la denominada AG. El AG recibe un conjunto de instrucciones, así como los datos de entrada para

¹⁵ <https://www.docker.com/>

¹⁶ Un socket es una entidad de comunicación bidireccional entre dos aplicaciones de software.

realizar la ejecución coordinada de los servicios. La comunicación entre servicios desplegados en infraestructuras diferentes se implementa a través de la IG.

Monitoreo e indexación

El sistema Xelhua, a través del AG, recibe las configuraciones del usuario, indicando los parámetros de ejecución de cada una de las aplicaciones de las estructuras ABox, el orden de ejecución de las entidades BBox y la secuencia de los servicios a ejecutar (el pipeline), formando así un grafo que parte de los datos de entrada, proporcionados por el usuario, hasta el resultado obtenido tras la ejecución de los servicios solicitados por el usuario. Una vez que las instrucciones llegan al AG, éste se encarga de enviarlas al primer servicio del pipeline para su ejecución. Durante la ejecución del pipeline, el AG sigue monitorizando cada servicio en busca de posibles fallos. Cuando un servicio termina el proceso de transformación de los datos, carga los resultados en una base de datos e informa al AG; este proceso se denomina indexación. Los resultados indexados pueden ser recuperados por el AG para entregarlos al usuario. El proceso de indexación permite a los usuarios acceder a los resultados producidos por los servicios del pipeline intermedios incluso cuando las tareas del BDServ están en ejecución. Sin embargo, el servicio de indexación es opcional para cada servicio. El usuario puede elegir entre acceder a los datos intermedios o mantener el rendimiento evitando la ejecución de procesos adicionales. En la figura 4 se muestran gráficamente cada uno de los componentes descritos anteriormente.

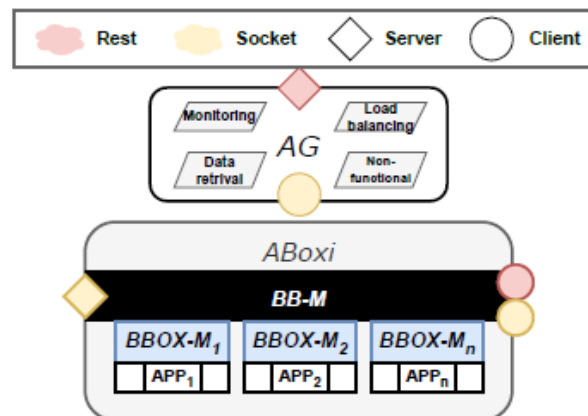


Figura 4. Representación gráfica de los componentes desarrollados para el prototipo funcional.

Definición de aplicaciones para estructuras ABox

El sistema Xelhua gestiona los diferentes servicios proporcionados como una red descentralizada tolerante a fallos. Un servicio está definido por un archivo de configuración en formato JSON¹⁷ que contiene los metadatos de la estructura ABox. Esto significa que la implementación de una estructura ABox se genera a partir de los metadatos definidos por el usuario y

¹⁷ JavaScript Object Notation

recolectados a través de la interfaz de usuario. La figura 5 muestra un ejemplo de un archivo de configuración JSON creado a través de los metadatos definidos por el usuario. El archivo de configuración define las etapas ETL, así como las instrucciones necesarias para ejecutar el servicio.

Las instrucciones, definidas en el archivo de configuración, pueden incluir palabras reservadas (en forma de parámetros), propias del sistema Xelhua, utilizadas como variables de la estructura ABox. El archivo de configuración distingue dos tipos de variables: variables reservadas (en minúscula) y variables de entorno. Las variables reservadas definen información sobre la configuración de la estructura ABox, mientras que las variables de entorno definen los valores que se utilizarán en la estructura ABox. En la figura 5, las variables siguen la sintaxis *@{< nombre_de_variable >}* donde *nombre_de_variable* es el nombre de la variable que se utilizará al llamar al servicio.

```

1 {
2   "NAME_APPLICATION": "Imputation",
3   "INPUT_DATA_FORMAT": ["csv"],
4   "EXTRACT": {
5     "COMPRESS": false
6   },
7   "TRANSFORM": {
8     "COMMAND": "Rscript @CWD\\T.r @"@{
9       SOURCE_PATH}\\ " @"@{
10        SOURCE_FILENAME}\\ " @"@{SINK}
11        output.csv\\ " @"@{columns}\\ " @"@{
12        method}\\ " "
13      "CUSTOM_APP": false
14    },
15    "LOAD": {
16      "OUTPUT_NAMEFILE": "output.csv",
17      "COMPRESS": false,
18      "ignore_list": []
19    }
20  }
21 }

```

Figura 5. Ejemplo de un archivo de configuración para una aplicación de una estructura ABox.

Despliegue de servicios mediante el uso de estructuras ABox

Como parte de la implementación del prototipo del sistema Xelhua, se desplegaron algunas aplicaciones analíticas como servicios. Estos servicios formaron parte de la evaluación experimental, presentado más adelante, y se describen brevemente a continuación:

- Generación de datos bibliográficos (GROBID¹⁸, por sus siglas en inglés). Es una herramienta de aprendizaje automático para transformar documentos técnicos o científicos en PDF al formato TEI/XML¹⁹. Se trata de una herramienta de código abierto para la extracción de texto, conservando los metadatos y las partes de los documentos originales. Los documentos resultantes están en un formato que puede ser fácilmente procesado por una computadora, permitiendo utilizar esta información para el consumo en línea o digital (presentación, visualización, lectura, etc.). La mayoría de las tareas de procesamiento de texto conllevan un alto consumo de recursos del procesador y la

¹⁸ GeneRation Of Bibliographic Data. <http://grobid.readthedocs.io/en/latest/Introduction>

¹⁹ TEI/XML, es un estándar ampliamente usado por bibliotecas, museos, editores, etc. para la representación de textos en formato digital (<https://tei-c.org>)

memoria, incrementando dicho consumo de acuerdo con el número de documentos a procesar.

- Adquisición de fuentes de datos. Conjunto de herramientas para la adquisición de información a partir de fuentes de datos provenientes de repositorios públicos (FTP, Google Drive, Hydroshare, etc.), principalmente.
- Preprocesamiento de textos. Realiza tareas de transformación sobre una fuente de texto que van desde la identificación de palabras en un documento hasta la identificación de entidades nombradas. El resultado del preprocesamiento es utilizado como fuente de entrada para algún algoritmo de minería de textos.
- Agrupación de textos. Es un conjunto de algoritmos para la agrupación de textos mediante el uso del lenguaje de programación Python. El procedimiento consiste en identificar los agrupamientos o clústeres más relevantes a partir de una fuente de texto utilizando algoritmos de agrupamiento o *clustering* de naturaleza distinta. Este enfoque también permite observar cómo cambian los agrupamientos según el algoritmo utilizado. Las tareas de preparación y agrupamiento de textos son demandantes en recursos del procesador y la memoria RAM, debido a los cálculos estadísticos y probabilísticos característicos de estos algoritmos.
- Clasificación de textos. La clasificación de textos conlleva el uso de diferentes algoritmos para extraer información de una fuente de texto. Por ejemplo, los clasificadores de texto para el análisis de sentimientos para detectar la polaridad (por ejemplo, positiva o negativa) en un texto [42]. El análisis de sentimientos es útil en tareas como la medición de la reputación de las marcas, la calificación de las críticas de las películas y los comentarios de los clientes. El análisis de sentimientos se aborda habitualmente como un problema de clasificación que utiliza algoritmos de procesamiento del lenguaje natural y de aprendizaje automático, y que requiere grandes recursos informáticos para manejar, transformar y procesar el texto de entrada.

La figura 6 ilustra un ejemplo de los servicios desplegados en diferentes infraestructuras utilizando el prototipo del sistema Xelhua. En la figura 6, se muestran un conjunto de servicios y réplicas montados en un clúster local y un conjunto de réplicas existentes en nodos del servidor de Amazon. En este contexto, un usuario envía peticiones al AG, que las redirige a los correspondientes servicios disponibles, generando así un flujo de datos (denotado por los números).

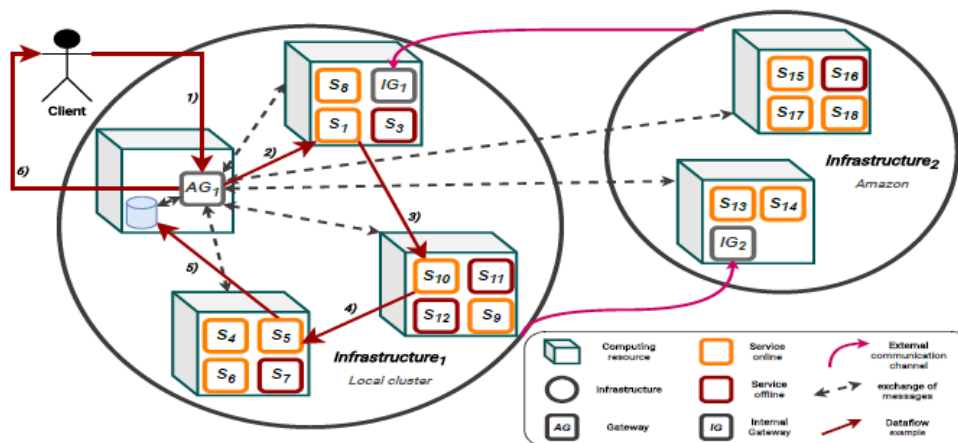


Figura 6. Representación gráfica del modelo de componentes implementado en una infraestructura.

Evaluación experimental y resultados.

La evaluación del prototipo del sistema Xelhua está basada en una metodología de experimentación de dos fases. La primera fase evalúa diferentes algoritmos para definir la configuración óptima que permita la orquestación eficiente y de alta disponibilidad de los servicios proporcionados por el sistema Xelhua. La segunda fase evalúa la implementación de la configuración obtenida en la primera fase.

El proceso de evaluación consideró tres casos de estudio: (i) Procesamiento de documentos, (ii) análisis de sentimientos y (iii) agrupamiento de sinopsis de películas. Para cada uno de los casos de estudio, los usuarios pudieron diseñar una solución, utilizando la interfaz gráfica del sistema Xelhua, a través de los diferentes servicios proporcionados. Se modificaron los parámetros para crear un banco de pruebas con el fin de realizar la evaluación experimental asociada a cada caso de estudio.

Métricas

Para realizar la evaluación experimental, se diseñó una estructura ABox especial para el monitoreo de las siguientes métricas:

- Tiempo de respuesta. Es el tiempo que transcurre desde que se realiza una petición, a una estructura ABox, hasta que el usuario final ve una respuesta.
- Tiempo de procesamiento. Es el tiempo que tarda un servicio para ejecutar su procesamiento sin tener en cuenta los tiempos de transferencia de datos, de intercambio de mensajes, etc.
- Tiempo de intercambio de mensajes. Es el tiempo total que transcurre en la ejecución del intercambio de mensajes entre las diferentes entidades de la malla de servicios.

Infraestructura

Para la evaluación experimental se utilizaron cuatro configuraciones de infraestructura diferentes que se describen en la siguiente tabla:

Config.	Nodos	Núcleos por nodo	Hilos por núcleo	Número de procesadores	Detalles del procesador	RAM	Notas
A	1	4	2	8	Intel Core i7 a 1.8 GHz	8 Gb	Computadora personal
B	3	4	1	4	Intel 3.0 GHz	16 Gb	Instancias t2.xlarge EC3
C	2	4	1	4	Intel 3.0 GHz	16 Gb	Instancias t2.xlarge EC3 acopladas por un IAG
D	1	16	1.5	24	Intel Core i9 a 3.2GHz	128 Gb	Servidor privado

Tabla 2: Especificaciones de la infraestructura. El número de nodos (recursos informáticos), los núcleos por nodo, las tareas paralelas por núcleo, la memoria RAM y una breve descripción de los equipos.

Evaluación del rendimiento y la disponibilidad del modelo descentralizado de Xelhua

En esta sección se presenta la evaluación experimental del sistema Xelhua utilizando los protocolos de consenso: Paxos [43] para alta disponibilidad, Chrod [44] para el balanceo de carga y Bully [44] para la eficiencia.

El conjunto de datos utilizado en esta fase de experimentación corresponde a datos meteorológicos obtenidos por sensores terrestres distribuidos en todo el territorio mexicano y cuenta con un total de 65,388 registros (10.2 Mb). Cada registro contiene valores diarios de 2018 a 2019 de temperatura, humedad, presión barométrica, precipitación, radiación solar, velocidad y dirección del viento, y velocidad y dirección de la ráfaga.

El prototipo del sistema Xelhua se utilizó para crear una solución basada en servicios de preprocesamiento de datos y clasificación. Para el preprocesamiento de datos, se desplegaron servicios para realizar la asignación, la eliminación de valores atípicos, la normalización y el muestreo del conjunto de datos. Posteriormente, los datos fueron procesados utilizando redes neuronales a través de los siguientes algoritmos: Redes neuronales convolucionales (CNN), redes neuronales de memoria a largo plazo (LSTM) y redes neuronales recurrentes (RNN). La carga de trabajo se distribuyó en múltiples réplicas de los servicios para procesar los datos en paralelo y mejorar el rendimiento del servicio.

La figura 7 muestra una disminución del tiempo de procesamiento del sistema al incluir réplicas de los servicios (*peers*) proporcionados, distribuyéndose las tareas a realizar. Sin embargo, el tiempo de paso de mensajes aumenta exponencialmente cuando aumentan los servicios, esto debido a que se debe implementar una comunicación de difusión diferente para cada nuevo servicio.

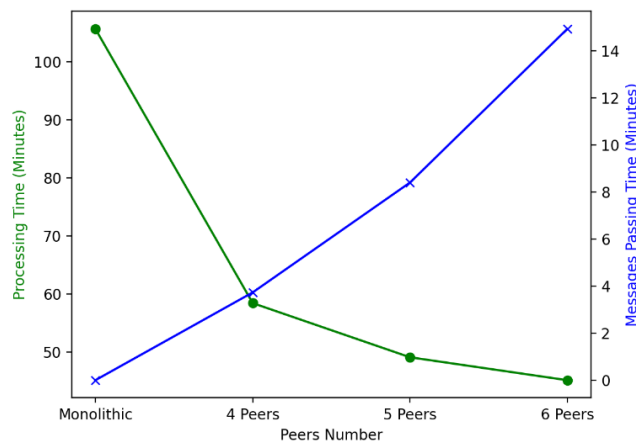


Figura 7. Comparativa del tiempo de procesamiento entre el desempeño de los servicios de aprendizaje profundo al añadir nuevos servicios a la red descentralizada y el desempeño de paso de mensajes al elegir servicios para procesar la carga de trabajo.

La figura 8 muestra cómo el paso de mensajes es similar al tiempo de procesamiento de la carga para las tareas que requieren muy poco tiempo de procesamiento (tareas de preprocesamiento). Mientras que para las tareas que requieren más tiempo (tareas analíticas centrales), este tiempo de paso de mensajes tiende a ser insignificante en comparación con el tiempo de procesamiento.

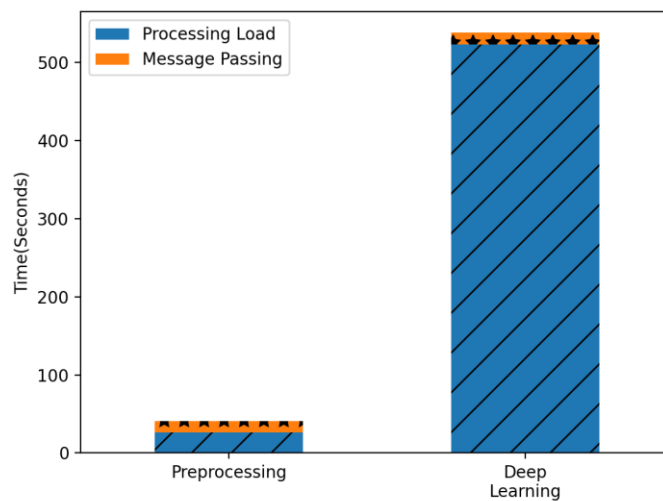


Figura 8. Comparativa del tiempo de procesamiento de las solicitudes de carga de trabajo y el tiempo de paso de mensajes producido por las tareas de preprocesamiento y aprendizaje profundo.

La figura 9 muestra la diferencia de tiempos de procesamiento entre los algoritmos de Paxos (alta disponibilidad), Chord (tabla hash distribuida) y Bully (elección económica de un coordinador). Paxos muestra el menor rendimiento entre los tres algoritmos debido a que produce el mayor intercambio de mensajes. Bully fue el más rápido en encontrar un consenso para definir un líder, mientras que Chord fue la mejor opción a largo plazo. En resumen, Paxos emite mensajes, Bully emite mensajes, pero sólo una vez para elegir al líder, y Chord utiliza el intercambio de mensajes unicast.

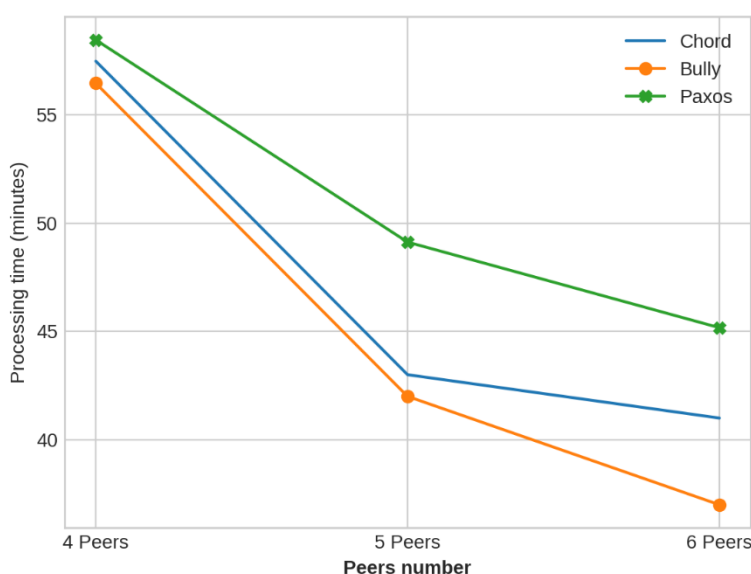


Figura 9. Tiempo de procesamiento del modelo descentralizado del sistema Xelhua al utilizar Paxos, Chord y Bully.

Como resultado de esta evaluación, el modelo descentralizado implementado en el sistema Xelhua incluye una combinación de estos tres algoritmos: el algoritmo Bully se utiliza para definir qué estructura ABox será la encargada de la distribución de datos, Chord se utiliza para distribuir las peticiones de los usuarios a los servicios, mientras que Paxos se utiliza para mantener el servicio de big data que recibe las peticiones de los usuarios.

Casos de estudio

Caso de estudio 1: procesamiento de documentos

Para el procesamiento de documentos se utilizó la herramienta GROBID para procesar documentos PDF. GROBID está desarrollado en Java y puede implementarse como un servicio independiente, una API de Java o un servicio web. Para este caso de estudio se implementó GROBID como servicio web debido a que dispone de múltiples configuraciones para el comportamiento y el procesamiento de los documentos, en comparación con las otras opciones. Para los experimentos se utilizó una configuración por defecto del servicio web que procesa los documentos en la opción de texto completo (identificación de párrafos, títulos de sección, llamadas de referencia, figuras, tablas, etc.). Esto se consiguió mediante clientes Python en modo de procesamiento por lotes, lo que permite utilizar multihilos para la paralelización de tareas.

Los datos de entrada para la herramienta GROBID fueron un conjunto de datos sintéticos con 200 archivos PDF. Con el objetivo de mantener la coherencia en el tamaño de los PDF, utilizamos 200 copias de un único archivo

PDF para crear este conjunto de datos. Este PDF corresponde a una versión preliminar del artículo [45] con un tamaño de aproximadamente 1 MB.

Para la adaptación de la herramienta como flujo de trabajo, se diseñaron tres configuraciones utilizando la herramienta GROBID como caso de estudio controlado, para obtener el comportamiento del servicio en la malla. Estas configuraciones se implementaron y probaron en las infraestructuras A y C, correspondientes a un entorno simulado de infraestructura local y múltiple, respectivamente. En este caso particular, las infraestructuras B y C son bastante similares dado que ambas utilizan dos nodos con las mismas características, por lo que se omitió B. Las características de cada configuración de GROBID se indican a continuación (figura 10):

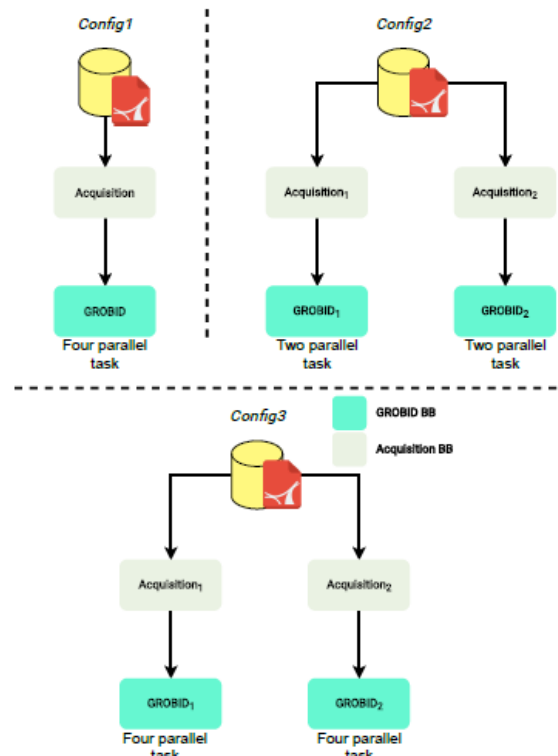


Figura 10. Estructura y configuración BDServ

- Configuración 1. 200 PDFs procesados por un pipeline usando una instancia de GROBID con cuatro trabajadores (cuatro tareas paralelas).
- Configuración 2. 200 PDFs procesados por dos pipelines con una instancia de GROBID con dos trabajadores (100,100).
- Configuración 3. 200 PDFs procesados por dos pipelines con una instancia de GROBID con cuatro trabajadores (100,100).

Como se puede ver en la figura 11, el tiempo de respuesta (el tiempo que tarda en terminar de procesar y responder al usuario) para GROBID con la configuración 1 y la configuración 2 es muy similar incluso cuando se implementan en diferentes infraestructuras. Entonces, es posible utilizar la malla de servicios en entornos de clúster o múltiples infraestructuras, esperando resultados similares (sin tener claro la latencia de la red). En cambio, para la configuración 3 se observa un cambio de comportamiento. Mientras que en un entorno local la variabilidad de los tiempos aumenta, en un entorno de nube con IAG los tiempos disminuyen respecto a configuración

1 y configuración 2. La explicación de este comportamiento radica en la característica de elasticidad. Mientras que, por un lado, la herramienta GROBID permite escalar en local (con hilos), con el uso de la malla de servicios es posible escalar a otros recursos de computación, pudiendo utilizar más recursos y dividir la carga de forma más eficiente.

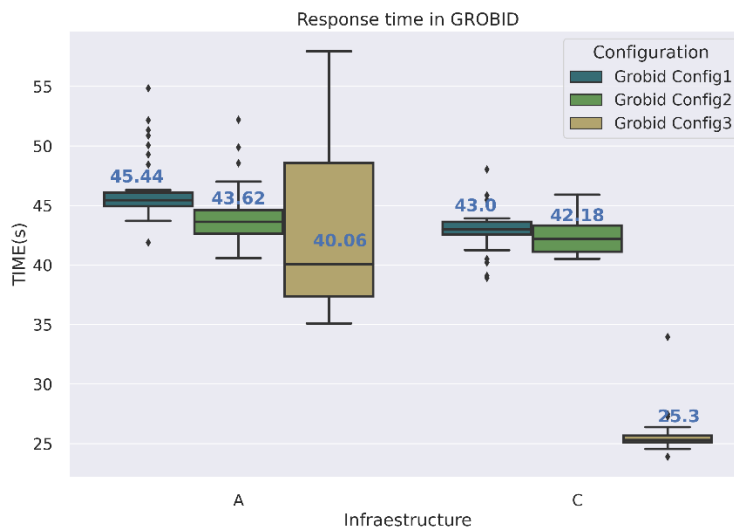


Figura 11. Tiempos de respuesta de GROBID para la infraestructura A y C.

Caso de estudio 2: Análisis de sentimientos

La pandemia provocada por el virus Covid-19 ha generado diversas opiniones entre los usuarios de las redes sociales, lo que ha repercutido en el estado psicológico de las personas. Una forma de captar las emociones de la gente es a través del análisis de sentimientos de los contenidos publicados en las redes sociales (por ejemplo, Twitter), lo que puede ayudar (inicialmente) a comprender algún tipo de problema más amplio. El objetivo de este experimento es poner a prueba el prototipo del sistema Xelhua a través de la configuración y ejecución de algoritmos de aprendizaje automático para la tarea de análisis de sentimientos sobre tuits relativos al tema de Covid-19.

El proceso de análisis de sentimientos fue abordado a través de cuatro diferentes algoritmos de clasificación: Red neuronal de memoria a largo y corto plazo (LSTM²⁰, por sus siglas en inglés), Perceptrón multicapa [46] (MLP²¹, por sus siglas en inglés), Naive Bayes y Máquina de vectores de soporte (SVM²², por sus siglas en inglés) (ver figura 12).

²⁰ Long short-term memory

²¹ Multilayer perceptron

²² Support vector machine

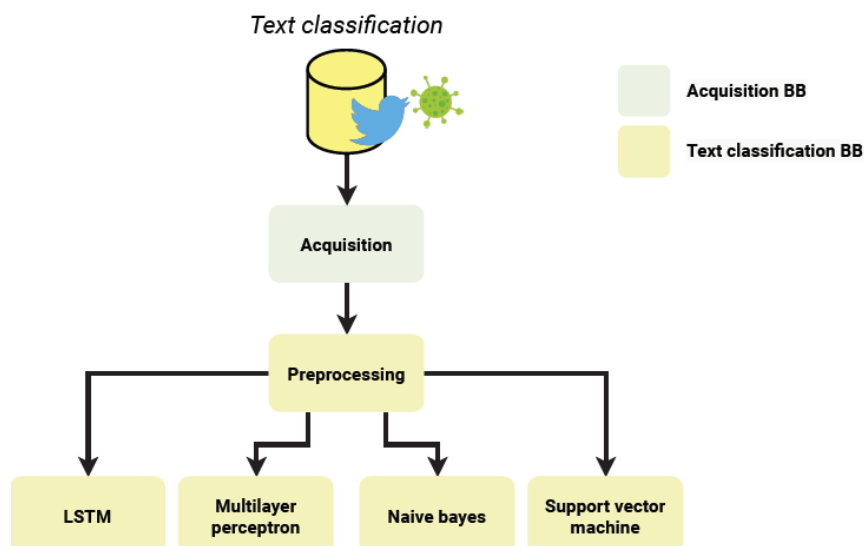


Figura 12. Representación gráfica del modelo de análisis de sentimientos.

El escenario de este experimento se compone de las siguientes etapas:

- **Análisis de datos.** Para este caso de estudio, se consideró una clasificación binaria de tuits. El conjunto de datos utilizado en la prueba consiste en 49,956 tuits en inglés²³. En esta fase, se analizó y transformó el conjunto de datos correspondientes a las categorías de opiniones positivas y negativas. Se eliminó la categoría de opiniones neutras y las categorías de opiniones extremadamente positivas y extremadamente negativas se fusionaron con las correspondientes categorías de positivas y negativas.
- **Preprocesamiento.** Esta etapa considera tareas de filtrado sobre el texto de entrada, la obtención de características y su posterior vectorización. Utilizando la librería NLTK en Python [34], las tareas aplicadas fueron:
 - Eliminación de palabras vacías (*stopwords*). Se eliminan aquellas palabras que no tienen información útil: artículos, preposiciones, etc.
 - *Stemming*. Transformación de palabras a su raíz utilizando el algoritmo de *SnowBallStemmer*.
 - Tokenización. El texto se dividió en unidades mínimas de lenguaje. De este paso se obtiene un diccionario de palabras, útil para codificación.
 - Codificación. De acuerdo con el diccionario de palabras, se realiza una codificación de los tuits, obteniendo una representación numérica de las palabras. Las secuencias obtenidas se acumulan a la misma longitud (utilizando la secuencia de *Keras Pad*). El resultado es la representación del texto como vectores que se utilizarán como entrada para los algoritmos de aprendizaje automático.

²³ El conjunto de datos analizados está disponible en el repositorio de Kaggle:
<https://www.kaggle.com/datatattle/covid-19-nlp-text-classification>

- División de datos. Se dividen los datos aleatoriamente para generar los conjuntos de entrenamiento y prueba (80-20 respectivamente).
- Configuración y entrenamiento del modelo. Los cuatro algoritmos fueron configurados y ejecutados con los datos resultantes de las etapas anteriores. A continuación, se presenta la configuración utilizada para cada algoritmo.
 - Red neuronal de memoria a largo y corto plazo (LSTM). Se trata de una variante de la Red Neural Recurrente (RNN²⁴, por sus siglas en inglés), útil para las secuencias de datos [21]. A diferencia de las redes *feedforward*, la RNN mantiene la memoria para preservar el error que puede ser retro propagado, lo que es útil para el aprendizaje continuo en los pasos de tiempo. En este caso, la LSTM fue configurada con los siguientes parámetros:
 - Incorporación. Utilizando un vocabulario de 68,921 palabras, una salida de 32 dimensiones y una entrada de 200 palabras.
 - SpatialDropout1D de 0.25
 - LSTM. 50 unidades, añadiendo un *dropout* (recurrente) de 0.5
 - Un *dropout* de 0.2
 - Capa densa (1 unidad) con función de activación sigmoidea.
 - Entropía cruzada binaria (pérdida), optimizador Adam y un enfoque en la precisión (métrica). Además, utilizamos épocas variadas (explicadas más adelante), y un tamaño de lote de 128.
 - Perceptrón multicapa (MLP). El MLP es una Red Neural Artificial (ANN²⁵, por sus siglas en inglés) compuesta por múltiples capas, útil para resolver problemas que son no linealmente separables (utilizando el algoritmo de entrenamiento de retro propagación). Se configuró una arquitectura MLP compuesta por la capa de entrada, dos capas ocultas (con 200 nodos y activación Relu), y un nodo como salida (activación sigmoidea).
 - Naive Bayes (NB). Configuramos un Naive Bayes gaussiano con los parámetros por defecto del entorno *Sklearn* [47].
 - Máquina de vectores de soporte (SVM). Es un modelo no paramétrico para la clasificación binaria. En los experimentos, se utilizó una SVM lineal [48].

Para obtener el comportamiento general del BDServ con y sin la malla de servicio, se realizaron un total de treinta y dos iteraciones para cada experimento. Además, para facilitar la ejecución de los experimentos, se utilizó un conjunto de datos reducido de 10,000 tuits. La figura 13 muestra el contraste de los tiempos obtenidos en la ejecución del BDServ para cada configuración de infraestructura. Aunque se observó un comportamiento consistente, varía en la escala de tiempos, donde la malla del servicio añadió

²⁴ Recurrent Neural Network

²⁵ Artificial Neural Network

una sobrecarga que va del 1% al 2% para este BDServ. En este contexto, tiene sentido utilizar la malla de servicios para este BDServ, sacrificando la adición de un 2% a los tiempos de respuesta, pero añadiendo las ventajas que el modelo proporciona.

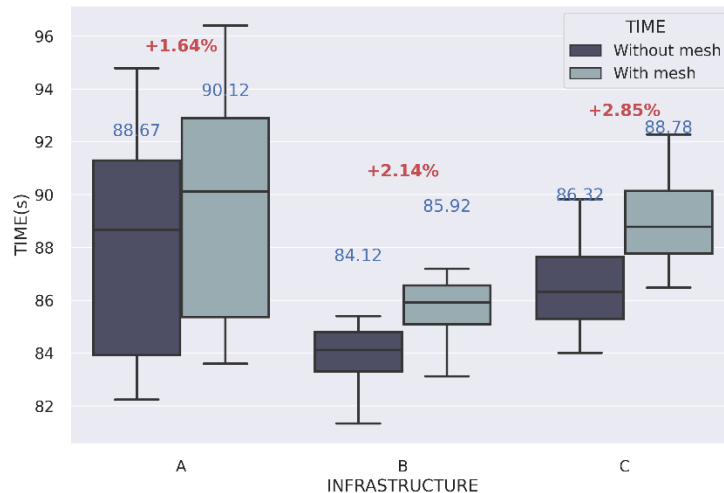


Figura 13. Tiempo de respuesta del modelo de análisis de sentimientos.

Por otro lado, el BDServ fue más eficiente al procesar el conjunto de datos completo. La LSTM obtuvo el mejor rendimiento con una puntuación de precisión del 85% en comparación con el 51%, 52% y 54% obtenidos por MLP, NB y SVM, respectivamente. Basado en esto, el BDServ fue rediseñado para experimentar con diferentes tamaños de lotes y épocas.

El BDServ se ejecutó en la configuración A de la infraestructura utilizando tres instancias del algoritmo LSTM, variando cada una el tamaño del lote (32, 64, 128 y 256) y un conjunto de épocas (5, 10, 20, 30 y 50). La figura 14 muestra los tiempos de respuesta con y sin la malla de servicio para cada instancia, integrando el porcentaje de sobrecarga y la cantidad de datos producidos (en este caso, informes con las métricas evaluadas). El *overhead* en tiempos para cada instancia es muy similar, siendo inferior a un minuto, esto es, a medida que aumenta el tiempo de procesamiento, la sobrecarga se vuelve menos significativa. Lo anterior resulta interesante para las aplicaciones que procesan grandes volúmenes de datos, ya que no hay una relación directa entre los procesos de la aplicación y la malla del servicio. También, la figura 14, muestra una disminución de cuatro horas (en torno al 30%) al procesar las configuraciones en tres instancias frente a utilizar sólo una.

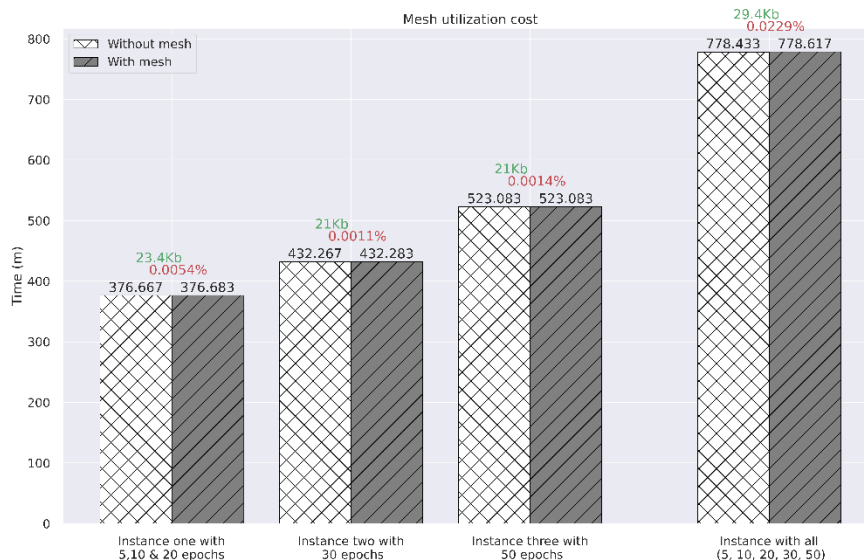


Figura 14. Costo de la malla al incrementar el tiempo de procesamiento de la LSTM.

Por último, se ejecutó un conjunto de experimentos con fallos intencionados en los servicios. Para esta prueba se utilizó la configuración C de la infraestructura ya que los tiempos de detección de fallos son más notables, esto debido a que los datos pasan por el IAG. Las características de los fallos intencionados fueron

- Provocar el fallo de todas las réplicas del servicio de preprocesamiento en N_1 . Esto hace que el sistema redirija las peticiones a N_2 a través del IAG.
- Provocar el fallo de todas las réplicas del servicio de clasificación en N_2 . Esto redirige todas las peticiones a N_1 .

Los resultados obtenidos se muestran en la figura 15. A mayor número de fallos, el tiempo de paso de los mensajes y la transferencia de datos aumentan debido a que la detección de fallos hace que las peticiones sean redirigidas a un servicio externo, que se ve afectado por la latencia de la red. Sin embargo, esto demuestra la eficiencia para detectar los fallos y, en ciertos casos, resolverlos redirigiendo las peticiones.

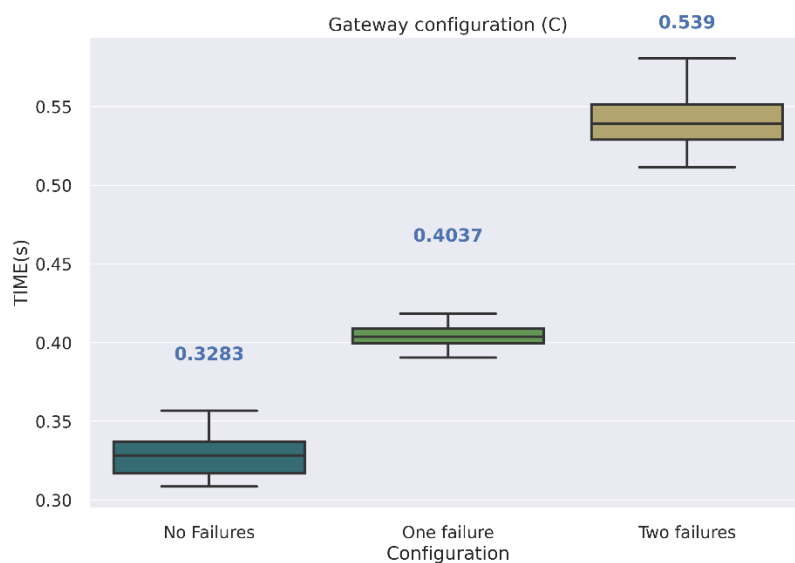


Figura 15. Evaluación del tiempo de intercambio de mensajes cuando se detectan fallos.

Caso de estudio 3: Agrupación de sinopsis de películas

Para este caso de estudio, se utilizaron tres algoritmos de agrupamiento de naturaleza distinta para la identificación de grupos a partir de texto: *K-means*, agrupamiento jerárquico y el algoritmo de *Latent Dirichlet Allocation* (LDA) [49], para el modelado de temas²⁶, que se considera un tipo de agrupamiento de texto (figura 16).

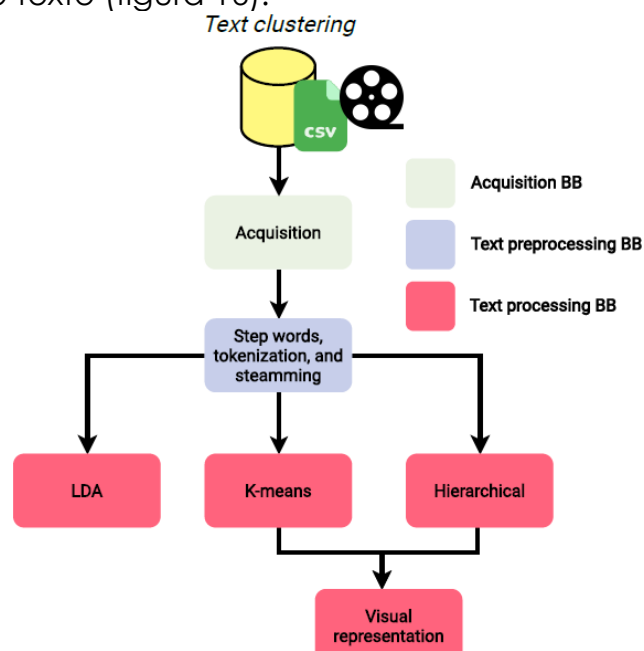


Figura 16. Representación gráfica de las aplicaciones para el agrupamiento de sinopsis de películas.

²⁶ El modelado de temas es un enfoque de minería de textos para el descubrimiento de textos abstractos a partir de un conjunto de textos y utilizando modelado estadístico. Un tema se define como una estructura semántica dentro de un texto.

Para la experimentación se utilizó un conjunto de datos de películas definido por: el título de la película, el género y la sinopsis²⁷. Los textos corresponden a las sinopsis, extraídas de IMDB²⁸, de las 100 mejores películas de todos los tiempos seleccionadas por Brandon Rose²⁹.

Antes de realizar la identificación del grupo de cada sinopsis, es necesario realizar el preprocesamiento del texto: eliminación de palabras vacías, tokenización, lematización y *stemming*.

Los algoritmos de K-means y clustering jerárquico únicamente reconocen elementos numéricos para su procesamiento; por lo tanto, para el agrupamiento de texto, éste debe ser mapeado en un espacio numérico preservando la importancia intrínseca de las palabras. Esto se consigue calculando la puntuación TF - IDF de cada una de las palabras. De esta forma, un documento se representa como un vector de valores TF-IDF.

Para evaluar la calidad de los grupos generados, se utilizan índices de validez de los agrupamientos, encargados de medir la tendencia de agrupación de los elementos. El mejor valor de la función significa que la partición es correcta.

El algoritmo de K-means sigue un enfoque particional para agrupar elementos en k grupos predefinidos. El algoritmo parte de la base de que existen grupos distintos no solapados con forma elipsoidal que tienen un centroide. De este modo, el centroide atrae a los elementos más cercanos a su grupo, donde cada elemento tiene un único grupo. La similitud entre elementos se calcula mediante la función de similitud euclidiana. En la experimentación se calculó el valor más adecuado para k utilizando los índices de validez de agrupamientos *Silhouette* [50] y *Calinski- Harabasz* [51]. Se ejecutaron 100 rondas de agrupación por cada valor de k, de 2 a 20 grupos.

El algoritmo de clustering jerárquico identifica grupos de elementos en un enfoque de árbol jerárquico. A diferencia del algoritmo de K-means, este algoritmo no requiere definir apriori el número de grupos (k). En este experimento, se realizó el enfoque aglomerativo con la vinculación por varianza mínima de Ward, esto significa que, en cada iteración, cada par de grupos con distancia mínima intragrupo³⁰ se fusiona. Además, para el clustering jerárquico, se utilizaron los índices de validez de clúster de *Silhouette* y *Calinski-Harabasz* para calcular el número de grupos más adecuado. Se ejecutaron 100 rondas de clustering, formando de 2 a 20 grupos.

El algoritmo LDA es uno de los algoritmos más utilizados en el análisis de textos. Se trata de un modelo temático estadístico para identificar estructuras semánticas ocultas (temas) en el texto. Asume que cada documento está constituido por varios conjuntos de palabras (temas), y que cada documento es una mezcla de temas, por lo que cada palabra del documento contribuye a algunos temas del documento. LDA calcula la contribución de cada palabra a cada tema, de modo que algunos temas son más representativos

²⁷ El conjunto de datos analizado está disponible en el siguiente repositorio de GitHub: <https://github.com/adaptivez/docsclustering>. Una versión preprocesada puede encontrarse en <https://doi.org/10.6084/m9.figshare.19100057>

²⁸ <http://www.imdb.com>

²⁹ <http://brandonrose.org/top100>

³⁰ La distancia intragrupo calcula la distancia entre los elementos de un mismo grupo.

de un documento. Este algoritmo se utiliza habitualmente para clasificar los documentos en un tema determinado. Inicialmente el algoritmo opera directamente sobre las palabras, internamente realiza una transformación de las palabras a un espacio latente de palabras mediante distribuciones Dirichlet y un modelo estadístico llamado Descomposición de Valor Singular. Esto produce estructuras latentes de Dirichlet (los temas). Primero se utilizan las palabras para modelar los temas, y luego los temas para modelar los documentos.

La evaluación de la calidad de los temas obtenidos a partir de los documentos se calculó a partir de una puntuación de coherencia por cada tema. Esta puntuación mide el grado de similitud semántica entre las palabras con mayor puntuación del tema. Existen diferentes medidas de coherencia [52], las medidas implementadas en este experimento fueron:

- C_uci utiliza la Información Mutua Puntual (PMI³¹, por sus siglas en inglés) para cada par de palabras con mayor puntuación.
- C_umass utiliza una medida de confirmación asimétrica entre los pares de las palabras con mayor puntuación, que se basa en la probabilidad condicional logarítmica.
- C_npmi se basa en la coherencia C_uci, pero utilizando la información mutua normalizada por puntos (NPMI³², por sus siglas en inglés).
- C_v utiliza una combinación de la NPMI y la similitud del coseno entre los pares de palabras con mayor puntuación.

Se realizaron cincuenta ejecuciones de LDA, cada una de las cuales calculaba las puntuaciones de coherencia. En este caso, a diferencia de los índices de validez de los agrupamientos, la prueba t-test se calculó para cada conjunto de valores de cada puntuación de coherencia. La prueba t-test se utiliza para determinar si hay diferencia estadísticamente significativa entre las medias de dos grupos de valores (valores de la distribución t). La hipótesis nula se planteó como "hay diferencia estadística", que se rechazó para las puntuaciones C_uci, C_umass y C_npmi, y se aceptó para la puntuación C_v. Los resultados se muestran en la figura 17, donde se observa una ligera variación del BDServ entre las diferentes configuraciones de infraestructura, sin embargo, la diferencia entre usar o no usar la malla de servicio es consistente.

³¹ Point-wise Mutual Information

³² Normalized Point-wise Mutual Information

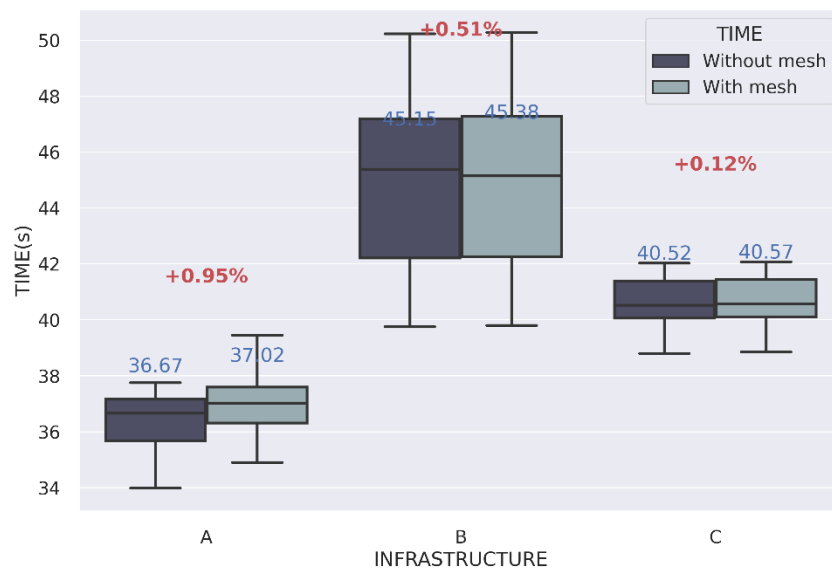


Figura 17. Tiempo de respuesta obtenido.

Por otro lado, se realizó un conjunto de iteraciones para analizar los resultados generados por cada algoritmo (servicio). Los índices de validación de los clusters se utilizaron para rechazar la hipótesis nula, es decir, que la variabilidad de los resultados obtenidos por el agrupamiento es inferior al 0.05%. La figura 18 muestra los tiempos de respuesta al utilizar la malla de servicios, estos tiempos corresponden a la ejecución del algoritmo cincuenta veces y al cálculo de los índices de validación de agrupamiento.

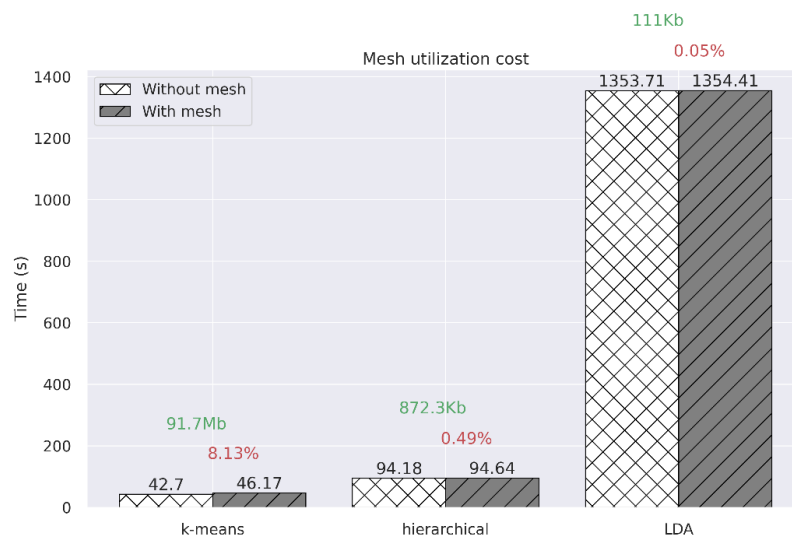


Figura 18. Costo de la malla al incrementar el tiempo de procesamiento

En este experimento se observa el mismo comportamiento observado para SAW, ya que, al aumentar el tiempo de procesamiento de una aplicación, la sobrecarga se vuelve insignificante. Además, se observa que para el algoritmo de K-means los datos resultantes son más grandes que los del resto de los algoritmos, lo que afecta directamente a la sobrecarga al utilizar la aplicación porque los resultados pasan por un proceso de

indexación. Por el contrario, LDA muestra una sobrecarga de un segundo que es mayor que la del algoritmo jerárquico, sin embargo, ésta es sólo del 0.05% sobre el tiempo de ejecución. En este contexto, tiene sentido utilizar la malla de servicios para publicar esta aplicación como un servicio disponible para múltiples usuarios y con las características no funcionales descritas en la sección 3. Finalmente, se muestra un comportamiento eficaz en los algoritmos, obteniendo resultados positivos para los índices de validación de clústeres, mostrados en la Figura 19.

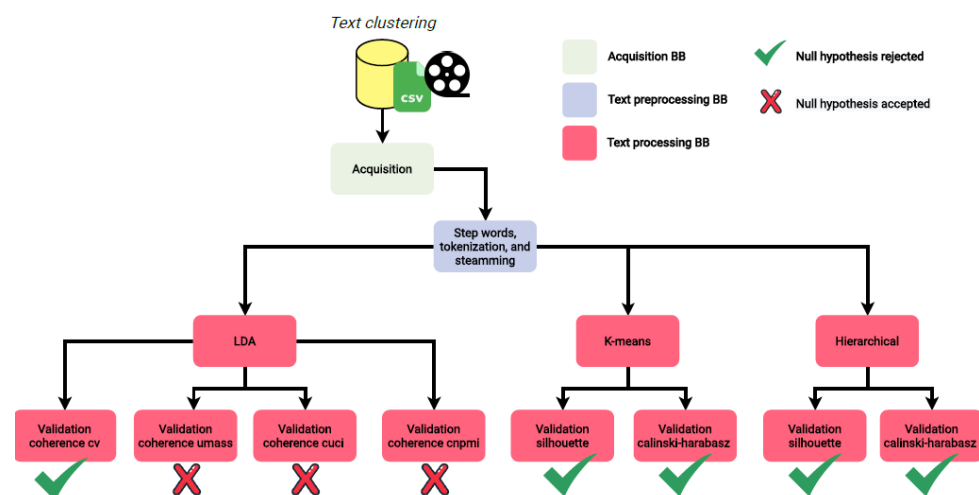


Figura 19. Diseño gráfico del BDServ con validación de índices y resultado obtenido.

Caso de estudio 4: Estudio exploratorio sobre el cáncer

Para demostrar la característica recursiva y el enfoque basado en datos del sistema Xelhua, se diseñó el presente caso de estudio para el análisis de la tasa de mortalidad debida al cáncer. Este conjunto de datos tiene un tamaño aproximado de 110 MB y contiene registros de recuentos y tasas de mortalidad de diferentes tipos de cáncer en México dividida en rangos de edad, sexo, ciudad y estado durante el periodo de 2000 a 2020.

El análisis exploratorio es una tarea que requiere el procesamiento y análisis de múltiples resultados generados en base a diferentes combinaciones de parámetros, por ejemplo, los resultados obtenidos de una regresión logística pueden diferir en función de las variables utilizadas, de la cantidad de datos, o de si estos datos pertenecen a un estado, ciudad o a toda la república mexicana. En este contexto, las estructura ABox del sistema Xelhua disponen de herramientas que permiten el tratamiento de conjuntos de datos estructurados en paralelo mediante la definición de una metodología basada en variables del propio conjunto de datos. Esta metodología en niveles consiste en la segmentación del conjunto de datos original en base a las combinaciones de los valores únicos de las variables del conjunto de datos que un usuario elija (de forma similar a un proceso de MapReduce³³).

³³ MapReduce es un modelo de programación y una implementación asociada para procesar y generar grandes conjuntos de datos con un algoritmo paralelo y distribuido en un clúster

El proceso de experimentación está basado en tres variables: (i) la causa de muerte o tipo de cáncer (Lv.1), (ii) el estado (Lv.2) y (iii) el año (Lv.3). El conjunto de datos original se dividió en n (85) subconjuntos, donde n es el número de tipos de cáncer diferentes; cada subconjunto n fue dividido en m partes (32), donde m es el número de estados; y finalmente cada subconjunto m se dividió en q partes (20) donde q es el número de años (20). En total se generaron 85 subconjuntos para el nivel 1 (Lv.1), aproximadamente 2720 para el nivel 2 (Lv.2), y aproximadamente 43150 para el nivel 3 (Lv.3). Para los niveles 1 y 2 el número varía dependiendo de los datos, por ejemplo, para un estado específico puede no haber datos de 2000 a 2010, o no haber datos para un tipo específico de cáncer.

Para este caso de uso se diseñó un BDServ con cuatro estructuras ABox para generar productos visuales que nos permitan analizar rápidamente el comportamiento de los datos, definiendo los niveles de los datos a procesar para cada uno de ellos: las tasas por rangos de edad (Lv.1-chart), un mapa de ciudades georreferenciado de las tasas de cáncer (Lv.1-map), una regresión lineal de las tasas de cáncer por año para cada estado (Lv.1- 2-regression), y un histograma de las tasas de cáncer por rango de edad para cada año y estado (Lv.3-chart). Por último, se especificó el número de trabajadores (hilos) que hay que desplegar para cada estructura ABox.

El procedimiento del proceso de experimentación se describe a continuación:

- Una estructura ABox se encargará de la división del conjunto de datos según las variables definidas por el usuario (S en la Figura 20).
- Las instrucciones serán heredadas por las cuatro estructuras ABox hijo de los productos visuales. Cada una de estas estructuras ABox será replicada automáticamente w número de veces (según el número de trabajadores especificado) y éstos cambiarán su rol a despachador.
- Una vez desplegadas las nuevas réplicas, los nuevos despachadores distribuirán las tareas (los subconjuntos de datos a procesar según el nivel) a cada una de sus réplicas, controlando los errores y equilibrando la carga de trabajo.

Este modelo sigue un proceso recursivo, ya que cada réplica a su vez puede generar más réplicas. Sin embargo, para este caso particular las réplicas sólo están definidas para procesar los subconjuntos siguiendo el modelo ETL, tomando los datos de entrada, procesándolos con la aplicación dentro, e indexando los resultados.

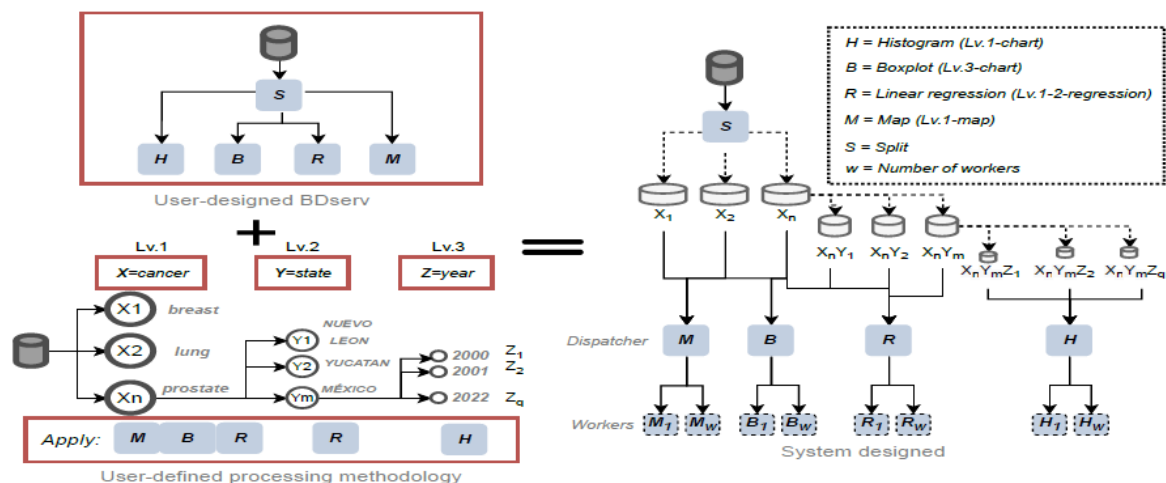


Figura 20. Representación gráfica del BDServ con índices de validación y resultado obtenido.

La evaluación del presente caso de estudio contempla el tiempo de servicio sin considerar fallos y considerando fallos. La figura 21 muestra el tiempo de servicio de cada una de las estructuras ABox y su comportamiento a medida que aumentan los trabajadores. Lv.3-chart fue la estructura ABox con mayor tiempo de servicio, ya que es el único encargado de procesar el nivel 3, lo que supone un total de aproximadamente 43150 histogramas. Sin embargo, al aumentar el número de trabajadores, se observa una mejora en el rendimiento, disminuyendo los tiempos de servicio en más de un 70% al llegar a los 12 trabajadores. Esto demuestra la capacidad del sistema Xelhua para generar soluciones bajo demanda basadas en el diseño del usuario y la selección de datos a procesar, gestionando automáticamente el despliegue en una infraestructura informática y manteniendo la fiabilidad de los servicios a través del despliegue de réplicas de estructuras ABox.

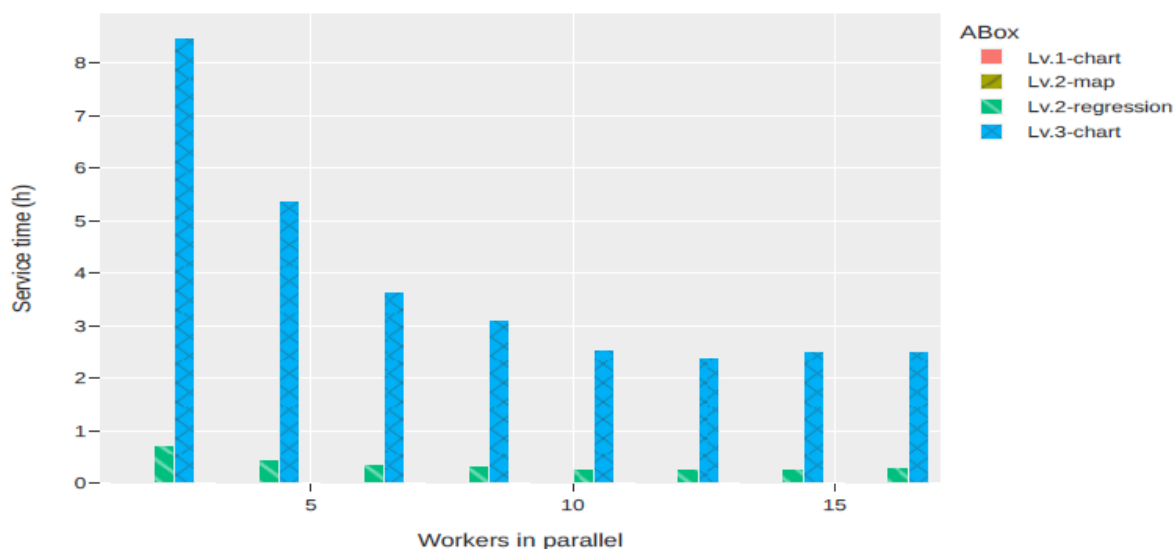


Figura 21. Tiempos de servicio escalando el número de trabajadores

Cada estructura ABox es un servicio en espera de peticiones, por lo que aquellas peticiones que no fueron completadas por alguna estructura ABox son redirigidas a las restantes réplicas que aún están disponibles, por lo que

cada réplica puede procesar varias peticiones simultáneas. Sin embargo, cada réplica es independiente y tiene recursos computacionales específicos y limitados, por lo que los tiempos de servicio aumentan a medida que las peticiones se ponen en cola. La figura 22 ilustra el tiempo de servicio cuando existen fallas.

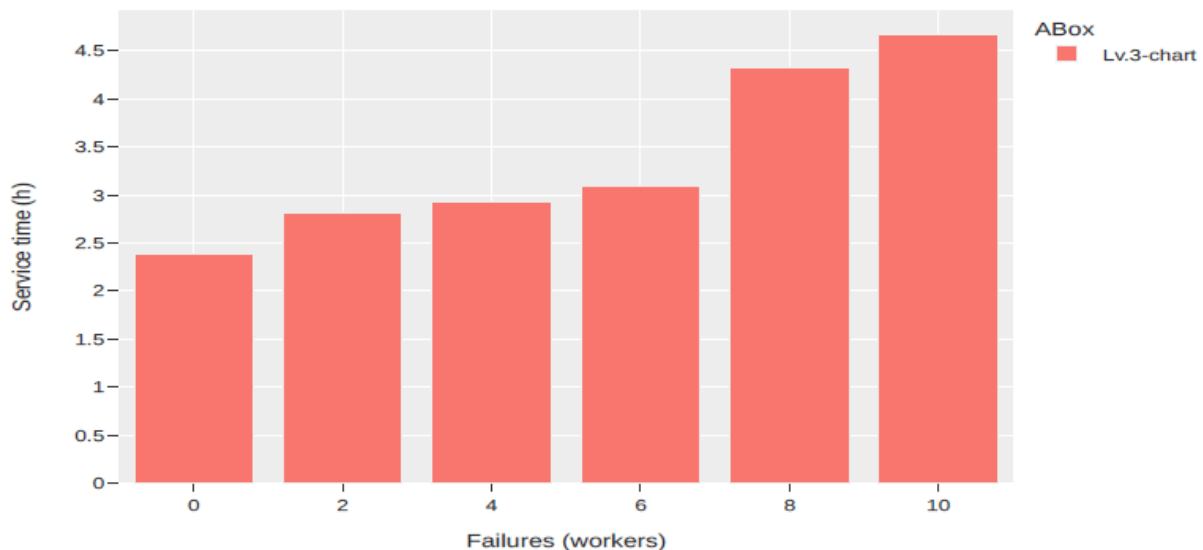


Figura 22. Tiempos de servicio cuando los trabajadores fallan

La ventaja que presenta el sistema Xelhua es que permite integrar nuevas estructuras ABox de acuerdo a las necesidades del usuario. Por ejemplo, si se necesitan otros algoritmos, basta con añadir a la solución ya generada una nueva estructura ABox que contenga el proceso deseado. Al volver a ejecutar la solución, ésta se ejecutará desde el punto en el que se añadió la nueva estructura ABox. Lo anterior debido a que los resultados ya generados quedan indexados y pueden ser consultados y reutilizados. De esta forma, un usuario puede decidir qué segmento de datos quiere seguir procesando, hacerlo gradualmente, y rediseñar la solución en función de lo que indiquen los resultados, o incluso, especificando en la estructura ABox qué proceso aplicar a continuación en función de los resultados.

Manual de usuario del sistema Xelhua

El sistema Xelhua está basado en la tecnología de contenedores, su configuración y despliegue toma como base la plataforma Docker³⁴. Esta sección presenta la información básica para el correcto despliegue del sistema Xelhua en un ambiente local.

Despliegue local del sistema Xelhua

El sistema Xelhua en línea permite interactuar con las diferentes herramientas de análisis que proporciona a través de una infraestructura predeterminada. Sin embargo, también puede configurarse y desplegarse de manera local a través del proyecto alojado en la plataforma GitHub³⁵. Algunas ventajas de la implementación local del sistema Xelhua son las siguientes:

- Creación de un repositorio propio.
- Despliegue de servicios en una infraestructura a la medida (computadora personal, servidores, o servicios en la nube).
- Construcción de soluciones dinámicas basadas en una configuración personalizada.
- Control sobre la administración de los recursos utilizados por el sistema Xelhua.

Configuración local del sistema Xelhua

La correcta ejecución y despliegue del sistema Xelhua requiere de los siguientes recursos de software:

- Docker. La plataforma Docker es utilizada para la administración de los contenedores, así como para la coordinación de los servicios ofrecidos por las diferentes estructuras ABox.
- Docker-compose. Es un módulo de la plataforma Docker utilizado para ejecutar configuraciones basadas en el lenguaje YAML³⁶. Docker-compose permite crear y ejecutar uno o varios servicios a partir de un solo comando (encargado de ejecutar el contenido del archivo de configuración).

Instalación

El proyecto Xelhua está dividido en diferentes carpetas, a continuación, se detalla el contenido de cada una de ellas:

- **AG/** - contiene los scripts o instrucciones de las API Gateway.
- **BuildingBlocks/** - Carpeta con todos los bloques de construcción, es decir, los módulos que abstraen las aplicaciones y les dotan de interfaces de comunicación y gestores de datos.

³⁴ <https://www.docker.com/>

³⁵ <https://github.com/ArmandoBarron/xel>

³⁶ Es un lenguaje de serialización de datos comúnmente utilizado para definir el contenido de los archivos de configuración.

- **IAG/** - contiene los scripts o instrucciones de los Internal API Gateway.
- **localdata/** - Carpeta predeterminada donde se almacenan los resultados de la aplicación y los conjuntos de datos del usuario.
- **NonFunctional/** - directorio con scripts o instrucciones de requerimientos no funcionales.

La ejecución del sistema Xelhua requiere de la instalación previa de los componentes de software Docker y Docker-compose. No se hace mención específica sobre el sistema operativo debido a que Docker es una plataforma que puede ejecutarse en la gran mayoría de sistemas operativos, sin embargo, a modo de ilustrar la ejecución del sistema Xelhua las instrucciones que se muestran a continuación están basadas en una terminal del sistema operativo Linux. Las instrucciones (remarcadas en color gris y que comienzan con el signo \$) ejemplifican cada paso para la correcta ejecución del sistema Xelhua. A continuación, se presentan las instrucciones para ejecutar localmente el sistema Xelhua.

- Clonar el proyecto del sistema Xelhua.
 - a. `$ git clone https://github.com/ArmandoBarron/xel`
- Acceder al directorio del proyecto
 - a. `$ cd xel`
- Ejecutar el comando de Docker-compose para desplegar los servicios definidos en el archivo de configuración.
 - a. `$ docker-compose -f Xel.yml up -d`
- Acceder a la interfaz de usuario. Una vez finalizado el despliegue de los servicios del sistema Xelhua, abrir el navegador predilecto y acceder a la siguiente dirección: <http://localhost:8080/> para visualizar la interfaz de usuario del sistema Xelhua.

Una de las ventajas del sistema Xelhua es la posibilidad de modificar cualquiera de los componentes que lo integran. En caso de que alguno de los servicios sea modificado, la siguiente instrucción permite reconstruir las imágenes definidas en el archivo de configuración:

- `$./build`

Despliegue de servicios

Todos los módulos de Xelhua se encuentran encapsulados mediante la tecnología de contenedores virtuales. Para poder desplegar los módulos es necesario adquirir las imágenes de contenedor, lo cual se puede hacer de 2 formas:

1. Mediante la construcción de las imágenes utilizando el código fuente. Para ello se ejecuta el siguiente comando:


```
$ ./build.sh
```
2. Mediante la descarga de las imágenes publicadas en el repositorio de Docker-hub. Para ello se utiliza el siguiente comando:


```
$ ./pull.sh
```

No es necesario descargar todas las imágenes, por lo que se pueden comentar dentro del script aquellas imágenes que no se deseen.

Finalmente, se ejecuta el siguiente comando para desplegar los contenedores:

```
$ docker-compose -f Xel.yml up -d
```

Diseño y ejecución de soluciones

Una vez desplegada la malla de servicios, ésta puede utilizarse para crear diferentes soluciones para problemas de big data. Estas soluciones se diseñan a partir de instrucciones definidas como una estructura de árbol, por ejemplo:

```
{
  "DAG": "[
    {id:c1-data_clean,
     service:cleanning,
     childrens:
     [
       {
         id:c2-s-imputation,
         service:imputation,
         childrens:
         [
           {
             id:c3-s-split-and-join,
             service:join_split,
             childrens:
             [
               {
                 id:c4-regression_serv,
                 service:regression,
                 childrens:[],
                 actions:[LINEARS],
                 params:{LINEARS:{actions:[LINEARS],
                  var_x:year,
                  list_var_x:0,
                  var_y:[PM10],
                  filter_column:,
                  filter_value:,
                  alpha:0.05,
                  SAVE_DATA:true}}
               },
               {
                 id:c5-regression_serv,
                 service:regression,
                 childrens:[],
                 actions:[LINEARS],
                 params:{LINEARS:{actions:[LINEARS],
                  var_x:year,
                  list_var_x:0,
                  var_y:[PM10],
```

```

        filter_column:,
        filter_value:,
        alpha:0.05,
        SAVE_DATA:true}}
    }
],
actions:[SCOLUMN],
params:{SCOLUMN:{actions:[SCOLUMN],
    column:FECHA,
    method:DT,
    date_format:%m/%d/%Y,
    split_value:/,
    columns:,
    separator:,column_name:,
    SAVE_DATA:true}}}
}
],
actions:[DROP],
params:{DROP:{columns:[CO,NO,NO2,NOX,O3,PM10,PM25,PMCO,S02],
    actions:[DROP],
    imputation_type:,
    strategy:,
    groupby:0,n_neighbors:2,
    fill_value:,
    method:any,n_na:0,
    to_drop:0,
    SAVE_DATA:true}}}
}
],
actions:[CLEAN],
params:{CLEAN:{actions:[CLEAN],columns:[CO,NO,NO2,NOX,O3,PM10,PM25,PMCO,
S02],
    outliers_detection:ZS,
    method:,
    encoding_method:,
    n_standard_desviations:2,
    min_range:1,
    max_range:1,
    norm_method:,
    list_values:-99,
    replace_with:,
    label_column:,
    list_labels:,
    SAVE_DATA:true}}}
}
],
"data_map": {
    "data": {

```

```

        "catalog": "Examples",
        "filename": "contaminantes_1986-2022.csv",
        "token_user":
"300d03efcac4bab98d04af639dba337d5350ebd955fa39b4912376f7715a9fc6"
    },
    "type": "LAKE"
},
"auth": {
    "user":
"300d03efcac4bab98d04af639dba337d5350ebd955fa39b4912376f7715a9fc6",
    "workspace": "Default"
},
"alias": "Regresion de contaminantes - RAMA",
"token_solution":
"3bf71a4c0e78bd6545da6dadac067a16c6113ec2610a600147c5201f49094f57"
}

```

DAG contiene el gráfico con la configuración y parámetros de los servicios que procesarán una fuente de datos. El grafo debe estar en formato de texto (String) y debe contener los siguientes parámetros:

- **Service:** es el nombre del servicio llamado.
- **Actions:** es un conjunto con los nombres de la aplicación dentro del bloque de construcción declarado en **service**
- **Params:** es un conjunto de parámetros definidos en un formato de valor clave que se usará para que la aplicación realice la transformación de datos.
- **Childrens:** es un conjunto de niños con las instrucciones que se ejecutarán para los servicios después de terminar.
- **Data_map:** contiene un objeto con la información del conjunto de datos a procesar. Siguiendo este ejemplo, el conjunto de datos *contaminantes_1986-2022.csv* pertenece al usuario *300d03efcac4bab98d04af639dba337d5350ebd955fa39b4912376f7715a9fc6* en la carpeta Ejemplos.

El conjunto de datos debe ser previamente subido a la plataforma. A continuación se muestran las formas disponibles hasta esta versión de xel para proporcionar un conjunto de datos

- *Subir un archivo.* El conjunto de datos se puede enviar a la malla como *multipart/form-data* a la ruta */UploadDataset*. Además del archivo, se debe proporcionar información para el almacenamiento en el lago, como el identificador de usuario y el catálogo (o espacio de trabajo) donde se almacenará, esto se realiza de la siguiente manera:

```

{"workspace": "<catalogo>", "user": "<token_de_usuario >"}

```


Una vez que se ha cargado el archivo, debe especificarse en el json con las instrucciones. En este caso, el conjunto de datos estará en el lago de datos de xel y se especifica de la siguiente manera:

```
{"data_map":{"data":{"token_user":"<token_de_usuario>","catalog":"<catalogo>","filename":"<nombre_del_archivo>"},"type":"LAKE"}}
```

- **Auth:** contiene el token del usuario y el espacio de trabajo en el que está trabajando. Esta información se utiliza para asignar permisos al grafo que se ejecutará.
- **Alias:** es el nombre que se le asignará al gráfico.
- **token_solution:** es el identificador único del gráfico. Si no se envía ninguno, se asignará uno automáticamente. Si se asigna un ID que ya existe, se sobrescribirá o se utilizará la información existente para comparar el estado de los servicios y los resultados existentes.

Ejecución de soluciones a través del sistema Xelhua

La ejecución de las soluciones puede hacerse de dos formas:

- Utilizando la GUI desplegada en <http://localhost:8080/>.
- Utilizando la aplicación Client.py con el siguiente comando:

```
$ python3 Client.py localhost:25000  
./Examples/Solutions/map_reduce.json ./localdata/EC_mun.csv
```

Donde:

- 127.0.0.1:25000 es la ip del host con el AG.
- ./Examples/Solutions/map_reduce.json es la ruta del directorio donde se encuentra el archivo de configuración JSON.
- ./localdata/EC_mun.csv es el directorio con el archivo o archivos del conjunto de datos (este parámetro es opcional y depende del tipo de operación que se vaya a realizar).

La interfaz gráfica del sistema Xelhua

El sistema Xelhua integra una interfaz web para el diseño y ejecución de un BDServ. Además, la interfaz web es utilizada para recolectar la información de configuración requerida por las estructuras ABox que integran la solución diseñada por el usuario. La interfaz de diseño permite a un usuario crear servicios de big data basado en estructuras ABox.

En la interfaz web de Xelhua, cada estructura ABox se representa gráficamente a través de un *Building block* (BB). Un BB se representa gráficamente como una caja interactiva, esto es, el usuario puede mover de posición la BB y configurar los parámetros de la estructura ABox correspondiente. La figura 23 ilustra la interfaz web de Xelhua. Cada uno de los recuadros negros (parte inferior izquierda de la figura 23) representan un BB que pueden ser integrados al plano de diseño gráfico (recuadro gris de la figura 23). Los BB pueden interconectarse, en el plano de diseño, a partir del uso de flechas. Los BB interconectados pueden formar un pipeline con un BB inicial conectado a la fuente de datos definida por el usuario (por ejemplo,

cualquier base de datos, almacén de datos, archivos estructurados como CVS, JSON, o archivos no estructurados como imágenes o documentos).

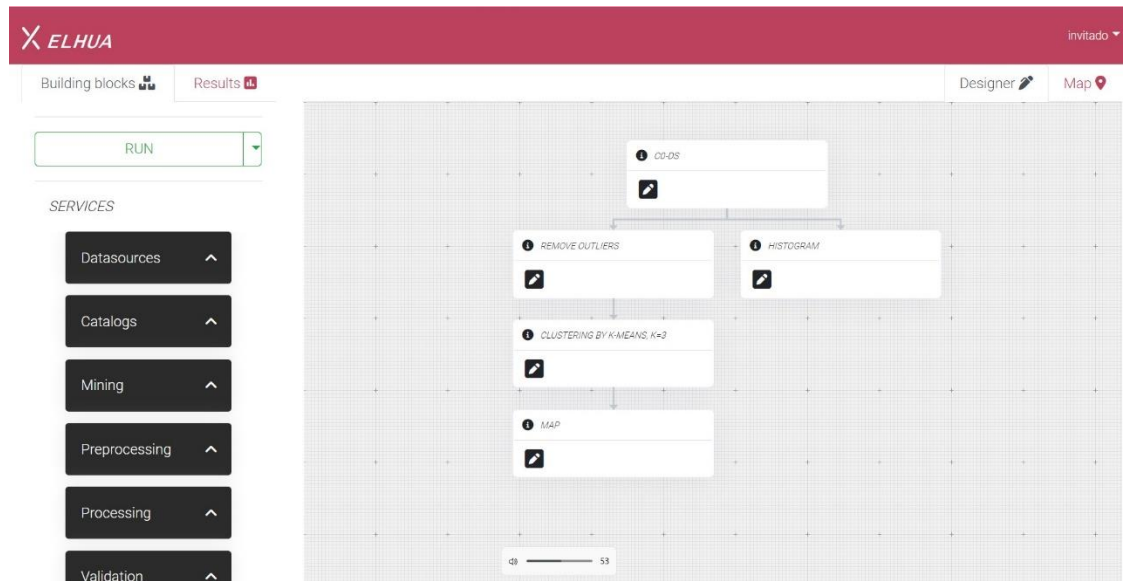


Figura 23. Interfaz gráfica de usuario del sistema Xelhua para el diseño de soluciones de problemas de análisis de datos. En el lado izquierdo se muestra la lista de servicios (cajas). Las cajas pueden arrastrarse al área de diseño (parte derecha).

El usuario de Xelhua puede diseñar una solución para un problema de big data (BDServ), a través de la interfaz web de Xelhua. La interfaz cuenta con un botón de ejecución (recuadro verde con la etiqueta Run, en la figura 23) el cual envía la información gráfica contenida en el área de diseño a la malla de servicios a través de archivos de configuración en formato JSON. El número de archivos de configuración dependerá del número de BB que estén presentes en el área de diseño.

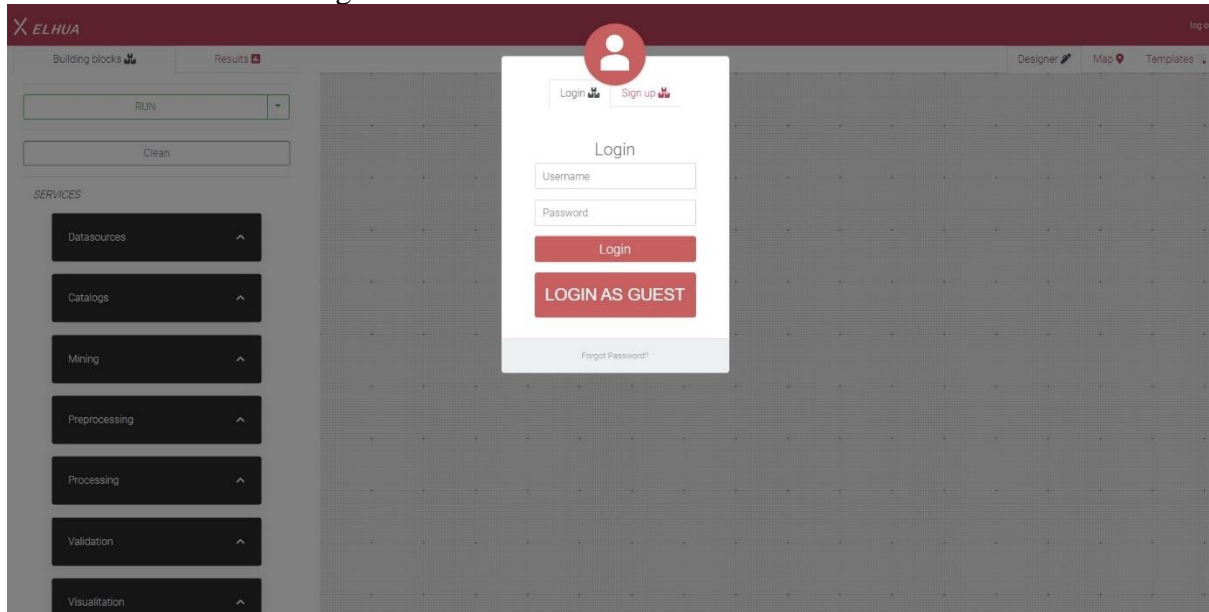
Los parámetros del archivo de configuración JSON, producidas por los BB, son, posteriormente, convertidas en un archivo de configuración YAML³⁷. Estos archivos de configuración son enviados a una plataforma de contenedores de software para el despliegue de los servicios correspondientes a la solución generada por el usuario.

Ejemplo de despliegue de un grafo utilizando la interfaz grafica

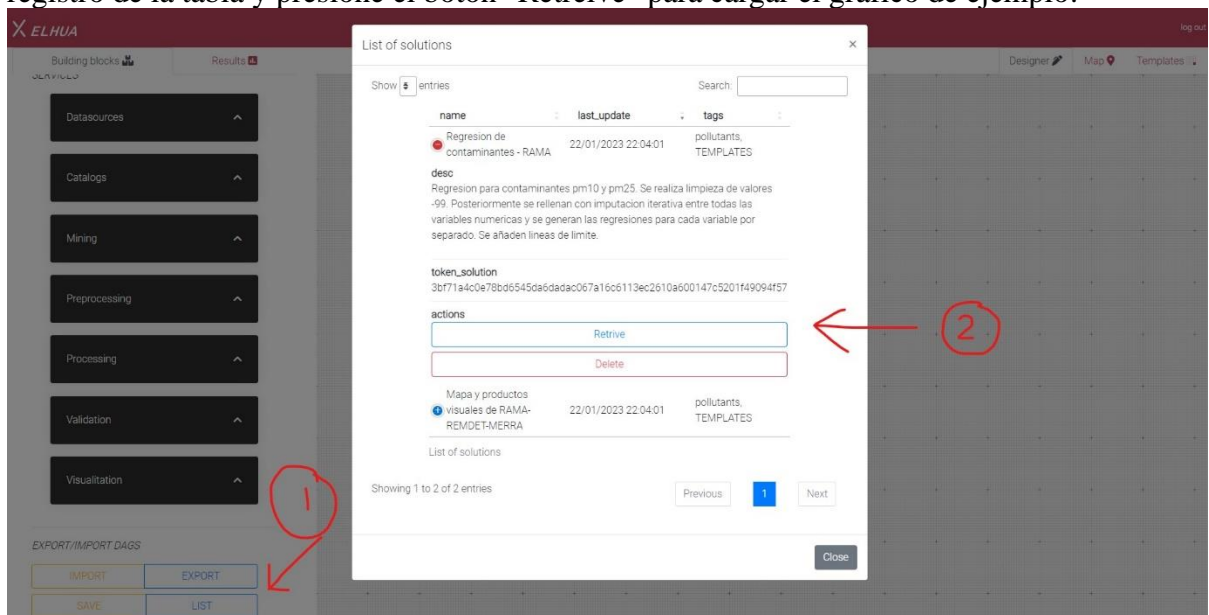
1. Abra la ruta <http://localhost:8080/> en su navegador.

³⁷ YAML es un lenguaje de serialización de datos. Es comúnmente utilizado para generar archivos de configuración

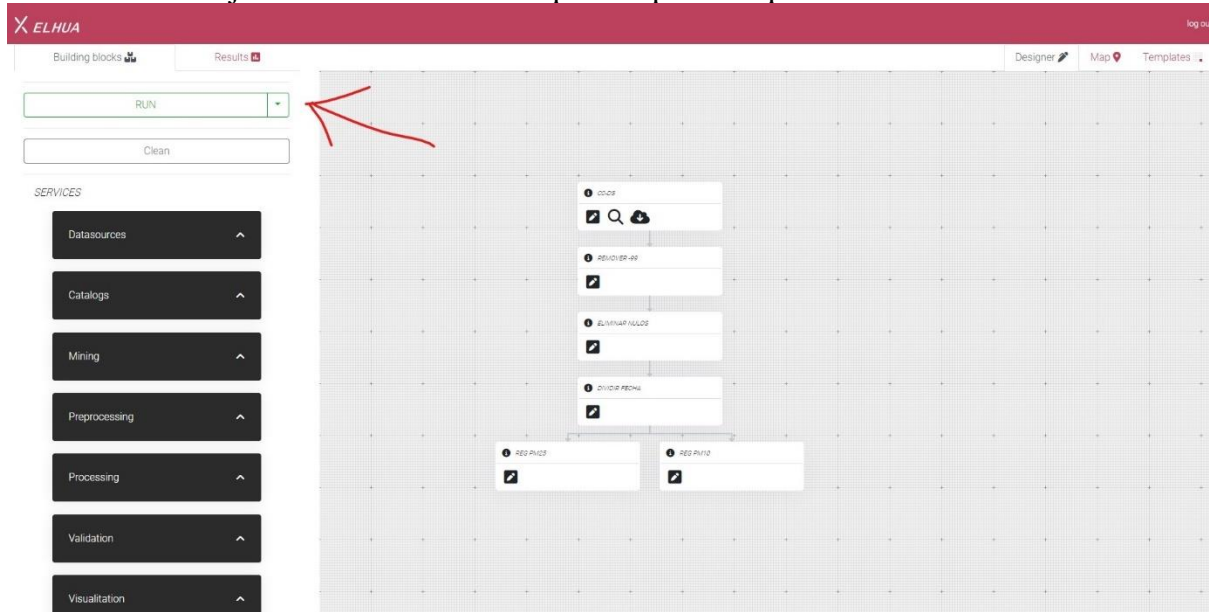
2. Seleccione el botón "ingresar como invitado".



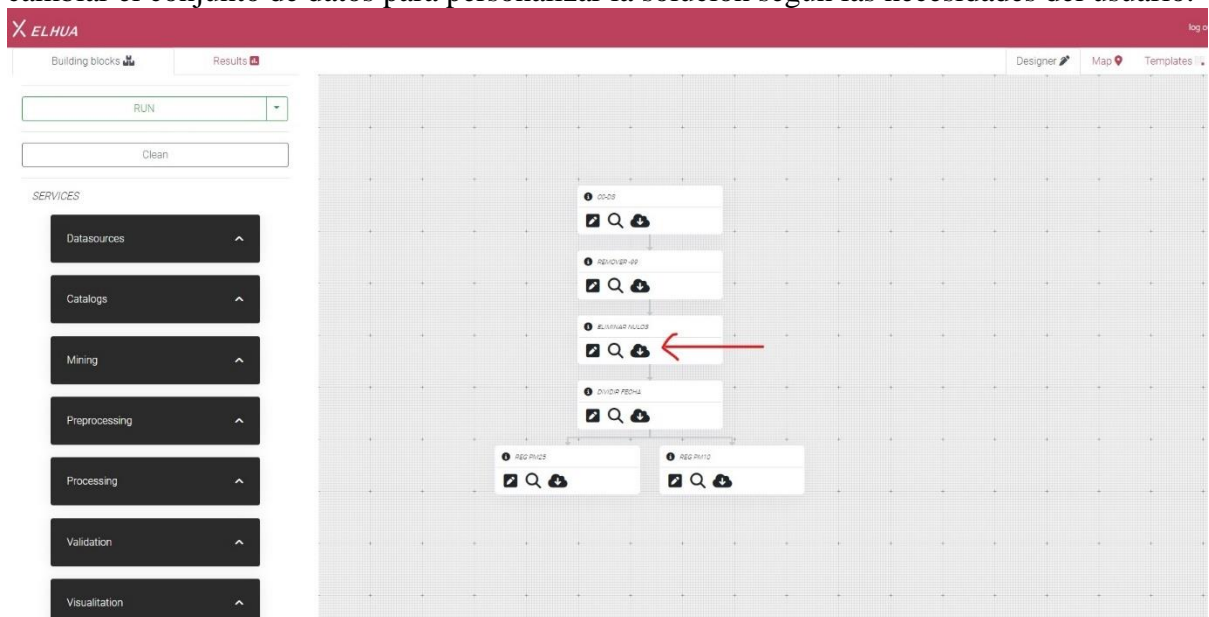
3. Vaya al botón "List" ubicado en la esquina inferior izquierda. Luego seleccione el primer registro de la tabla y presione el botón "Retreive" para cargar el gráfico de ejemplo.



4. Se mostrará como ejemplo una gráfica previamente configurada. Para ejecutarlo, presione el botón verde "Ejecutar" ubicado en la esquina superior izquierda.



5. Al final de la ejecución aparecerán iconos para descargar los resultados de cada una de las BB de la gráfica. Es posible cambiar los parámetros de cada casilla, agregar más casillas o cambiar el conjunto de datos para personalizar la solución según las necesidades del usuario.



Configuraciones avanzadas del sistema

Cambiar la ruta de almacenamiento de resultados

De manera predeterminada, xelhua almacenara los datasets y los resultados de las ejecuciones de los grafos en la carpeta `./localdata/LAKE`. Sin embargo puede cambiar la ruta de almacenamiento definiéndola en la línea 34 y 35 del archivo YML (Xel.yml):

```
coordinator: # COORDINATOR
```

```

image: xel_coordinator:v2.0
build:
  context: ./AG
  dockerfile: ./Dockerfile
ports:
  - "25000:5555"
volumes:
  - ./localdata/LAKE/USERS:/home/app/SUPPLIES/ #Cambiar esta ruta
  - ./localdata/LAKE/SOLUTIONS:/home/app/BACKUPS/ #Cambiar esta ruta
restart: always
networks:
  - service_mesh
environment:
  NETWORK: LocalCluster
  MODE: SERVERLESS
  TOKEN_REQUIERD: "True"
  ALLOW_REGISTER_NEW_USERS: "True"
  INIT_EXAMPLE: "True"
  API_KEY:
  AUTH_HOST: auth
mem_limit: 1024M
mem_reservation: 128M
cpus: 6
deploy:
  placement:
    constraints:
      - node.role == manager

```

Bloquear registro de nuevos usuarios

Por defecto cualquier persona que tenga acceso a la interfaz gráfica de Xelhua es capaz de registrarse como usuario para utilizar la plataforma. Cada usuario es aislado y cuenta con sus propios datasets y áreas de trabajo. Si se desea eliminar la posibilidad a los usuarios de registrarse mediante la interfaz, se debe cambiar el parámetro `ALLOW_REGISTER_NEW_USERS` a `"False"` del contenedor coordinador en el archivo YML. Por otro lado, si se desea eliminar totalmente la necesidad de utilizar usuarios, se puede realizar cambiando la variable `TOKEN_REQUIERD` a `False`.

```

coordinator: # COORDINATOR
  image: xel_coordinator:v2.0
  build:
    context: ./AG
    dockerfile: ./Dockerfile
  ports:
    - "25000:5555"
  volumes:
    - ./localdata/LAKE/USERS:/home/app/SUPPLIES
    - ./localdata/LAKE/SOLUTIONS:/home/app/BACKUPS/

```

```

restart: always
networks:
  - service_mesh
environment:
  NETWORK: LocalCluster
  MODE: SERVERLESS
  TOKEN_REQUIRE: "True" #Cambiar esta variable
  ALLOW_REGISTER_NEW_USERS: #Cambiar esta variable
  INIT_EXAMPLE: "True"
  API_KEY:
  AUTH_HOST: auth
mem_limit: 1024M
mem_reservation: 128M
cpus: 6
deploy:
  placement:
    constraints:

```

Administrar base de datos de usuarios

Para la administración de los usuarios directamente a la base de datos, es necesario conectarse desde la máquina host al contenedor. Esto se realiza mediante el siguiente comando:

```
$ docker exec -it xel_db_auth_1 psql -U postgres
```

Por cuestiones de seguridad, se recomienda cambiar la contraseña de administrador, así como también cambiarla las variables de entorno relacionadas a usuario y contraseña en el YML

```

auth: #Authentication service
  image: auth:v1.0
  build:
    context: ./NonFunctional/auth
    dockerfile: ./DF_Auth
  depends_on:
    - db_auth
  expose:
    - "80"
  networks:
    - service_mesh
  environment:
    DB_USER: postgres #cambiar
    DB_PASSWORD: postgres #Cambiar
    DB_NAME: auth
    DB_HOST: db_auth #cambiar en caso de cambiar la IP de la BD
    DB_PORT: 5432
    AUTH_PORT: 47011

```



```
FRONTEND_PORT: 30004
SERVER_PORT: 30500
```

API KEY de servicios

Para garantizar que solo contenedores conocidos puedan acceder al Gateway, se les proporciona un hash sha256 con el cual pueden validar su identidad con el AG. De esta manera, el AG garantiza el acceso a todos los endpoints disponibles a todos aquellos contenedores que cuenten con este token. Este token es estático y se define en el YML en cada uno de los contenedores. El Módulo encargado de asignar el token es el contenedor llamado *service_creator*.

```
services_creator:
  image: xel_yaml_creator:v2.0
  build:
    context: ./NonFunctional/Services_Creator
    dockerfile: ./Dockerfile
  restart: unless-stopped
  working_dir: /home/app/
  networks:
    - service_mesh
  volumes:
  entrypoint: python3 server.py
  environment:
    FLASK_ENV: development
    HOST_PATH: /home/app/
    IMAGES_REPO_URL: 148.247.201.222:31978/
    IMAGES_REPO: "local"
    API_KEY: 9aa491c85508cfeead30c569c88c8f26e3881792a3f158a323ee9ac6150ab1cd
  deploy:
    placement:
      constraints:
        - node.role == manager
```

Este módulo toma la variable de entorno `API_KEY` y se la asigna a todos los contenedores que sean desplegados, y por lo cual, la que usaran para autenticarse con el AG (coordinador).

En este sentido, el coordinador compara este token con aquel que se le haya asignado en el YML, como se puede ver en la siguiente imagen:

```
coordinator: # COORDINATOR
  image: xel_coordinator:v2.0
  build:
    context: ./AG
```

```

    dockerfile: ./Dockerfile
ports:
volumes:
restart: always
networks:
  - service_mesh
environment:
  NETWORK: LocalCluster
  MODE: SERVERLESS
  TOKEN_REQUIERD: "True"
  ALLOW_REGISTER_NEW_USERS: "True"
  INIT_EXAMPLE: "True"
  API_KEY: 9aa491c85508cfeead30c569c88c8f26e3881792a3f158a323ee9ac6150ab1cd
  AUTH_HOST: auth
mem_limit: 1024M
mem_reservation: 128M
cpus: 6
deploy:
  placement:
    constraints:
      - node.role == manager

```

En este contexto, el coordinador aceptara las peticiones cuyo token sea **9aa491c85508cfeead30c569c88c8f26e3881792a3f158a323ee9ac6150ab1cd**.

Por cuestiones de seguridad, se recomienda cambiar estos tokens.

Añadir nuevas aplicaciones a Xelhua

Xelhua permite añadir mas aplicaciones a su entorno de malla de servicios para ser consumidas por usuarios finales. Para añadir una nueva aplicación esta debe cumplir con los siguientes requisitos:

- Seguir un modelo de procesamiento ETL (Tener una entrada y una salida)
- Ser capaz de ser ejecutada mediante comandos de consola (e.g. Python app.py "hola" "mundo")
- Tener listadas las dependencias de software (y preferentemente también las versiones) que requiere la aplicación para ejecutarse.

Si la aplicación cumple con estos requisitos, es posible añadir la aplicación a Xelhua sin modificar el código fuente. Para ellos se deben de seguir los siguientes pasos:

1) Crear estructura de ABox

Todos los servicios de Xelhua siguen una misma estructura de directorios, siendo de la siguiente manera:

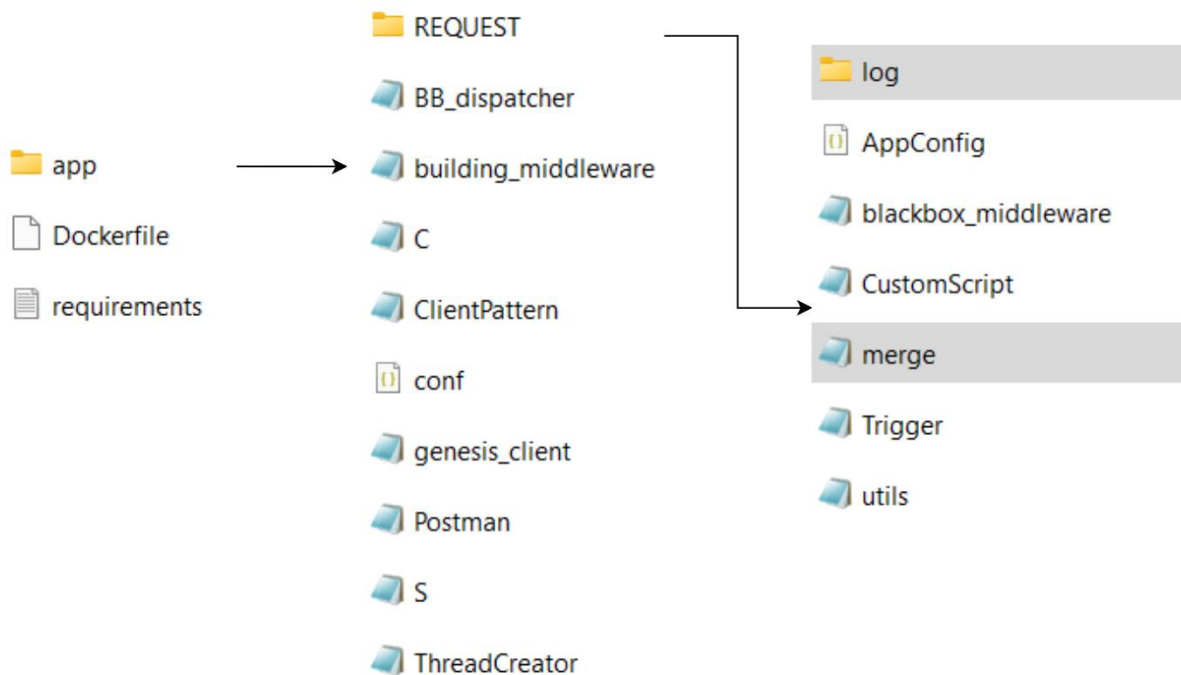


Figura 24. Estructura de directorios de un ABox..

- Dockerfile: Archivo de configuración donde se definen las dependencias de las aplicaciones para generar una imagen de contenedor virtual.
- Requerimientos.txt: lista de requerimientos necesarios para el funcionamiento de los módulos de Xelhua.
- App: es la carpeta contenedora del código del ABox. Esta carpeta contiene los módulos de comunicación, gestores de datos, coordinadores, etc. Estos módulos no se modifican.

Para añadir una aplicación, se debe de crear una carpeta dentro de APP con un nombre en **MAYUSCULAS** y **SIN ESPACIOS NI CARACTERES ESPECIALES**. Se pueden añadir cuantas carpetas se deseen, y cada una puede contener una aplicación distinta.

Cada carpeta debe contener, además de la aplicación a ejecutar, los módulos *Blackbox_middleware.py*, *CustomScript.py*, *Trigger.py*, *Utils.py* y *AppConfig.json*. Salvo en *CustomScript* y *AppConfig*, no se requiere cambiar nada en estos módulos.

- *AppConfig*: Es el archivo de configuración de la aplicación. En este archivo se define la ruta de ejecución de la aplicación. El archivo tiene la siguiente estructura:

```
{
  "NAME_APPLICATION": "Merge",
```

```

"INPUT_DATA_FORMAT":["zip"],
"EXTRACT":{
    "COMPRESS":true
},
"TRANSFORM":{
    "COMMAND": "",
    "CUSTOM_APP":true
},
"LOAD":{
    "OUTPUT_NAMEFILE":"merged.csv",
    "COMPRESS":false,
    "ignore_list":[]
}
}

```

Donde:

- NAME_APPLICATION: Es el nombre que se le dará al servicio
- INPUT_DATA_FORMAT: Un array con las extensiones de datos validas
- EXTRACT.COMPRESS: Si los datos de entrada serán recibidos en un ZIP, este se descomprimirá.
- COMMAND: Indica el comando de ejecución de la aplicación. Para definir el comando se pueden utilizar variables reservadas para indicar la ruta de entrada de los datos (@{SOURCE}), la salida (@{SINK}) y el directorio actual (@{CWD}). Así mismo, Si se requiere que la aplicación reciva el valor de algún parámetro de entrada definida por los usuarios finales, se pueden definir parámetros personalizados con la sintaxis @{<nombre_del_parametro>}. Por ejemplo:

```

{
    "NAME_APPLICATION":"GRAPHICS",
    "INPUT_DATA_FORMAT":["csv"],
    "EXTRACT":{
        "COMPRESS":false
    },
    "TRANSFORM":{
        "COMMAND": "python @{CWD}graph.py @{SOURCE} @{SINK} '@{chart}' '@{histo_type}'
'@{columns}' '@{label_column}' '@{colorscale_column}' '@{column_x}' '@{column_y}'
'@{temporal_column}' '@{subgroup}' '@{size}' '@{if_log_scale}' '@{groups_path}' '@{title}'
'@{column_z}'",
        "CUSTOM_APP":false
    },
    "LOAD":{
        "OUTPUT_NAMEFILE": "",
        "COMPRESS":true,
        "ignore_list":[]
    }
}

```

```
}
}
```

En este caso particular. `python @{{CWD}}graph.py @{{SOURCE}} @{{SINK}} '@{{chart}}' '@{{histo_type}}' '@{{columns}}' '@{{label_column}}' '@{{colorscale_column}}' '@{{column_x}}' '@{{column_y}}' '@{{temporal_column}}' '@{{subgroup}}' '@{{size}}' '@{{if_log_scale}}' '@{{groups_path}}' '@{{title}}' '@{{column_z}}'` El comando ejecutará la aplicación de Python, `graph.py`, recibirá como primer parámetro la ruta de los datos de entrada (`@{{SOURCE}}`), como segundo parámetro la ruta donde depositara los datos (`@{{SINK}}`), y el resto de parámetros son los que requiere la aplicación para funcionar (`'@{{chart}}' '@{{histo_type}}' '@{{columns}}' '@{{label_column}}'`, etc). En tiempo de ejecución, `xelhua` transformara este comando en una sintaxis interpretable por el SO, quedando algo como lo siguiente: `python /home/REQUEST/graph.py /tmp/dshdnyada/data.csv /tmp/du7bmdauj/resultados.csv 'histograma' 'probabilidad' 'edad,tasa_10hab,tasa_100hab' 'clase' ...`

- `CUSTOM_APP`: Bandera que indica si se utilizara un script de Python para la ejecución. En caso de indicar "True", el ABox ejecutara el archivo `CustomScript.py` en lugar de el comando definido en `COMMAND`.
- `OUTPUT_NAMEFILE`: ruta del archivo de resultados que producirá la aplicación. Por defecto se devolverá el primer archivo de la carpeta `@{{SINK}}`.
- `LOAD.COMPRESS`: Si es true, todo el contenido que exista en la carpeta `@{{SINK}}` de empaquetará en un ZIP y se entregará como resultado.

Dentro de la carpeta creada se deben colocar las dependencias locales y el script de ejecución de la aplicación y se debe rellenar el archivo de configuración `AppConfig.json` para cada una de las carpetas que se creen.

Dentro de la carpeta `./Update/` existe un script llamado `InstallNewService.sh`, el cual permite crear la estructura de directorios requerida y copiar los módulos y la propia aplicación a las rutas correspondientes y de manera automática y asistida.

2) Definir dependencias en dockerfile

Posterior a crear la estructura del ABox, es necesario definir las dependencias de la aplicación en el archivo `dockerfile`. Este proceso consiste en añadir las líneas de comandos requeridas para la instalación de los paquetes de software requeridos por la aplicación. Por ejemplo, si la aplicación requiere utilizar funciones del paquete `scikit learn` de Python, entonces se debe de añadir la siguiente línea al `dockerfile`:

```
RUN pip install scikit-learn==1.0.1
```

3) Construir imágenes de contenedor del ABox

Para construir la imagen de contenedor se requiere añadir ejecutar el siguiente comando:

```
docker build -t <nombre_de_la_imagen> <ruta_abosluta_del_ABox>
```

Se recomienda añadir el comando al script ./build.sh.

4) Añadir Abox al despliegue

Una vez se tenga la imagen de contenedor del ABox, se debe añadir al repositorio de servicios de Xelhua para que esta pueda ser utilizada. Para esto se debe de abrir el archivo resources.json ubicado en el directorio raíz del proyecto.











Dentro de resources se debe agregar un registro nuevo como el siguiente:

```
"<nombre_del_servicio>": {  
  "image": "<nombre_de_la_imagen_del_ABOX>",  
  "restart": "always",  
  "networks": ["service_mesh"],  
  "environment": {  
    "TPS_MANAGER": "http://tps_manager:5000",  
    "API_GATEWAY": "coordinator:5555",  
    "NETWORK": "LocalCluster",  
    "SERVICE_NAME": "",  
    "SERVICE_IP": "",  
    "SERVICE_PORT": "80"  
  }  
}
```

En este contexto, solo se debe indicar el nombre por el cual el ABox podrá ser llamado mediante las APIs, y el nombre de la imagen de contenedor que contiene la aplicación.

Añadir nuevas aplicaciones a la interfaz gráfica de Xelhua

Los fomrularios utilizados por la interfaz grafica de Xel se encuentran en la ruta \Frontend\resources\js\myjs\ServicesForms\. En esta ruta se definen los forumarios de cada uno de los ABox disponibles en la plataforma de xel, para añadir uno, se debe modificar y agregar un nuevo registro a alguna de las secciones (si se desea añadir una nueva sección, hay que definirla en el archivo 0_config_sections.json).

 0_config_sections	3/14/2022 6:12 PM	JavaScript Source File	1 KB
 catalogs	10/7/2022 12:24 PM	JavaScript Source File	6 KB
 datasource	10/7/2022 12:25 PM	JavaScript Source File	10 KB
 mining	10/7/2022 12:26 PM	JavaScript Source File	30 KB
 preprocessing	10/7/2022 1:23 PM	JavaScript Source File	37 KB
 procesing	10/7/2022 12:36 PM	JavaScript Source File	30 KB
 templates	7/19/2022 4:03 PM	JavaScript Source File	3 KB
 tools	10/7/2022 12:39 PM	JavaScript Source File	10 KB
 validation	10/7/2022 12:40 PM	JavaScript Source File	3 KB
 visualization	11/14/2022 4:23 PM	JavaScript Source File	27 KB

El registro a añadir debe estar conformado de la siguiente forma:

```
ServicesArr.push(
  {
    id: "<id_base_para_generar_ids>",
    section:SECTION,
    process_tag:"<tag_libre>",
    valid_datatypes:{input:["CSV"],output:["CSV"]},
    name: "<nombre_del_servicio_definido_en_resources.json>",
    desc: `<descripcion breve del servicio>`,
    columns:{
      default: [],
      parent: []
    },
    params: { # lista de parametros definidos en AppConfig.json
      columns: [],
      method: "",
      actions: [],
      SAVE_DATA:true
    },
    html: `` #html del formulario
  }
)
```

En este contexto, el desarrollador debe diseñar el formulario en base a las necesidades de la aplicación que desee declarar. Así mismo, debe declarar los parámetros que se enviarán al servicio y cuyos cuales fueron definidos en el archivo *AppConfig.json*.

Conclusiones y estatus del sistema Xelhua

El sistema Xelhua es una plataforma de big data agnóstica a la nube para la construcción guiada por el diseño de servicios de ciencia de datos de alta disponibilidad y con tolerancia a fallos. En este trabajo se propusieron nuevos modelos y esquemas para hacer frente a tres cuestiones principales: (i) evitar la dependencia con los proveedores o infraestructuras de big data con la construcción de un nuevo modelo de procesamiento basado en el modelo ETL recursivo para convertir automáticamente los diseños de pipeline en estructuras de software agnósticas a la infraestructura, (ii) la gestión de datos para facilitar el cómputo agnóstico a través de un nuevo modelo de orquestación que gestione, de forma transparente, la entrega de datos a lo largo de las etapas de los sistemas de ciencia de datos, y (iii) enmascarar de forma transparente los fallos del servicio, como las interrupciones de la nube por falta de disponibilidad y la indisponibilidad de las aplicaciones o los datos, lo que se consigue a través de un modelo descentralizado de datos basado en la infraestructura como código.

Se implementó un prototipo basado en el sistema Xelhua que permite crear servicios de analítica para problemas de big data. Se desarrollaron casos de uso para evaluar el desempeño del sistema en dominios como: procesamiento de documentos, análisis de sentimientos y agrupamiento. La evaluación mostró la eficacia y flexibilidad del sistema Xelhua para crear múltiples tipos de servicios de alta disponibilidad, evitando los efectos secundarios asociados a los escenarios de bloqueo de proveedores. La evaluación experimental reveló la eficacia de los servicios de big data de alta disponibilidad contruidos para soportar escenarios como las interrupciones de la nube y la indisponibilidad de las aplicaciones o los datos.

Actualmente, prototipos del sistema Xelhua se utilizan para crear servicios de analítica en problemas de big data. Uno de estos prototipos es utilizado por el Instituto Mexicano de Psiquiatría, de la Universidad Autónoma de México (UNAM), para el procesamiento de datos relacionados a temas como el suicidio, marginalidad, pobreza, drogadicción, entre otros. Otro prototipo es utilizado para crear el observatorio mexicano del cáncer, cuyos resultados se darán a conocer en un futuro próximo.

Referencias

- [1] D. Trabucchi y T. Buganza, «Data-driven innovation: Switching the perspective on big data,» *European Journal of Innovation Management*, 2019.
- [2] I. FutureScape, «Worldwide digital transformation 2022 predictions.,» 2021. [En línea]. Available: <https://goto.webcasts.com/starthere.jsp>.
- [3] S. Fitzgerald, B. Parker, S. Ng, P. Carter, L. Dunbrack, L. Hand, S. Findling y M. Versace, «Idc futurescape: Worldwide digital transformation,» *IDC FutureScape*, 2017.
- [4] J. Ejarque, R. Badia, L. Albertin, G. Aloisio, E. Baglione, Y. Baglione, S. Boschert, J. Berlin, A. D'Anca y D. Elia, «Enabling dynamic and intelligent workflows for hpc, data analytics, and ai convergence.,» *Future generation computer systems* 134, pp. 414-429, 2022.
- [5] S. Yang, «A novel study on deep learning framework to predict and analyze the financial time series information,» *Future Generation Computer Systems* 125, pp. 812-819, 2021.
- [6] M. Salman y B. Wang, «Near-optimal responsive traffic engineering in software defined networks based on deep learning,» *Future Generation Computer Systems* 135, pp. 172-180, 2022.
- [7] L. Patrice, «Grobid.,» 2021. [En línea]. Available: <https://github.com/kermitt2/grobid>.
- [8] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving y M. Isard, «Tensorflow: A system for large-scale machine learning,» *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, pp. 265-283, 2016.
- [9] J. Cid-Fuentes, S. Sola, P. Alvarez, A. Castro-Ginard y R. Badia, «dislib: Large scale high performance machine learning in python,» *15th International Conference on eScience (eScience), IEEE*, pp. 96-105, 2019.
- [10] C. Ardagna, P. Ceravolo y E. Damiani, «ig data analytics asa-service: Issues and challenges,» *IEEE international conference on big data (big data), IEEE*, pp. 3638-3644, 2016.
- [11] R. Wu, L. Huang, P. Yu y H. Zhou, «Edaws: A distributed framework with efficient data analytics workspace towards discriminative services for critical infrastructures,» *Future Generation Computer Systems* 81, pp. 78-93, 2018.
- [12] E. Amazon, «Amazon web services,» 11 2012. [En línea]. Available: <http://aws.amazon.com/es/ec2/>.
- [13] D. Chappell, *Introducing the azure services platform*, 2008.
- [14] S. Challita, F. Zalila, C. Gourdin y P. Merle, «A precise model for google cloud platform,» *IEEE International Conference on Cloud Engineering (IC2E)*, pp. 177-183, 2018.

- [15] S. Celis y D. Musicant, «Weka-parallel: machine learning in parallel,» *Carleton College, CS TR, Citeseer*, 2002.
- [16] P. Lopez, A. Arjona, J. Sampé, A. Slominski y L. Villard, «Triggerflow: trigger-based orchestration of serverless workflows,» *Proceedings of the 14th ACM International Conference on Distributed and Event-based Systems*, pp. 3-14, 2020.
- [17] R. Lisle, «Google earth: a new geological resource,» *Geology today* 22, pp. 29-32, 2006.
- [18] X. Liu, Y. Liu, H. Song y A. Liu, «Big data orchestration as a service network,» *IEEE Communications Magazine* 55, pp. 94-101, 2017.
- [19] T. Krishna, H. Kwon, A. Parashar, M. Pellauer y A. Samajdar, «Data orchestration in deep learning accelerators,» *Synthesis Lectures on Computer Architecture* 15, pp. 1-164, 2020.
- [20] L. Vaquero, F. Cuadrado, Y. Elkhatab, J. Bernal-Bernabe, S. Srirama y M. Zhani, «Research challenges in nextgen service orchestration,» *Future Generation Computer Systems* 90, pp. 20-38, 2019.
- [21] W. Li, Y. Lemieux, J. Gao, Z. Zhao y Y. Han, «Service mesh: Challenges, state of the art, and future research opportunities,» *IEEE International Conference on Service-Oriented System Engineering (SOSE)*, IEEE, pp. 122-1225, 2019.
- [22] D. Tarboton, R. Idaszak, J. Horsburgh, J. Heard, D. Ames, J. Goodall, L. Band, V. Merwade, A. Couch y J. Arrigo, «Hydroshare: advancing collaboration through hydrologic data and model sharing,» *7th International congress on environmental modelling and software*, 2014.
- [23] M. Wilkinson, M. Dumontier, I. Aalbersberg, G. Appleton, M. Axton, A. Baak, N. Blomberg, J. Boiten, L. da Silva Santos y P. Bourne, «The fair guiding principles for scientific data management and stewardship,» *Scientific data* 3, pp. 1-9, 2016.
- [24] Z. Sun, L. Di y J. Gaigalas, «Suis: Simplify the use of geospatial web services in environmental modelling,» *Environmental Modelling & Software* 119, pp. 228-241, 2019.
- [25] M. Liu, L. Pan y S. Liu, «Effeclouds: A cost-effective cloud-of-clouds framework for two-tier storage,» *Future Generation Computer Systems*, pp. 33-49, 2022.
- [26] R. Gracia-Tinedo, C. Cotes, E. Zamora-Gómez, G. Ortiz, A. Moreno-Martinez, M. Sanchez-Artigas, P. García-López, R. Sánchez, A. Gómez y A. Illana, «Giving wings to your data: A first experience of personal cloud interoperability,» *Future Generation Computer Systems* 78, pp. 1055-1070, 2018.
- [27] Y. Aldwyen y R. Sinnott, «Latency-aware failover strategies for containerized web applications in distributed clouds,» *Future Generation Computer Systems* 101, pp. 1081-1095, 2019.
- [28] Y. Xia, X. Xu, W. Wang, X. He, W. Wu y X. Fang, «Recovering cloud services using hybrid clouds under power outage,» *International cloud services using hybrid clouds under power outage*, pp. 496-503, 2020.

- [29] M. Bansal, M. Nanda y M. Husain, «Security and privacy aspects for internet of things (iot),» *6th International Conference on Inventive Computation Technologies (ICICT)*, IEEE, pp. 199-204, 2021.
- [30] P. Wang, C. Zhao, W. Liu, Z. Chen y Z. Zhang, «Optimizing data placement for cost effective and high available multi-cloud storage,» *Computing and Informatics* 39, pp. 51-82, 2020.
- [31] N. Bouzerzour, S. Ghazouani y Y. Slimani, «A survey on the service interoperability in cloud computing: Client-centric and providercentric perspectives,» *Software: Practice and Experience* 50, pp. 1025-1060, 2020.
- [32] J. Opara-Martins, R. Sahandi y F. Tian, «Critical analysis of vendor lock-in and its impact on cloud computing migration: a business perspective,» *Journal of Cloud Computing* 5, pp. 1-18, 2016.
- [33] A. Khajeh-Hosseini, D. Greenwood y I. Sommerville, «Cloud migration: A case study of migrating an enterprise it system to iaas,» *IEEE 3rd International Conference on cloud computing*, IEEE, pp. 450-457, 2010.
- [34] J. Adzic, V. Fiore y L. Sisto, «Extraction, Transformation, and Loading Processes,» R. Wrembel, & C. Koncilia (Ed.), *Data Warehouses and OLAP: Concepts, Architectures and Solutions*, pp. 88-110, 2007.
- [35] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann y I. Witten, «The weka data mining software: an update,» *ACM SIGKDD explorations newsletter* 11, pp. 10-18, 2009.
- [36] M. Zaharia, A. Chen, A. Davidson, A. Ghodsi, S. Hong, A. Konwinski, S. Murching, T. Nykodym, P. Ogilvie y M. Parkhe, «Accelerating the machine learning lifecycle with mlflow,» *IEEE Data Eng. Bull.* 41, pp. 39-45, 2018.
- [37] D. Talia, P. Trunfio y O. Verta, «Weka4ws: a wsrf-enabled weka toolkit for distributed data mining on grids,» *European Conference on Principles of Data Mining and Knowledge Discovery*, Springer, pp. 309-320, 2005.
- [38] S. Vaucher, R. Pires, P. Felber, M. Pasin, V. Schiavoni and C. Fetzer, "SGX-Aware container orchestration for heterogeneous clusters," in *Proc. IEEE 38th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, pp. 730-741, July 2018.
- [39] J. Adzic, V. Fiore y L. Sisto, «Extraction, Transformation, and Loading Processes,» In R. Wrembel, & C. Koncilia (Ed.), *Data Warehouses and OLAP: Concepts, Architectures and Solutions*, pp. 88-110, 2007.
- [40] N. R. May, H. W. Schmidt y I. E. Thomas, «Service Redundancy Strategies in Service-Oriented Architectures,» *35th Euromicro Conference on Software Engineering and Advanced Applications*, pp. 383-387, 2009.
- [41] L. Lamport, «The part-time parliament,» *Concurrency: the Works of Leslie Lamport.*, pp. 277-317, 2019.
- [42] A. S. Imran, S. M. Daudpota, Z. Kastrati y R. Batra, «Cross-cultural polarity and emotion detection using sentiment analysis and deep learning on COVID-19 related tweets,» *IEEE Access*, vol. 8, pp. 181074-181090, 2020.
- [43] L. e. a. Lamport, *Paxos made simple*, *ACM Sigact News* 32, 2001, pp. 18-25.

- [44] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek y H. Balakrishnan, «Chord: A scalable peer-to-peer lookup service for internet applications,» *ACM SIGCOMM Computer Communication Review* 31, pp. 149-160, 2001.
- [45] A. J. Barron-Lugo, J. L. Gonzalez-Compean, I. Lopez-Arevalo, J. Carretero y J. L. Martinez-Rodriguez, «Xel: A cloud-agnostic big data platform for the design-driven building of high-availability data science services,» 2023.
- [46] H. Ramchoun, M. J. Idrissi, Y. Ghanou y M. Ettaouil, «Multilayer perceptron: Architecture optimization and training,» *Int. J. Interact. Multim. Artif. Intell.* 4, pp. 26-30, 2016.
- [47] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot y E. Duchesnay, «Scikit-learn: Machine learning in Python,» *Journal of Machine Learning Research* 12, pp. 2825-2830, 2011.
- [48] N. Eliguzel, C. Cetinkaya y T. Dereli, «Comparison of different machine learning techniques on location extraction by utilizing geotagged tweets: A case study,» *Adv. Eng. Informatics* 46, 2020.
- [49] D. M. Blei, A. Y. Ng y M. I. Jordan, «Latent dirichlet allocation,» *J. Mach. Learn. Res.* 3, pp. 993-1022, 2003.
- [50] P. J. Rousseeuw, «Silhouettes: A graphical aid to the interpretation and validation of cluster analysis,» *Journal of Computational and Applied Mathematics*, vol. 20, pp. 53-65, 1987.
- [51] T. Calinski y J. Harabasz, «A dendrite method for cluster analysis,» *Communications in Statistics*, vol. 3, pp. 1-27, 1974.
- [52] M. Roder, A. Both y A. Hinneburg, «Exploring the space of topic coherence measures,» *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining*, pp. 399-408, 2015.
- [53] E. Tejedor, Y. Becerra, G. Alomar, A. Queralt, R. Badia, J. Torres, T. Cortes y J. Labarta, «Pycompss: Parallel computational workflows in Python,» *The International Journal of High Performance Computing Applications* 31, pp. 66-82, 2017.