UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

BIG DATA TECHNOLOGIES FOR HIGH PERFORMANCE COMPUTING

Trabajo Fin de Grado

Realizado en

Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona

Universitat Politècnica de Catalunya

por

Miquel Martínez Blanco

En cumplimiento parcial de los requisitos para

Grau en Enginyeria de Tecnologies i Serveis de Telecomunicació

Tutor: Jorge Mata Diaz

Barcelona, Junio 2020

Abstract

Hecuba is a tool written in Python and C++ developed in the Barcelona Supercomputing Center (BSC), it allows to simplify the process of reading and writing in Cassandra databases. The objective is to integrate Hecuba into Dislib - another Python library developed in the BSC that allows the treatment and manipulation of large amounts of data using the COMPSs framework, which allows the development and execution of parallel applications in distributed computing architectures in a simple way. This thesis has been carried out within the European project I-BiDaaS, which was created with the aim of providing a self-service platform for Big Data analysis. A use case of this integration has been carried out to demonstrate its powerful capacity, it has consisted on a data analysis of a bank transfer dataset, provided by CaixaBank, a collaborating company in the I-BiDaaS project, with the aim of finding possible anomalies.

Resum

Hecuba és una eina escrita en Python i C++ desenvolupada al Barcelona Supercomputing Center, que permet simplificar el procés de lectura i escriptura en bases de dades Cassandra. El objectiu és integrar Hecuba dins de Dislib, una altra llibreria de Python desenvolupada al BSC que permet el tractament i manipulació de grans quantitats de dades fent ús del framework de COMPSs, que permet desenvolupar i executar aplicacions en paral · lel en arquitectures de càlcul distribuït. Aquesta tesis s'ha dut a terme dins del projecte d'àmbit europeu I-BiDaaS, que neix amb la finalitat de proporcionar una plataforma self-service per al anàlisis Big Data. Com a cas pràctic d'un futur ús que si li donarà a aquesta integració, s'ha dut a terme un anàlisis d'un data set de transferències bancàries, proporcionat per CaixaBank, empresa col · laboradora en el projecte I-BiDaaS, amb el objectiu de trobar possibles anomalies.

Resumen

Hecuba es una herramienta escrita en Python y C++ desarrollada dentro del Barcelona Supercomputing Center, que permite simplificar el proceso de lectura y escritura en bases de datos Cassandra. El objetivo es integrar Hecuba en Dislib, otra librería de Python desarrollada en el BSC que permite el tratamiento y manipulación de grandes cantidades de datos haciendo uso del framework de COMPSs, que permite desarrollar y ejecutar de manera sencilla aplicaciones en paralelo en arquitecturas de cálculo distribuidas. Esta tesis se ha llevado a cabo dentro del proyecto de ámbito europeo I-BiDaaS, que nace con la finalidad de proporcionar una plataforma self-service para el análisis Big Data. Como caso práctico de un futuro uso que se le dará a esta integración, se ha llevado a cabo un análisis de un dataset de transferencias bancarias, proporcionado por CaixaBank, empresa colaboradora en el proyecto I-BiDaaS, con el objetivo de encontrar posibles anomalías.

Agradecimientos

Con la entrega de este proyecto doy por finalizada una etapa muy importante, tras de mi dejo 4 años de mucho trabajo y esfuerzo, pero también grandes momentos y recuerdos que me acompañarán para siempre.

Me gustaría agradecer a la directora del proyecto, Yolanda Becerra, por su ayuda durante el desarrollo de todo el proyecto y a Jorge Mata por su implicación y consejos para el desarrollo de la memoria. Agradecer también a todo el equipo de desarrollo de COMPSs y Dislib por dar respuesta a todas mis dudas, y en especial a Javier Conejero por la ayuda en la búsqueda de los misteriosos bugs.

Para acabar me gustaría agradecer a toda mi familia por todo el apoyo recibido, a mis amigos por hacerme más amenos estos años y a Berta por todos los consejos y estar siempre dispuesta a escucharme y ayudarme.

Historial de revisiones y registro de aprobación

Revisión	**Fecha**	**Motivo**
0	15/02/2020	Creación del documento
1	08/04/2020	Revisión del documento
2	15/06/2020	Revisión del documento

Nombre	**e-mail**
Miquel Martínez	mbmiquel@gmail.com
Jorge Mata Díaz	jmata@entel.upc.edu
Yolanda Becerra	yolanda.becerra@bsc.es

Escrito por: Miquel Martínez Blanco		Revisado y aprobado por:	
Fecha	29/06/2020	Fecha	29/06/2020
Nombre	Miquel Martínez	Nombre	Jorge Mata Díaz
Cargo	Autor del proyecto	Cargo	Supervisor del proyecto

Índice general

Lista de Figuras

Lista de Tablas

Glosario:

Listado de acrónimos utilizados.

BSC	**B**arcelona **S**upercomputing **C**enter
PCA	**P**rincipal **C**omponent **A**nalysis
UUID	**U**niversal **U**nique **Id**entifier
NoSQL	**N**ot **O**nly **SQL**
CQL	**C**assandra **Q**uery **L**anguage

1. Introducción

Esta tesis se ha llevado a cabo en el Barcelona Supercomputing Center (BSC) dentro del equipo "Data-Driven Scientific Computing" del departamento de Computer Sciencies.

En los últimos años la cantidad de datos que produce nuestra sociedad han aumentado exponencialmente, debido en gran parte a la adopción de las tecnologías IoT y la mejora de los servicios de computación en *cloud*. Estas grandes cantidades de datos generados por las interacciones entre humanos y máquinas esconden un potencial muy grande para la mejora de la eficiencia y la reducción de costes, pudiendo esto ayudar a tomar mejores decisiones económicas y sociales.

I-BiDaaS[1] (Industrial-Driven Big Data as a Self-Service Solution), es un proyecto europeo[2] con el identificador: 780787, que nace de la necesidad de democratizar el análisis de Big Data, ya que en la actualidad la mayoría de empresas necesitan contar con perfiles propios expertos en la materia o contratar a consultores. Por ello, el principal objetivo de I-BiDaaS es proporcionar una plataforma self-service para el análisis Big Data, que sea sencilla de utilizar, rápida y accesible de manera remota. La plataforma se ha desarrollado pensando sobretodo en tres áreas críticas: las telecomunicaciones, la banca y la manufacturación. Es por ello que forman parte del consorcio y participan en el proyecto grandes empresas como: Telefónica, CaixaBank y Fiat.

El objetivo de esta tesis es aportar al proyecto I-BiDaaS una capa de almacenamiento y procesado de los datos que integre las librerías de Python, Hecuba [1] y Dislib [2], ambas desarrolladas en el BSC, y utilizadas respectivamente para el almacenamiento en una base de datos Cassandra [3] y el procesado de grandes cantidades de datos. Además esta capa debe ser compatible con el modelo de computación en paralelo, por ello se deberá de configurar el entorno para poder hacer uso del *framework* de computación distribuida COMPSs [4], también desarrollado en el BSC, y de la cual hacen uso Hecuba y Dislib.

Debido al entorno en el que se encontrará funcionando dicha integración y la dificultad de instalación de esta, el despliegue se ha llevado a cabo haciendo uso de contenedores de Docker [5], para facilitar este proceso en todo tipo de entornos y sistemas Linux.

La plataforma I-BiDaaS es un proyecto conjunto entre distintas entidades y que se encuentra actualmente en desarrollo. Como muestra del potencial que podrá tener todo lo desarrollado en esta tesis, se ha llevado a cabo un caso de uso analizando un *dataset* de transacciones bancarias proporcionado por CaixaBank, empresa colaboradora en el proyecto, para la búsqueda de posibles irregularidades.

[1] Página web del proyecto I-BiDaaS: https://www.ibidaas.eu/

[2] Página web europea del proyecto: https://cordis.europa.eu/project/id/780787

Apache Cassandra es un sistema de almacenamiento de tipo no relacional que permite tratar con grandes cantidades de datos de una manera muy eficiente haciendo uso de distintos nodos. De esta manera consigue dar un servicio muy robusto y estable que puede ser redimensionado de manera dinámica en caso de necesidad.

Hecuba se trata de una librería de Python con un conjunto de herramientas que permiten implementar de manera más fácil y eficiente el acceso y escritura de datos guardados en servidores de almacenamiento Big Data bajo plataformas como Apache Cassandra.

COMP Superscalar (COMPSs) consiste en un *framework* para el desarrollo de aplicaciones en sistemas distribuidos, permitiendo al usuario desarrollar ejecuciones con computación en paralelo, abstrayéndolo de las complicaciones que esto implica. Se utilizará su versión para Python, bajo el nombre de PyCOMPSs.

Distributed Computing Library (Dislib) se trata de una librería de Python, construida para ser operada con COMPSs, que implementa algoritmos de Machine Learning y cálculos distribuidos, presentándolos con una interfaz de fácil uso para el usuario. Actualmente tan solo es capaz de leer datos de archivos en formato CSV.

Docker es una herramienta que permite la creación de contenedores, entendiendo por contenedor como una máquina virtual que contiene el software desarrollado y todas las dependencias necesarias, para su posterior uso en otras máquinas.

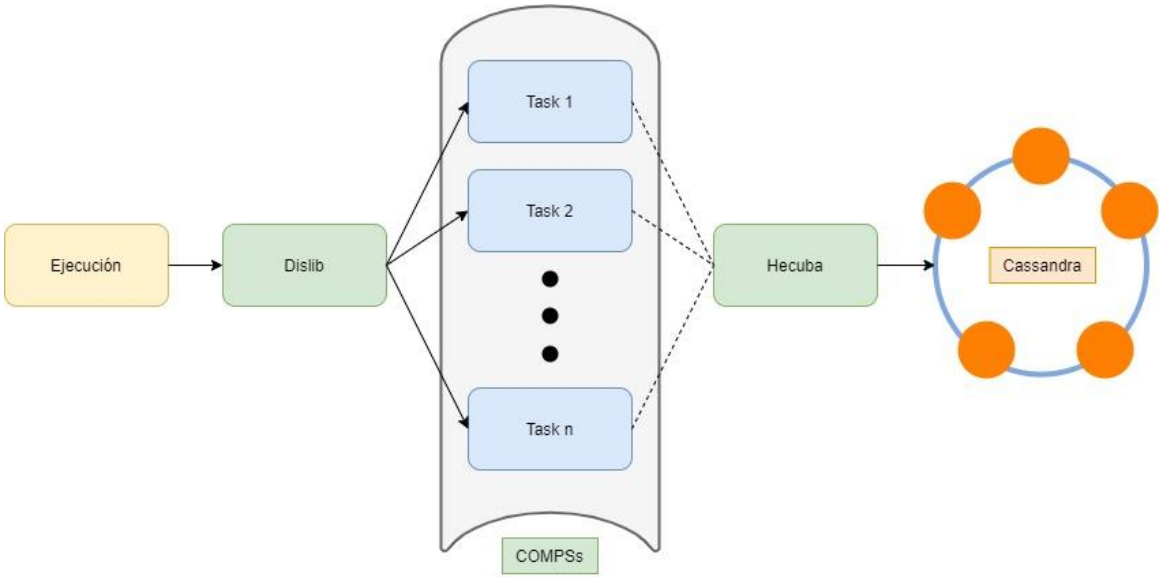

Figura 1. Diagrama de ejecución

1.1. Formulación del problema

En I-BiDaaS se quiere poder hacer uso de Dislib y almacenar la información en una base de datos distribuida y no relacional como Cassandra. En la actualidad Dislib realiza la carga de datos mediante archivos de texto, con lo cual el rendimiento de la aplicación se ve severamente afectado, esto contradice el objetivo de dicha librería, que busca realizar las computaciones de la manera más eficiente posible. Además Dislib utiliza como formato de datos de entrada para todos sus algoritmos un array bidimensional, llamado ds-Array, con el que se implementan distintos métodos para operar en paralelo con COMPSs y que no es compatible con el sistema de almacenamiento de Cassandra. Por lo tanto es necesario recurrir a la interfaz de comunicación que aporta Hecuba y hacer que esta sea compatible con Dislib.

La instalación de las librerías Hecuba, Dislib y COMPSs, es un proceso delicado a la hora de hacer un despliegue del sistema e implementarlo en distintas máquinas, ya que todas las librerías presentan dependencias necesarias antes de su instalación y son librerías que están en constante actualización, hecho que puede provocar incompatibilidades según las versiones. Todo este proceso que se complica todavía más si la instalación se debe llevar a cabo en un sistema de computación distribuido como es el caso.

1.2. Objetivos

El proyecto se encuentra centrado principalmente en cumplir con tres objetivos. Por un lado dotar a Dislib de la capacidad para almacenar datos en Cassandra haciendo uso de la herramienta Hecuba, aportando así valor al proyecto i-BiDasS y también a ambas librerías para otras posibles utilizaciones en otros proyectos en un futuro.

Hecuba es capaz de almacenar en Cassandra ndarrays de la librería Numpy, los cuales implementan arrays multidimensionales que guardan cierta similitud con los usados por la librería de Dislib. Por ello, para conseguir integrar ambas librerías se debe realizar una traducción entre las estructuras de datos utilizadas por ambas librerías, haciendo además este proceso transparente para el usuario. Con esta integración se pretende conseguir que se pueda hacer uso de todos los algoritmos y cálculos distribuidos de Dislib con la potencia de almacenamiento que nos proporciona Hecuba, sin necesidad de que el usuario deba de incorporar código de acceso a la base de datos ya que de ello se encarga Hecuba.

Hoy en día en el ámbito del despliegue de software es habitual hacer uso de herramientas como Docker, que nos permiten instalar de manera muy sencilla en cualquier servidor o

sistema, nuestro software y todas las dependencias necesarias para poder funcionar. Debido a los grandes beneficios derivados de la utilización de Docker uno de los objetivos definidos para esta tesis pasa por realizar una implementación del sistema haciendo uso de contenedores de Docker, permitiendo así una fácil instalación de estas herramientas en todo tipo de máquinas. Una imagen de Docker puede ser subida al repositorio DockerHub[3] de manera que sea accesible para ser descargada e instalada en cualquier ordenador.

Por último, se quiere mostrar la capacidad de análisis que aportaría esta integración en un caso de uso real, mediante el estudio de un dataset de transferencias bancarias proporcionado por CaixaBank, una de las empresas colaboradoras del proyecto I-BiDaaS. Con el objetivo de detectar posibles fraudes o anomalías mediante una clusterización no supervisada en un conjunto de datos de 50.000 transferencias bancarias.

1.3. Requerimientos y especificaciones

Los requerimientos del proyecto son:

- Poder ejecutar los algoritmos que implementa Dislib con datos almacenados de manera física en un *data storage* Apache Cassandra.
- Hacer un despliegue sencillo del proyecto y de todas las dependencias de software necesarias para su correcta ejecución.
- Llevar a cabo un análisis de un caso de uso real, con datos de transferencias bancarias de Caixabank, utilizando la implementación de Hecuba y Dislib desarrollada.

Por lo tanto para hacer frente a los distintos requerimientos las especificaciones técnicas consistirán en:

- Poder hacer frente al acceso a los datos almacenados en Cassandra de manera recurrente y poder crear estructuras de datos complejas, tal y como requiere Dislib, se hará uso de la librería de Hecuba y la clase de datos StorageNumpy que implementa.
- Un despliegue del software integrado con Docker, mediante la creación de un contenedor con el software y todos los requerimientos, que será almacenada en DockerHub. Otra posibilidad es la creación de un archivo Dockerfile en el que se incluyan todos los comandos para la instalación, permitiendo así también al usuario hacer modificaciones al contenedor más fácilmente.

[3] Página web Docker Hub: https://hub.docker.com/

- Llevar a cabo un prepocesado de los datos con el objetivo de cumplir los requisitos de los algoritmos de Machine Learning de Dislib. Además decidir qué tipo de algoritmos se aplicarán a los datos.

1.4. Software implicado

Actualmente tanto Dislib como Hecuba se tratan de dos librerías de Python totalmente funcionales ya desarrolladas por otros equipos del BSC. Estas librerías por separado funcionan y son operativas pero ahora se marca como objetivo hacer que trabajen de manera conjunta para unir lo mejor da cada una.

Dentro de la pipeline del sistema se utilizan distintos software, todos ellos gratuitos y de libre acceso. Para la implementación de algoritmos de Machine Learning se utiliza Dislib, y para la ejecución de estos de manera distribuida el *framework* de COMPSs. Los datos requeridos y generados por todos los procesos se almacenarán en una base de datos Apache Cassandra usando la librería de Hecuba de Python. Para poder hacer el despliegue de todo el proyecto se hará uso de contenedores de Docker.

1.4.1. Apache Cassandra

A nivel de arquitectura Cassandra establece un sistema distribuido donde la información es repartida entre los distintos nodos de un clúster. Para obtener un mayor rendimiento el sistema tiene consistencia eventual, es decir, no se implementan mecanismos rígidos que garanticen que todos los cambios sean vistos por la totalidad de los nodos antes de dar por validos los cambios efectuados. La cantidad de nodos que deben aceptar una modificación se regula según el valor de consistencia que se ha querido configurar, cuanto más alto el nivel menor eficiencia habrá en el sistema pero más seguridad en cuanto a la consistencia de los datos.

Otro de los beneficios de Cassandra es la posibilidad de replicar los datos entre los distintos nodos, de manera que en caso de que uno de ellos fallara no habría una interrupción de los servicios que este prestando la base de datos ya que otro nodo tendría disponible esos datos. La cantidad de nodos que replicaran una misma porción de los datos se configura según el nivel de replicación que se establece en el *keyspace*. Típicamente en otros sistemas de almacenamiento se siguen patrones de maestro y esclavo, pero en el caso de Cassandra se utiliza una arquitectura *Peer to Peer* en las que la escritura se lleva a cabo entre los nodos y son ellos mismos los que se coordinan para sincronizar la copia de los datos, definiendo un nodo como coordinador, además cualquier

nodo puede interactuar con el cliente tanto para escrituras como lecturas y actuar como coordinador.

La capacidad de procesamiento de este tipo de sistemas escala de manera lineal y horizontal con respecto al número de nodos. Entendiendo escalabilidad horizontal como el incremento del rendimiento del sistema simplemente añadiendo más servidores, y que ese incremento se produce de manera lineal con respecto a la cantidad. De manera que doblando el número de nodos disponibles se aumenta el doble la capacidad de computación.

La estructura de almacenamiento de Cassandra se basa en:

- *Keyspaces*, son la unidad más grande dentro de un sistema de almacenamiento de Cassandra. Este contiene a su vez múltiples Column family. Todos los elementos de un mismo *keyspace* tienen definido el mismo factor de replicación.
- *Column family* o *table*. Se trata de un contenedor capaz de agrupar conjuntos ordenados de *rows*, su estructura se asemeja a la de una tabla.
- *Rows*. Formadas por conjuntos ordenados de columnas. Entre distintas *rows* de un mismo *Column family* no tienen por qué tener definidos siempre las mismas columnas ya que Cassandra no lo fuerza.

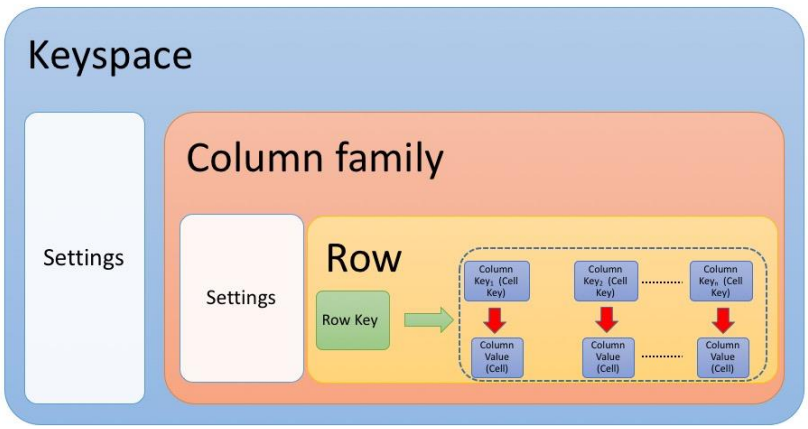

Figura 2. Estructura de almacenamiento de Cassandra

1.4.2. Hecuba

La implementación actual de Hecuba soporta aplicaciones desarrolladas en Python y con almacenamiento de los datos en memoria usando Apache Cassandra, un sistema de administración de bases de datos.

El uso de esta herramienta permite al usuario abstraerse de los procesos de almacenamiento y recuperación de datos persistentes, pudiendo acceder a ellos como si de una variable del entorno de ejecución de Python se tratara, Hecuba se encargará de traducir en tiempo de ejecución el acceso a las variables yéndolas a buscar al sistema de almacenamiento. Además Hecuba también implementa optimizaciones para favorecer la localidad de los datos y reducir el número de interacciones con el sistema de almacenamiento.

Esta librería tiene soporte para almacenar distintos tipos de variables de básicas, colecciones y clases propias.

- Dentro del conjunto de variables básicas se incluyen: str, bool, decimal, float, int, blob, tuple, float, buffer, double y counter.
- Como colecciones soportadas por el sistema se encuentran: dict, set, list, frozenset y los ndarrays de la librería de Numpy.
- Además se implementan tres tipos distintos de clases:
    - Un StorageObj permite la creación de la abstracción más simple de Hecuba, un objeto en el cuál el usuario define dentro una serie de atributos que se almacenaran de manera persistente en el *data storage*. Un StorageObj puede contener en su interior variables básicas, colecciones u otros objetos instancia de las tres clases propias de Hecuba
    - Por otro lado el StorageDict permite la creación de un diccionario de Python y la lógica que acompaña a este tipo de colección.
    - Por último los StorageNumpy implementan objetos que en su interior contienen un Numpy ndarray. Este objeto será trabajado en mayor profundidad para la parte de la integración de Dislib y Hecuba.

Figura 3. Logo de Hecuba

1.4.3. COMPSs

COMPSs permite al usuario desarrollar aplicaciones con computación distribuida de una manera sencilla, dándole una serie de herramientas al desarrollador con las que dimensionar la cantidad de nodos e indicar mediante decoradores que partes del código quiere que sean computadas en paralelo. Los decoradores [6] de Python permiten modificar el comportamiento de una función de una manera sencilla.

COMPSs de manera automática y transparente para el usuario se encargará de gestionar la carga de los datos entre los distintos nodos creados, cada uno de estos nodos reciben el nombre de *workers*. Además el sistema *runtime* de ejecución es capaz de encontrar las distintas dependencias de datos entre tareas y de manera dinámica construye un gráfico de dependencias el cual seguirá a la hora de ejecutarse.

El framework se encuentra desarrollada en Java pero dispone de un binding[4] en Python, que recibe el nombre de PyCOMPSs, y que será utilizado en este proyecto.

1.4.4. Dislib

Distributed Computing Library (Dislib) se trata de una librería de Python, que implementa toda una serie de algoritmos y al estar desarrollada a partir de PyCOMPSs permite al usuario abstraerse de los procesos de paralelización necesarios para cálculos con grandes cantidades de datos. Los tres pilares conceptuales de Dislib son:

- Algoritmos. Gran cantidad de algoritmos ya implementados haciendo uso de las herramientas para computación en paralelo de PyCOMPSs. Los algoritmos han sido inspirados en la librería de Python Scikit-Learn.
- Distributed arrays. Creación de un nuevo de tipo de dato que implementa un array de 2 dimensiones que puede ser operado en paralelo y que sirve como input para los distintos algoritmos implementados. Este será uno de los focos de trabajo durante la integración entre Dislib y Hecuba.
- Manejo de datos. Incluye métodos para tratar datos provenientes de archivos en formato CSV y LibSVM.

[4] Un binding es una adaptación de una biblioteca para ser usada en un lenguaje de programación distinto de aquel en el que ha sido escrita.

2. Estado del arte

A continuación se pasara a hablar sobre cómo se suelen solucionar en la actualidad los problemas enfrentados en este proyecto, y si alguna de las soluciones actuales puede ayudar en el desarrollo de este proyecto.

2.1. Trabajar con bases de datos Cassandra

Tal y como ha sido introducido en apartados anteriores, Cassandra es una base de datos de tipo NoSQL, esta tiene una serie de inconvenientes a la hora de almacenar datos complejos en su interior, siendo necesario la utilización de una herramienta intermedia capaz de gestionar Cassandra y solucionar sus limitaciones. Pudiendo así almacenar en ella todo tipo de información además de proveer de todo un conjunto de métodos para facilitar determinadas tareas de una manera eficiente.

Desde un principio, y como requisito del proyecto, se ha decidido hacer uso de Hecuba como herramienta de gestión de Cassandra. Pese a ello en la actualidad existen otras herramientas con propósitos y funciones parecidas, una de ellas es DataStax, cuya empresa recibe el mismo nombre. Dicha empresa se dedica a ayudar en el desarrollo *open source* de Apache Cassandra, a la vez que comercializar una versión modificada de Cassandra que incluye distintas funcionalidades adicionales, bajo el nombre de DataStax[5] Enterprise.

En la actualidad DataStax es utilizada por grandes clientes a lo largo del mundo como Netflix, Spotify o eBay, siendo un producto que desde su nacimiento en 2010 ha conseguido una rápida y gran expansión en el mundo del *Big Data*.

Figura 4. Logo de la empresa DataStax

2.2. Despliegue de la implementación

El tipo de herramientas que se utilizarán a lo largo de este proyecto presentan un reto importante en cuanto a su instalación y correcta configuración en la máquina escogida. Gran cantidad de dependencias de software y la configuración de manera manual de

[5] Página web de DataStax: https://www.datastax.com/

ciertos parámetros en los sistemas en los que se ejecutará hacen que sea necesario encontrar una manera más sencilla de instalarlo todo.

En la actualidad el *despliegue* de este tipo de proyectos se suele hacer mediante contenedores de Docker, de manera que en su interior se encuentran todas las dependencias y configuraciones necesarias, pudiendo instalar dicho contenedor en otra máquina y que funcione independientemente del sistema operativo y su versión. Un contenedor en si es una unidad independiente, su funcionamiento es muy parecido a una imagen de un sistema utilizada en herramientas de virtualización de sistemas, pero al hacer uso de los recursos del sistema en el que se encuentra alojado, puede ocupar mucho menos espacio y ser más eficiente.

Docker aporta muchos beneficios como encapsulación, aislamiento y portabilidad, además de tratarse de un software libre, por todo ello ha conseguido hacerse con gran parte del mercado de herramientas utilizadas para el *despliegue*, incluyendo también este proyecto.

2.3. Análisis Big Data

En la actualidad existen multitud de herramientas para facilitar el análisis de grandes cantidades de datos, dentro de este conjunto se pueden diferenciar dos grandes grupos.

Aquellas herramientas enfocadas sobre todo a la visualización de los datos, como podrían ser QlikSense, PowerBI o Tableau[6]. En su mayoría suelen ser softwar de pago y están orientados a mostrar reportes dentro de una interfaz interactiva, en donde el usuario puede llevar a cabo filtrados de los datos en tiempo real. Este tipo de herramientas están pensadas para todo tipo de usuarios ya que no requieren de fuertes conocimientos de programación e incorporan todo un conjunto de plantillas y gráficos predefinidos.

El otro conjunto de herramientas lo conforman lenguajes de programación como R y Python, los cuales dan un enfoque en el tratamiento estadístico de los datos y sus posibles transformaciones. En los últimos años Python ha ido ganando mucha popularidad y con ello también su utilización para el análisis de datos gracias a la existencia de librerías como Pandas, NumPy, Scikit-learn o Matplotlib[7].

[6] Páginas web de QlikSense, PowerBI i Tableau: https://www.qlik.com https://powerbi.microsoft.com/ https://www.tableau.com/

[7] Páginas web de Pandas, NumPy, ScikitLearn i Matplotlib: https://pandas.pydata.org/ https://numpy.org/ https://scikit-learn.org/ https://matplotlib.org/

3. Metodología

3.1. Fases del desarrollo

Para el desarrollo correcto de las soluciones de software de este proyecto, se ha determinado una metodología constituida por dos fases, con el objetivo de englobar desde la definición de los objetivos hasta la comprobación de los requisitos de la solución desarrollada.

1. **Análisis de requerimientos y diseño de la implementación.** En esta primera fase se quiere analizar cuál es el conjunto de requisitos que debe cumplir la solución desarrollada. Posteriormente se debe realizar un diseño de implementación y si fuera necesario, analizar el funcionamiento de herramientas de terceros que serán utilizadas en el desarrollo.
2. **Implementación del software y ejecución de los test.** Teniendo en cuenta el diseño creado en la fase anterior se procede a desarrollar la solución. Para validar las implementaciones es necesario que el software pase correctamente los test. Esta fase ha sido trabajada de manera iterativa, cuando los test no se ejecutaban correctamente se analizaba dónde estaba el fallo y se modificaba la implementación.

3.2. Posibles obstáculos

Este proyecto se basa en la utilización de tres herramientas principalmente, Hecuba, COMPSs y Dislib, las cuales han sido desarrolladas en el Barcelona Supercomputing Center (BSC). Como fuente de información se ha tenido acceso a las guías de estas herramientas, disponibles de manera pública en sus correspondientes repositorios online.

Esto implica que la información disponible en internet acerca de posibles errores es bastante limitada. Por ello se ha considerado oportuno disponer de canales de comunicación directos con los desarrolladores de dichas herramientas, ayudando así a acelerar el proceso de desarrollo.

3.3. Herramientas de seguimiento

Todo el software desarrollado se encuentra disponible en un repositorio de GitHub. Mediante el uso de esta plataforma se ha gestionado todo el control de versiones del código, también ha ayudado a disponer del código en cualquier momento, facilitando la comunicación con compañeros de equipo.

3.4. Métodos de validación

Para validar las implementaciones desarrolladas se ha hecho uso de un conjunto de tests, los cuales pretenden simular la mayor parte de los posibles casos de uso de la herramienta. Como primera fase de validación de la implementación, estos test han sido ejecutados en local, en la máquina en la que se ha configurado todo el sistema de contenedores de Docker.

Debido a que la solución desarrollada formará parte de los archivos de una nueva versión de Dislib ha sido necesario cumplir con los requisitos y métodos de validación utilizados por su equipo de desarrollo. Por ello se ha hecho uso de Travis CI[8], este es un servicio de integración continua de GitHub que permite ejecutar los test y garantizar que el código funciona correctamente. En el momento en el que se sube nuevo código al repositorio, Travis CI crea un entorno remoto de ejecución, según que ha sido especificado en un archivo .yml. En el interior de este fichero se debe definir la estructura de contenedores de Docker, como deben de ser iniciados y que test deben ser ejecutados.

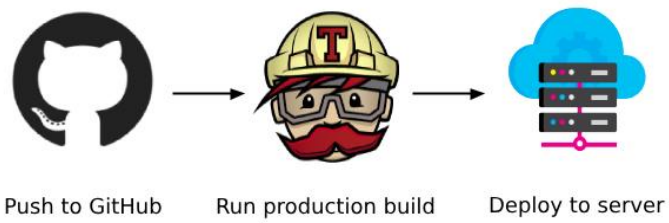

Figura 5. Proceso de despliegue usando Travis CI

[8] Página web de la platforma Travis CI: https://travis-ci.com/

3.5. Gestión del tiempo

El proyecto ha sido dividido en diferentes *work packages*, cada uno de ellos con sus tareas internas y objetivos a cumplir. Previamente a empezar con el desarrollo de la implementación se llevó a cabo una definición de la cantidad de tiempo necesaria, marcando unas fechas de inicio y final. Durante el desarrollo de esta tesis, debido a la situación provocada por la Covid-19 fue necesario hacer modificaciones importantes en la organización, dichos cambios se encuentran reflejados en detalle en el anexo.

Para tener una representación visual de dicha organización y llevar a cabo un control de las tareas, se hizo uso del diagrama Gantt de la Fig. 6.

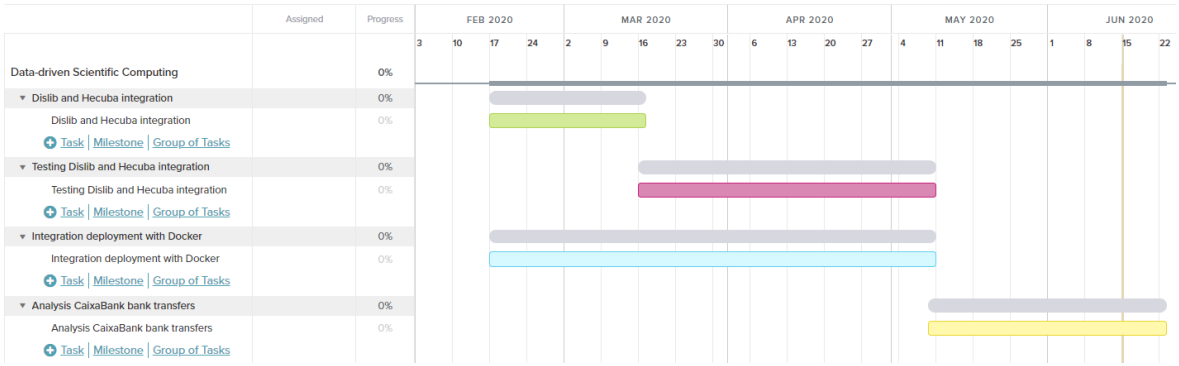

Figura 6. Diagrama Gantt inicial del proyecto

4. Desarrollo del proyecto

4.1. Entorno Docker

Debido a las ventajas que aporta la utilización de contenedores de Docker para el despliegue del proyecto y su instalación en distintas máquinas, se decide hacer uso de esta tecnología de virtualización. El primer paso a llevar a cabo consiste en el diseño del sistema y la configuración del conjunto de contenedores que formarán parte de este.

Se decide optar por un sistema formada por dos contenedores, uno de ellos encargado de almacenar todos los datos y el otro de efectuar los cálculos requeridos.

- Contenedor de almacenamiento, tiene instalado la base de datos Apache Cassandra. Pese a que en el entorno propuesto los dos contenedores se encuentran en la misma máquina, la base de datos podría estar ubicada en otra máquina física distinta siempre y cuando esta fuese accesible por red desde el otro contenedor.

- Contenedor de procesamiento, contiene la instalación de COMPSs, Dislib y Hecuba. En este contenedor se encontrará almacenado el código del usuario y llevará a cabo la ejecución de los programas haciendo uso de COMPSs y Dislib. Por otro lado Hecuba será la herramienta encargada de gestionar las distintas peticiones a Cassandra con el fin de evitar inconsistencias en el almacenamiento y facilitar su uso.

4.2. Integración Dislib y Hecuba

Tal y como ha sido desarrollado previamente, Dislib hace uso de ds-Arrays para sus procesos de cálculo, estos guardan ciertas similitudes con los objetos StorageNumpy que Hecuba es capaz de almacenar en Cassandra. Por lo tanto es necesario implementar un proceso de traducción entre ambos tipos de objeto.

4.2.1. Análisis ds-Array de Dislib

Cuando se está trabajando con grandes cantidades de datos la computación distribuida permite agilizar todo el proceso, como requisito para poder trabajar en paralelo es necesario realizar un tratamiento previo de los datos.

Por ejemplo, si se quiere calcular el valor medio de un *array* de 2 dimensiones de gran tamaño se deberían sumar todos los valores del objeto y después dividirlo entre el número de valores totales. Si este proceso se quiere realizar de manera distribuida se puede dividir

el *array* en porciones y que cada una de ellas vaya a una unidad de computación distinta, estas suelen recibir el nombre de *workers*, una vez cada *worker* tiene el valor medio de su porción lo devuelve al proceso maestro que lo había invocado. A continuación el proceso maestro calculará el valor medio de todo array original a partir de las medias de las porciones, obteniendo así el mismo resultado que el calculado de manera no distribuida.

Mediante el anterior ejemplo se puede entender como existe la necesidad de dividir los datos para que puedan ser tratados de manera simultánea, como solución a esta necesidad la librería Dislib utiliza el concepto de *block*, el cual es una porción del array original con las medidas que el usuario ha querido definir. La clase ds-Array implementa un objeto que en su interior contiene una lista con estos *blocks*, además de todo un conjunto de métodos que facilitan su manipulación al usuario.

En la Fig. 7 se puede observar un ejemplo de ejecución con la interfaz de Dislib, en la que se crea un ds-Array a partir de una matriz 10x10, cada block resultante consiste en un *array* de 2 filas y 10 columnas, tal y como el usuario ha definido en la variable *block_size*.

```
# Se define el tamaño de los bloques que formarán parte del DS-Array como una mátriz 2x10
block_size = (2, 10)
```

```
# Se crea un numpy array de dimensiones 10x10
x = np.array([[j for j in range(i * 10, i * 10 + 10)]
                  for i in range(10)])
x
```

```
array([[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14, 15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24, 25, 26, 27, 28, 29],
       [30, 31, 32, 33, 34, 35, 36, 37, 38, 39],
       [40, 41, 42, 43, 44, 45, 46, 47, 48, 49],
       [50, 51, 52, 53, 54, 55, 56, 57, 58, 59],
       [60, 61, 62, 63, 64, 65, 66, 67, 68, 69],
       [70, 71, 72, 73, 74, 75, 76, 77, 78, 79],
       [80, 81, 82, 83, 84, 85, 86, 87, 88, 89],
       [90, 91, 92, 93, 94, 95, 96, 97, 98, 99]])
```

```
# Se crea un DS-Array con el numpy array de 2 dimensiones y el tamaño de bloque definido
data = ds.array(x=x, block_size=block_size)
data
```

```
ds-array(blocks=(...), top_left_shape=(2, 10), reg_shape=(2, 10), shape=(10, 10), sparse=False)
```

```
# Objeto resultante
data._blocks
```

```
[[array([[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9],
         [10, 11, 12, 13, 14, 15, 16, 17, 18, 19]])],
 [array([[20, 21, 22, 23, 24, 25, 26, 27, 28, 29],
         [30, 31, 32, 33, 34, 35, 36, 37, 38, 39]])],
 [array([[40, 41, 42, 43, 44, 45, 46, 47, 48, 49],
         [50, 51, 52, 53, 54, 55, 56, 57, 58, 59]])],
 [array([[60, 61, 62, 63, 64, 65, 66, 67, 68, 69],
         [70, 71, 72, 73, 74, 75, 76, 77, 78, 79]])],
 [array([[80, 81, 82, 83, 84, 85, 86, 87, 88, 89],
         [90, 91, 92, 93, 94, 95, 96, 97, 98, 99]])]]
```

Figura 7. Ejemplo creación de un ds-Array

4.2.2. Análisis StorageNumpy de Hecuba

La clase StorageNumpy de Hecuba permite instanciar objetos que en su interior contienen un Numpy array, están diseñados para ser almacenados en Cassandra de manera que el objeto en si hace una referencia a la base de datos. Cuando se crea un StorageNumpy es necesario pasarle al constructor el Numpy array de los datos y el nombre que queremos que tenga en Cassandra, Hecuba de manera automática hará persistente este objeto

almacenándolo. A continuación en la Fig. 8 se muestra un ejemplo de creación de un StorageNumpy mediante la interfaz de Hecuba.

```
# Se crea un numpy array de 10x10
x = np.arange(100).reshape(10, -1)
x
```

```
array([[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14, 15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24, 25, 26, 27, 28, 29],
       [30, 31, 32, 33, 34, 35, 36, 37, 38, 39],
       [40, 41, 42, 43, 44, 45, 46, 47, 48, 49],
       [50, 51, 52, 53, 54, 55, 56, 57, 58, 59],
       [60, 61, 62, 63, 64, 65, 66, 67, 68, 69],
       [70, 71, 72, 73, 74, 75, 76, 77, 78, 79],
       [80, 81, 82, 83, 84, 85, 86, 87, 88, 89],
       [90, 91, 92, 93, 94, 95, 96, 97, 98, 99]])
```

```
# Llamamos al constructor del StorageNumpy pasando como argumentos los datos y el nombre
data = StorageNumpy(input_array=x, name="hecuba_dislib.test_array")
data
```

```
StorageNumpy([[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9],
              [10, 11, 12, 13, 14, 15, 16, 17, 18, 19],
              [20, 21, 22, 23, 24, 25, 26, 27, 28, 29],
              [30, 31, 32, 33, 34, 35, 36, 37, 38, 39],
              [40, 41, 42, 43, 44, 45, 46, 47, 48, 49],
              [50, 51, 52, 53, 54, 55, 56, 57, 58, 59],
              [60, 61, 62, 63, 64, 65, 66, 67, 68, 69],
              [70, 71, 72, 73, 74, 75, 76, 77, 78, 79],
              [80, 81, 82, 83, 84, 85, 86, 87, 88, 89],
              [90, 91, 92, 93, 94, 95, 96, 97, 98, 99]])
```

```
# Se puede comprobar si el objeto se ha persistido de manera correcta mirando el flag ._is_persistent
data._is_persistent
```

```
True
```

```
# ID identificador del objeto dentro del sistema de almacenamiento de Cassandra
data.getID()
```

```
'b211ff27-4d6b-3810-a0ef-ee243ec3cece'
```

Figura 8. Ejemplo creación de un StorageNumpy

Como se puede observar se ha creado de manera satisfactoria un StorageNumpy y este se ha almacenado en Cassandra bajo el nombre de "test_array", con este nombre se podrá recuperar el objeto en cualquier momento. Los StorageNumpy al almacenarse en Cassandra también disponen de un identificador UUID único, este se encuentra en el campo "storage_id" o invocando al método "getID()".

Cuando se define un StorageNumpy la información de su interior es una referencia a la base de datos, siguiendo con el ejemplo de la Fig. 7 el StorageNumpy no contiene el conjunto de números hasta el 99, tan solo tiene una referencia a Cassandra donde sí se encuentran esos datos. Gracias a los métodos que implementa Hecuba, de manera transparente para el usuario, le permite a este tratar el StorageNumpy como si fuera un Numpy array, por ello puede imprimirlo por pantalla o modificar sus valores sin ser consciente que Hecuba está trabajando en todo momento de manera transparente.

A continuación en la Fig. 9 se accede al contenedor de Docker donde se encuentra Cassandra. Haciendo uso de la consola de comandas CQLSH, la cual permite la interacción con Cassandra mediante *querys* en lenguaje CQL, se accede a la tabla "hecuba_dislib" y se busca el objeto "test_array", Cómo se puede observar, por el

resultado de la *query*, el objeto se encuentra almacenado de manera persistente en Cassandra.

Figura 9. StorageNumpy almacenado en Cassandra mediante Hecuba

4.2.3. Métodos implementados

Para llevar a cabo los procesos de escritura y lectura entre ds-Arrays y StorageNumpys se han creado las funciones make_persistent() y load_from_hecuba(). Además, la función _merge_blocks(), ya existente en la librería Dislib, ha sufrido modificaciones para poder tratar con ds-Arrays persistentes.

Los objetos de la clase ds-Array disponen del método make_persistent(), proporcionado por Dislib, este se encarga de invocar a los métodos de Hecuba necesarios para almacenar el objeto en Cassandra. Como parámetro de entrada necesita saber el nombre con el que se almacenará el objeto en la base de datos y devuelve el ds-Array con cada block definido como un StorageNumpy que hace referencia a Cassandra. También se crea y almacena el conjunto de datos sin división en blocks en un solo StorageNumpy, esto ayuda al rendimiento del *slicing* [7] de Python, una forma de extraer subestructuras de un conjunto más grande.

```
def make_persistent(self, name):
        """
        Stores data in Hecuba.

        Parameters
        ----------
        name : str
            Name of the data.

        Returns
        -------
        dsarray : ds-array
            A distributed and persistent representation of the data
            divided in blocks.
        """
        if self._sparse:
            raise Exception("Data must not be a sparse matrix.")

        x = self.collect()
        persistent_data = StorageNumpy(input_array=x, name=name)
        # self._base_array is used for much more efficient slicing.
        # It does not take up more space since it is a reference to the db.
        self._base_array = persistent_data

        blocks = []
        for block in self._blocks:
            persistent_block = StorageNumpy(input_array=block, name=name,
                                            storage_id=uuid.uuid4())
            blocks.append(persistent_block)
        self._blocks = blocks

        return self
```

Figura 10. Función make_persistent()

La función load_from_hecuba() permite al usuario recuperar la información almacenada en Cassandra en forma de ds-Array persistente, se entiende como tal a un ds-Array donde

cada block es un StorageNumpy almacenado en Cassandra con las dimensiones que el usuario ha especificado. Esta función tan solo requiere como parámetro de entrada el nombre del objeto.

```python
def load_from_hecuba(name, block_size):
    """
    Loads data from Hecuba.

    Parameters
    ----------
    name : str
        Name of the data.
    block_size : (int, int)
        Block sizes in number of samples.

    Returns
    -------
    storagenumpy : StorageNumpy
        A distributed and persistent representation of the data
        divided in blocks.
    """
    persistent_data = StorageNumpy(name=name)

    bn, bm = block_size

    blocks = []
    for block in persistent_data.np_split(block_size=(bn, bm)):
        blocks.append(block)

    arr = Array(blocks=blocks, top_left_shape=block_size,
                reg_shape=block_size, shape=persistent_data.shape,
                sparse=False)
    arr._base_array = persistent_data
    return arr
```

Figura 11. Función load_from_hecuba()

La anterior versión de la función _merge_blocks(), existente en la librería Dislib, era capaz de transformar en un único Numpy array el conjunto de *blocks* de un ds-Array. Ahora esta función también es capaz de realizar lo mismo incluso si el objeto de entrada se trata de un ds-Array persistente en el cuál cada uno de sus *blocks* es un StorageNumpy.

```python
@staticmethod
def _merge_blocks(blocks):
    """
    Helper function that merges the _blocks attribute of a ds-array into
    a single ndarray / sparse matrix.
    """
    sparse = None

    if blocks[0][0].__class__.__name__=="StorageNumpy":
        res=[]
        for block in blocks:
            value=list(block)
            line=np.hstack(value)
            res.append(line)
        return np.concatenate(res)


    b0 = blocks[0][0]
    if sparse is None:
        sparse = issparse(b0)

    if sparse:
        ret = sp.bmat(blocks, format=b0.getformat(), dtype=b0.dtype)
    else:
        ret = np.block(blocks)

    return ret
```

Figura 12. Función _merge_blocks()

A continuación en la Fig. 13 se muestra un ejemplo de ejecución de merge_blocks() con un ds-Array no persistente. El resultado es un array que contiene todos los datos del conjunto de *blocks* del objeto inicial.

```
# Se crea un ds-Array
block_size = (2, 10)
x = np.array([[j for j in range(i * 10, i * 10 + 10)]
                        for i in range(10)])
data = ds.array(x=x, block_size=block_size)
data._blocks
```

```
[[array([[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9],
         [10, 11, 12, 13, 14, 15, 16, 17, 18, 19]])],
 [array([[20, 21, 22, 23, 24, 25, 26, 27, 28, 29],
         [30, 31, 32, 33, 34, 35, 36, 37, 38, 39]])],
 [array([[40, 41, 42, 43, 44, 45, 46, 47, 48, 49],
         [50, 51, 52, 53, 54, 55, 56, 57, 58, 59]])],
 [array([[60, 61, 62, 63, 64, 65, 66, 67, 68, 69],
         [70, 71, 72, 73, 74, 75, 76, 77, 78, 79]])],
 [array([[80, 81, 82, 83, 84, 85, 86, 87, 88, 89],
         [90, 91, 92, 93, 94, 95, 96, 97, 98, 99]])]]
```

```
# Se pasa la lista de arrays a la función merge_blocks
array_from_dsarray=data._merge_blocks(data._blocks)
array_from_dsarray
```

```
array([[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14, 15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24, 25, 26, 27, 28, 29],
       [30, 31, 32, 33, 34, 35, 36, 37, 38, 39],
       [40, 41, 42, 43, 44, 45, 46, 47, 48, 49],
       [50, 51, 52, 53, 54, 55, 56, 57, 58, 59],
       [60, 61, 62, 63, 64, 65, 66, 67, 68, 69],
       [70, 71, 72, 73, 74, 75, 76, 77, 78, 79],
       [80, 81, 82, 83, 84, 85, 86, 87, 88, 89],
       [90, 91, 92, 93, 94, 95, 96, 97, 98, 99]])
```

Figura 13. Ejemplo ejecución _merge_blocks() con un ds-Array no persistente

En la Fig. 14 se replica el ejemplo anterior pero haciendo primeramente el objeto ds-Array persistente llamando al método de Dislib make_persistent(). Con Hecuba se consigue que para el usuario el hecho de que un objeto sea o no persistente, no implique ningún cambio en su código, todo es un proceso transparente para él.

```
data.make_persistent(name="hecuba_dislib.test_array")
data._blocks
```

```
[StorageNumpy([[[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9],
                [10, 11, 12, 13, 14, 15, 16, 17, 18, 19]]]),
 StorageNumpy([[[20, 21, 22, 23, 24, 25, 26, 27, 28, 29],
                [30, 31, 32, 33, 34, 35, 36, 37, 38, 39]]]),
 StorageNumpy([[[40, 41, 42, 43, 44, 45, 46, 47, 48, 49],
                [50, 51, 52, 53, 54, 55, 56, 57, 58, 59]]]),
 StorageNumpy([[[60, 61, 62, 63, 64, 65, 66, 67, 68, 69],
                [70, 71, 72, 73, 74, 75, 76, 77, 78, 79]]]),
 StorageNumpy([[[80, 81, 82, 83, 84, 85, 86, 87, 88, 89],
                [90, 91, 92, 93, 94, 95, 96, 97, 98, 99]]])]
```

```
array_from_persistent=data._merge_blocks(data._blocks)
array_from_persistent
```

```
array([[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14, 15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24, 25, 26, 27, 28, 29],
       [30, 31, 32, 33, 34, 35, 36, 37, 38, 39],
       [40, 41, 42, 43, 44, 45, 46, 47, 48, 49],
       [50, 51, 52, 53, 54, 55, 56, 57, 58, 59],
       [60, 61, 62, 63, 64, 65, 66, 67, 68, 69],
       [70, 71, 72, 73, 74, 75, 76, 77, 78, 79],
       [80, 81, 82, 83, 84, 85, 86, 87, 88, 89],
       [90, 91, 92, 93, 94, 95, 96, 97, 98, 99]])
```

Figura 14. Ejemplo ejecución _merge_blocks() con un ds-Array persistente

4.2.4. Proceso escritura en Cassandra

Desde el punto de vista del usuario, para almacenar los datos en Cassandra la interacción se llevaría a cabo con la interfaz de Dislib. En la Fig. 15 se puede ver un sencillo ejemplo de creación de un ds-Array y su posterior escritura en Cassandra mediante el método de Dislib, make_persistent().

```
input = np.random.rand(30,5)
data = dislib.array(x=input, block_size=(15,5))
data.make_persistent(name="hecuba_dislib.test_persistent")
```

Figura 15. Ejemplo de escritura de un ds-Array

Por dentro el método make_persistent() de Dislib se comunica con la interfaz de Hecuba, a continuación se desarrolla el conjunto de pasos que se llevan a cabo para la escritura de un ds-Array dentro de la base de datos.

1. Se define el ds-Array haciendo uso de su constructor, al cual se le pasa el conjunto de datos y se define el tamaño de los *blocks*. Como resultado se obtiene una variable de tipo ds-Array (para el caso de la Fig. 15 corresponde a la variable data) que guarda todos los *blocks* como Arrays dentro de la lista *_blocks*.
2. Para guardar este ds-Array en Cassandra se llama al método de Dislib make_persistent() indicando el nombre con el que se quiere identificar a dicho objeto.
3. Dentro de make_persistent() se llevan a cabo dos acciones, por un lado con la función collect() y merge_blocks() se crea un StorageNumpy en la variable _base_array que contiene todo el array de datos, sin la división en blocks.
4. Por otro lado, se itera en cada *block* del ds-Array original, creando para cada uno de ellos un StorageNumpy con los mismos datos. Cada StorageNumpy será almacenado en la lista _blocks y se encontrará siempre sincronizado con su homólogo de Cassandra.

Cabe recalcar que a partir de este momento cualquier modificación que sufra el contenido del ds-Array persistente, será automáticamente sincronizado con la base de datos gracias a Hecuba. Con esta implementación el usuario no requiere tener conocimientos de la interfaz de Hecuba o Cassandra ya que la implementación hecha en Dislib lleva a cabo todos los procedimientos especificados en la lista anterior, permitiendo al usuario despreocuparse de los procesos de almacenamiento o el modelo de datos y que pueda centrarse en el desarrollo con Dislib.

4.2.5. Proceso lectura de Cassandra

Suponiendo un caso en el que el usuario ya ha creado y hecho persistente un ds-Array, se puede hacer uso de la función load_from_hecuba() para recuperarlo, en la Fig. 16 se muestra la interacción que el usuario debe de llevar a cabo con Dislib, siendo la variable *data*, el ds-Array almacenado en Cassandra bajo el nombre de "hecuba_dislib.test_persistent" y con un tamaño de los blocks de 15 filas y 5 columnas.

```
data = dislib.load_from_hecuba(name="hecuba_dislib.test_persistent",block_size=(15,5))
```

Figura 16. Ejemplo de lectura de un ds-Array persistente

A continuación se muestran el conjunto de procesos que se efectúan dentro de la función load_from_hecuba() para llevar a cabo dicha tarea, y que interactúan con la interfaz de Hecuba.

1. Se invoca al método load_from_hecuba() indicándole el nombre que tiene el objeto en Cassandra y el tamaño de *block* con el que se quiere que devuelva los datos.
2. Dentro de load_from_hecuba() se llama al constructor de StorageNumpy, indicándole el nombre del objeto que se quiere recuperar, Hecuba al detectar que este objeto ya existe en Cassandra lo devuelve.
3. Para la creación del ds-Array es necesario que los datos del array estén divididos en *blocks* del tamaño indicado por el usuario. Mediante la función np_split() el array del StorageNumpy queda dividido en porciones.
4. Por último, se crea el objeto ds-Array, pasándole el conjunto de *blocks* persistentes basados en un StorageNumpy.

4.3. Procesado datos CaixaBank

Tras llevar a cabo la integración de las distintas herramientas y su testeo, se ha procedido a hacer uso de ellas para el análisis de un *dataset* de transferencias bancarias proporcionado por CaixaBank, empresa colaboradora con el proyecto I-BiDaaS, en torno al cual se ha desarrollado esta tesis.

El objetivo es detectar posibles anomalías en las transferencias, mediante una clusterización no supervisada. Los datos recibidos han sido previamente encriptados por CaixaBank haciendo uso del Coeficiente de Dice [8], el cual transforma los datos de

entrada devolviendo valores similares que mantienen el peso y la distribución de las muestras con respecto a los valores originales. Antes de empezar a trabajar con dichos datos es necesario hacer un preprocesado, ya que estos no cumplen con los requisitos de los algoritmos que se utilizaran posteriormente. Principalmente se han llevado a cabo tres acciones:

- Eliminar aquellas columnas que tan solo contienen un único valor igual para todo el *dataset*. Ya que no aportan ningún valor diferencial y por lo tanto son innecesarias.
- Remplazar todos los nulos por un valor numérico como el -1. Esto es necesario para el poder utilizar los algoritmos de Dislib.
- Hacer una codificación de tipo *one hot encoding* [9] para aquellas columnas cuyos datos sean de tipo categórico, es decir, valores no numéricos pero limitados en el número de posibilidades distintas.

La codificación *one hot encoding* es necesaria ya que determinados algoritmos de Machine Learning de Dislib solo pueden trabajar con valores numéricos. El procedimiento consiste en crear tantas columnas como valores posibles puede tener la columna original y que se determine con un 1 o un 0 si dicha característica es cumplida o no por la transacción. A simple vista podría parecer más sencillo hacer una tabla de equivalencias numérica para cada valor, sin embargo esto provocaría que los posteriores algoritmos de Machine Learning vieran con más peso aquellos con el mayor valor.

Label Encoding

Food Name	Categorical #	Calories
Apple	1	95
Chicken	2	231
Broccoli	3	50

→

One Hot Encoding

Apple	Chicken	Broccoli	Calories
1	0	0	95
0	1	0	231
0	0	1	50

Figura 17. Ejemplo codificación “One Hot”

Como resultado se obtiene que los datos han pasado de 89 columnas a tener un total de 501, ante tal situación una clusterización con K-means sería difícil de representar y analizar. Por ello se decide reducir el número de columnas mediante la aplicación de una PCA con la que transformar los datos a un espacio de 3 dimensiones.

El algoritmo PCA [10] calcula las nuevas dimensiones según como las variables se relacionan entre sí, queriendo conseguir que con el nuevo número reducido de dimensiones

se pueda caracterizar, de la mejor manera, el espacio inicial y a continuación, calcular los valores de los datos adaptados a las nuevas dimensiones. En la Fig. 18 se ejemplifica dicho proceso para la conversión de un espacio tridimensional a bidimensional.

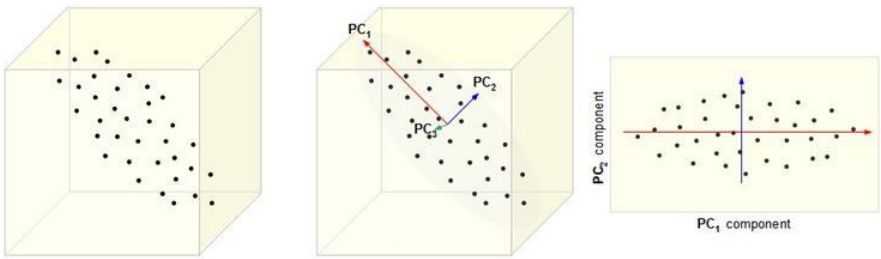

Figura 18. Transformación PCA de en un espacio 3D a uno 2D

Cuando se quiere aplicar una PCA a un *dataset* con tantas dimensiones surge un problema, todas ellas son numéricas pero no se tiene en cuenta el escalado entre las medidas. Un ejemplo claro, sería si se compararan unos datos en los que se mide la altura y el peso de un conjunto de personas, una diferencia entre dos personas de 0,5 Kg es poco significativa pero 0,5 m de altura supone algo mucho más característico e inusual y por lo tanto tiene mucha más importancia.

Por ello antes de aplicar una PCA es recomendable hacer una estandarización de los datos. En este caso se ha hecho uso de la función StandardScaler [11] de la librería Scikit-learn [12] de Python la cual aplica la transformación de la Fig. 19.

$$z = \frac{x - u}{\sigma}$$

$$x = datos\ de\ entrada$$
$$u = media\ de\ los\ datos$$
$$\sigma = desviación\ estándar$$

Figura 19. Transformación aplicada por StandardScaler

Por último previamente a llevar a cabo la clusterización es necesario que se le indique al algoritmo de K-means [13] cuantos clústeres deberá formar, para agilizar este proceso Dislib incluye el algoritmo DBScan [14]. Este algoritmo sirve para la clusterización según los patrones de densidad de los datos, en este caso concreto su uso será para determinar el número de clústeres que deberá formar el K-means.

5. Resultados

5.1. Verificación de la integración de Dislib y Hecuba

Para verificar que la integración llevada a cabo es correcta y que las funciones añadidas sean publicadas en el repositorio de GitHub, ha sido necesario llevar a cabo unos tests de verificación. Por una parte con los tests se ha querido cerciorarse de que una vez la estructura de datos se ha hecho persistente en Cassandra no se ha perdido información y se permite que esta sea manipulada de la misma forma. Por otro lado se ha probado que los algoritmos de Machine Learning que implementa Dislib funcionan correctamente, debido a la naturaleza de los resultados de estos algoritmos la manera de validarlo ha sido ejecutando el mismo algoritmo a dos ds-Arrays, ambos con los mismos datos, pero uno siendo persistente y el otro no, determinando por lo tanto que el resultado era correcto si en ambos casos era el mismo. Concretamente se ha probado el funcionamiento de los algoritmos K-means, PCA, DBScan, KNN.

Para llevar a cabo estos tests se ha hecho uso de la herramienta de integración continua Travis CI, la cual se encuentra integrada con GitHub, y que se ejecuta de manera automática cuando se actualiza un repositorio. Travis además utiliza imágenes de Docker para poder ejecutar en ellas las pruebas que se le indiquen, por lo que se ha podido hacer uso de la imagen creada durante este proyecto y que contiene todas las herramientas instaladas. Para la ejecución Travis necesita de un archivo .travis.yml, que será detectado automáticamente, el cual contiene el conjunto de tareas a realizar, como por ejemplo indicar que dos contenedores levantar y los archivos script a ejecutar. Este archivo, los tests ejecutados y los scripts se encuentran detallados en el anexo.

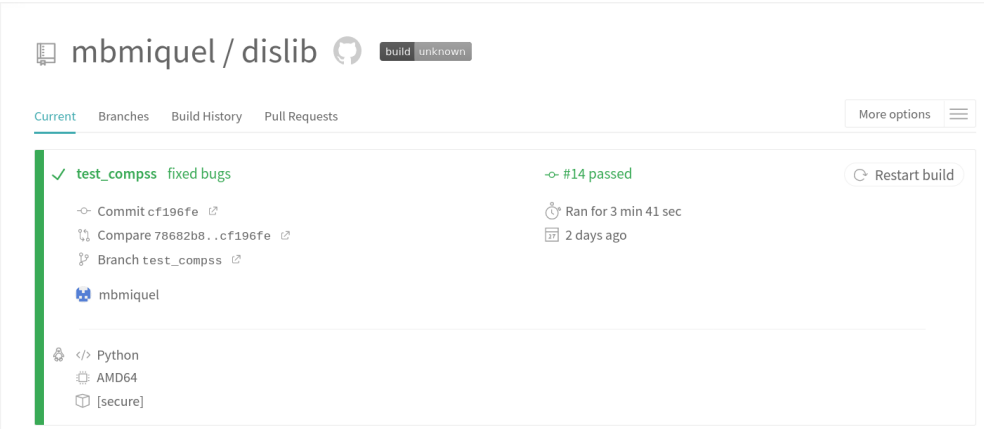

Figura 20. Mensaje resultante de la ejecución de los tests en Travis CI

5.2. Análisis datos CaixaBank

Tras todo el procesado de los datos llevado a cabo en apartados previos, se ha podido aplicar una clusterización no supervisada con el algoritmo de K-means. Finalmente el resultado obtenido es el mostrado en la Fig. 21 en donde cada color representa un clúster distinto y se tienen tan solo las tres dimensiones espaciales resultantes de la PCA aplicada, los puntos rojos representan los centros de los clústers.

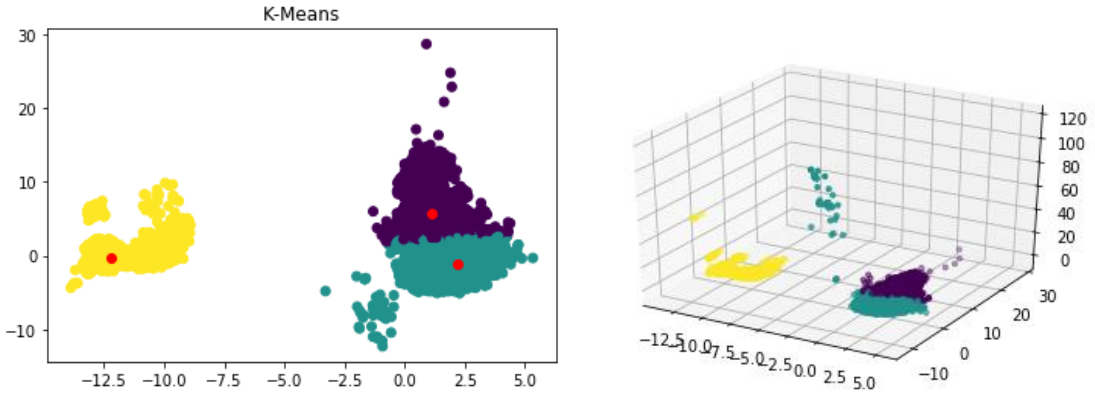

Figura 21. K-means bidimensional (izquierda) y tridimensional (derecha)

Como se puede observar la mayoría de transacciones se encuentran en un espacio prácticamente bidimensional en el que los clústers tienen muy poca variación en el eje de las Z, correspondiente a la tercera dimensión de la PCA. Sin embargo se puede ver que hay un conjunto de muestras que destacan por encontrarse muy separadas del resto de muestras.

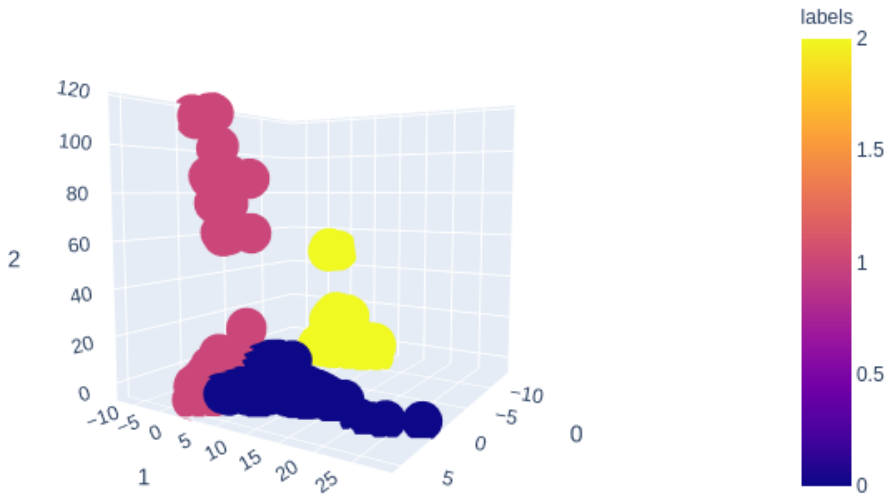

Figura 22. Detalle muestras lejanas al núcleo

Debido a que la PCA es un proceso no supervisado es difícil determinar cuál es la relación entre las dimensiones resultantes y las originales, para ayudar en este proceso se ha representado la correlación entre los distintos atributos. Como se puede observar en la Fig. 23, la tercera dimensión, que corresponde al eje de representación Z, tiene una fuerte caracterización por un reducido número de atributos.

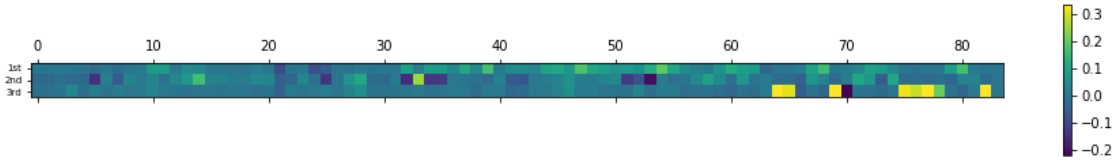

Figura 23. Heat-plot que relaciona la PCA con las primeras 83 dimensiones

Para analizar esta casuística con mayor detalle se ha procedido a aislar las muestras marcadas como posible anomalías y se han comparado con el resto de muestras pertenecientes al mismo clúster. El resultado ha sido que todas las muestras del mismo clúster, excepto aquellas marcadas como posibles anomalías, tenían igual valor para los atributos resaltados en amarillo en la Fig. 23. Estos resultados fueron reportados al equipo de CaixaBank que trabaja en el proyecto I-BiDaaS, para así tratar de entender el significado de negocio de dichos atributos.

El dataset utilizado recoge gran cantidad de información entorno a las transferencias, como por ejemplo los datos personales de quien la efectúa, la cantidad de dinero, si había adjunto un asunto, quien había sido el empleado que había atendido, entre otros datos. En concreto los atributos resaltados con anterioridad, corresponden a indicativos de las pantallas de procedimiento por las cuales el empleado ha pasado para efectuar la transferencia, estos procedimientos tienen cierta variabilidad ya que los empleados no tienen por qué hacerlo siempre de la misma forma, por lo tanto no se puede afirmar que se trate de una anomalía.

Para acabar de determinar que se trataba de un falso positivo se procedió a comparar el valor de los atributos de estas muestras con la media de todo el clúster, viendo así que no había ninguna diferencia remarcable para el resto de los atributos no destacados en la Fig. 23 y confirmando así la teoría de que se trataba de falsos positivos.

Analizando los valores medios de cada clúster se pueden extraer las siguientes conclusiones sobre sus características (los colores de clúster indicados corresponden a los mostrados en la Fig. 21):

- Clúster 0 (morado). Este clúster se caracteriza por la falta de información en los atributos relacionados con los datos personales de la persona que efectúa la transferencia. Cabe destacar que está formado en su gran mayoría por transferencias que pertenecen a cuentas asociadas a gente joven. También se trata del clúster formado por las transferencias de mayor cantidad.
- Clúster 1 (verde). Se trata del clúster del cual las muestras tienen una mayor cantidad de información respecto a los datos personales del pagador. Aproximadamente el género de los titulares de las cuentas se encuentra dividido en un 50/50 entre varones y mujeres. Este clúster también se caracteriza por estar formado en un 31% por personas consideradas de la tercera edad, del total de cuentas bancarias de este clúster un 28% recibe una pensión y otro 25% tiene domiciliado su nómina. Por último en este clúster se encuentran las transferencias de menor cantidad monetaria.
- Clúster 2 (amarillo). Al igual que con el clúster 0, este se encuentra formado también por transferencias de cuentas de las cuales no constan datos personales como la dirección física o edad y género. En su conjunto también se tratan de transferencias cuyo nivel de rango de validación necesario del empleado es menor.

6. Presupuesto

A lo largo de este apartado se llevará a cabo un análisis de los costes del desarrollo del proyecto según sean directos o indirectos. Para el cálculo de las amortizaciones se ha tenido en cuenta que la duración del proyecto ha sido de 4 meses.

6.1. Costes directos

Para un mejor entendimiento de los costes directos derivados de este proyecto se ha querido analizar desde tres áreas distintas: recursos humanos, software y hardware.

6.1.1. Recursos humanos

En el desarrollo de este proyecto han participado distintos roles, en la tabla adjunta se muestra el coste por hora de cada uno de ellos.

Rol	**Coste por hora (€/h)**	**Horas**	**Coste (€)**
Desarrollador junior de software	11	480	5280
Desarrollador sénior de software	16	48	768
Gestor de proyecto	20	64	1280
Total		**592**	**7328**

Tabla 1. Costes por hora según el rol

El desarrollador junior ha sido el principal encargado de llevar a cabo todo el proyecto, por eso su implicación horaria ha sido mucho mayor, cumpliendo con una jornada laboral de 6 horas diarias.

6.1.2. Software

En la siguiente tabla se lista el conjunto de software utilizado para el desarrollo del proyecto, cuál ha sido su coste total y su amortización dependiendo de la vida útil del mismo.

Software	**Coste (€)**	**Vida útil (años)**	**Amortización (€)**
Github	0,00	1	0,00
Ubuntu 18.04	0,00	1	0,00

Microsoft Visual Code	0,00	1	0,00
Travis for open source project	0,00	1	0,00
COMPSs	0,00	1	0,00
Hecuba	0,00	1	0,00
Dislib	0,00	1	0,00
Apache Cassandra	0,00	1	0,00
Google Drive Docs	0,00	1	0,00
Total	**0,00**		**0,00**

Tabla 2. Costes software utilizado

Como se puede observar la totalidad del software utilizado es libre, su vida útil es de aproximadamente 1 año, ya que dependiendo del programa hay actualizaciones más o menos constantes, estas actualizaciones también son gratuitas.

6.1.3. Hardware

Para desarrollar el proyecto se ha hecho uso del hardware listado en la siguiente tabla:

Hardware	**Coste (€)**	**Vida útil (años)**	**Amortización (€)**
Portátil Dell Latitude 7490 [9]	1.189	4	99,09
Teclado Logitech K120 [10]	9,99	4	0,84
Ratón Logitech B100 [11]	4,99	4	0,42
Monitor Dell 25 [12]	179	4	14,92
Total	**1.382,98**		**115,27**

Tabla 3. Presupuesto hardware necesario

6.2. Costes indirectos

Como costes indirectos asociados al proyecto se encuentran:

Concepto	**Coste mensual (€)**	**Coste total (€)**
Alquiler oficina	1300	5200
Gastos consumibles (luz, agua, café)	200	800
Mobiliario oficina	400	1200
Total		**7200**

Tabla 4. Costes indirectos

El mobiliario utilizado en la oficina pertenece al arrendador y supone un incremento mensual de 400 € con respecto al alquiler de la oficina.

Precios extraídos de:

[9] https://www.pccomponentes.com/dell-latitude-7490-intel-core-i7-8650u-8gb-256gb-ssd-14

[10] https://www.pccomponentes.com/logitech-keyboard-k120-teclado-usb

[11] https://www.pccomponentes.com/logitech-b100-rat-n-negro

[12] https://www.pccomponentes.com/dell-p2419h-238-led-ips-fullhd?

7. Conclusiones y futuros desarrollos

Para poder llevar a cabo esta tesis primero ha sido necesario ser capaz de entender como es el funcionamiento por dentro de las distintas herramientas, esto ha requerido de mucho tiempo invertido ya que sobretodo el funcionamiento interno de COMPSs no ha sido fácil de entender. Para trabajar con este conjunto de librerías ha sido muy necesaria la coordinación y comunicación con los otros equipos de desarrollo, tanto para resolución de dudas sobre el funcionamiento como para reportar posibles modificaciones en sus herramientas, las cuales fueron costosas de encontrar pero imprescindibles para poder completar la integración.

Otra de las dificultades de esta implementación ha sido que al utilizar herramientas desarrolladas en el Barcelona Supercomputing Center, pese a ser de código abierto y de utilización libre, su uso no está muy extendido y por lo tanto la cantidad de información disponible en internet únicamente ha sido la de sus manuales de uso oficiales.

La utilización de Docker ha aportado gran cantidad de beneficios al desarrollo, permitiendo una fácil instalación de todas las herramientas y proporcionando un sistema base sobre el que ejecutar los tests en la plataforma Travis. Dicha imagen de Docker se encuentra disponible en la plataforma DockerHub para su descarga y utilización bajo el nombre dislib_hecuba_compss_production:0.6 [13].

Llevar a cabo un testeo exhaustivo de la integración ha sido muy importante para encontrar irregularidades en el funcionamiento que no habían sido detectadas a simple vista. Además toda integración con los test ha sido probada en el servidor de I-BiDaaS, para lo cual ha sido necesario entender cómo funciona el sistema de levantamiento de los distintos nodos, en este caso, correspondiente a distintas máquinas conectadas en una misma red. Para ello se ha necesitado pequeñas modificaciones en la imagen Docker utilizada, para que todos los requisitos sean cumplidos y se pueda ejecutar correctamente.

La propuesta de valor que añade esta integración al proyecto I-BiDaaS es muy importante si es comparado con la utilización únicamente de la librería Scikit-learn, muy extendida hoy en día para el análisis de datos. Gracias a Dislib se tienen todas las herramientas para el análisis de los datos y mediante COMPSs el cálculo de estas se consigue hacer de manera distribuida usando varios nodos. Con esta integración se añade además una funcionalidad muy importante que es el almacenamiento persistente de los datos en Cassandra mediante Hecuba, librería que hace todo este proceso transparente

[13] Link de la imagen de Docker alojada en el repositorio Docker Hub: https://hub.docker.com/r/emebemb/dislib_hecuba_compss_production/tags

para el usuario el cual no se debe de preocupar en ningún momento y puede trabajar con la estructura de datos como si de una variable alojada en su sistema se tratara.

El caso de uso llevado a cabo en esta tesis ha sido enfocado desde el punto de vista de las tareas a realizar por un científico de datos, el cual sería el potencial usuario del conjunto de herramientas de esta integración. En este aspecto se puede determinar que para llevar a cabo un buen análisis y extraer unas conclusiones que puedan aportar valor a la empresa, es muy necesario tener conocimiento de la lógica de negocio que hay detrás de los datos.

En definitiva se puede decir que las herramientas resultantes de esta integración cumplen con los requisitos especificados y aportan un valor añadido a las soluciones tecnológicas actualmente existentes.

Durante las primeras fases de organización de esta tesis se planificó que se iba a desarrollar una memoria cache virtual, pero debido a la situación provocada por la COVID-19 esta parte finalmente se desestimó en favor de llevar a cabo el caso de uso de los datos de CaixaBank. Es por ello que a corto plazo los siguientes pasos de desarrollo consistirán en añadir una caché, con la que se quiere conseguir mejorar el rendimiento de aquellos algoritmos de Dislib que utilizan de manera recursiva un mismo set de datos persistentes. Con esto se quiere evitar que Hecuba tenga que intervenir de manera innecesaria en aquellos casos en los que de una iteración a otra los datos no se han visto modificados.

En si el proyecto I-BiDaaS se seguirá desarrollando hasta finales de año por lo que más a largo plazo tocará trabajar con los otros equipos participantes, para conseguir conectar la interfaz gráfica que proporcionará la web, con todo el *backend* que incluirá en parte lo desarrollado en esta tesis.

Bibliografía:

[1] Alomar, Guillem; Becerra Fontal, Yolanda; Torres Viñals, Jordi. Hecuba: NoSql made easy. A: "BSC Doctoral Symposium (2nd: 2015: Barcelona)". 2nd ed. Barcelona: Barcelona Supercomputing Center, 2015, p. 136-137. https://upcommons.upc.edu/handle/2099/16589.

[2] J. Álvarez Cid-Fuentes, S. Solà, P. Álvarez, A. Castro-Ginard, and R. M. Badia, "dislib: Large Scale High Performance Machine Learning in Python," in *Proceedings of the 15th International Conference on eScience*, 2019, pp. 96-105. https://ieeexplore.ieee.org/abstract/document/9041782

[3] Lakshman, Avinash & Malik, Prashant. (2010). Cassandra — A Decentralized Structured Storage System. Operating Systems Review. 44. 35-40. 10.1145/1773912.1773922. https://www.researchgate.net/220624179_Cassandra_A_Decentralized_Structured_Storage_System

[4] Tejedor, E., Becerra, Y., Alomar, G., Queralt, A., Badia, R. M., Torres, J., … Labarta, J. S. (2017). PyCOMPSs: Parallel computational workflows in Python. Original Article The International Journal of High Performance Computing Applications, 31(1), 66–82. https://doi.org/10.1177/1094342015594678.

[5] Bashari Rad, Babak & Bhatti, Harrison & Ahmadi, Mohammad. (2017). An Introduction to Docker and Analysis of its Performance. IJCSNS International Journal of Computer Science and Network Security. 173. 8. https://www.researchgate.net/publication/318816158_An_Introduction_to_Docker_and_Analysis_of_its_Performance

[6] Kevin D. Smith, Jim J. Jewett, Skip Montanaro, Anthony Baxter. Decorators for Functions and Methods. https://www.python.org/dev/peps/pep-0318/

[7] Jakub Przywóski. Slicing – Pyhton Referemce (The Right Way) https://python-reference.readthedocs.io/en/latest/docs/brackets/slicing.html

[8] A Variation Coefficient Similarity Measure and Its Application in Emergency Group Decision-making. Xuanhua Xu, Liyuan Zhang and Qifeng Wan. Systems Engineering Procedia (5); 119 – 124. 2012 https://www.researchgate.net/publication/271561004_A_Variation_Coefficient_Similarity_Measure_and_Its_Application_in_Emergency_Group_Decision-making

[9] Potdar, Kedar & Pardawala, Taher & Pai, Chinmay. (2017). A Comparative Study of Categorical Variable Encoding Techniques for Neural Network Classifiers. International Journal of Computer Applications. 175. 7-9. 10.5120/ijca2017915495. https://www.researchgate.net/publication/320465713_A_Comparative_Study_of_Categorical_Variable_Encoding_Techniques_for_Neural_Network_Classifiers

[10] Matt Brems. A One-Stop Shop for Principal Component Analysis. (Visitado el 13-5-2020) https://towardsdatascience.com/a-one-stop-shop-for-principal-component-analysis-5582fb7e0a9c

[11] StandardScaler – ScikitLearn Documentation https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

[12] Scikit-learn: Machine Learning in Python, Pedregosa *et al.*, JMLR 12, pp. 2825-2830, 2011. http://jmlr.csail.mit.edu/papers/v12/pedregosa11a.html

[13] Martin Ester, Hans-Peter Kriegel, Jiirg Sander, Xiaowei Xu. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. https://www.aaai.org/Papers/KDD/1996/KDD96-037.pdf

[14] Jin X., Han J. (2011) *K*-Means Clustering. In: Sammut C., Webb G.I. (eds) Encyclopedia of Machine Learning. Springer, Boston, MA. https://doi.org/10.1007/978-0-387-30164-8_425

Anexo:

Anexo 1. Organización work packages

Previamente al inicio de esta tesis se redactó un documento de organización en el cual se definieron los objetivos a cumplir y las fechas en las que llevarlos a cabo. A mitad del desarrollo se hizo una revisión de dicha planificación en la que tan solo hubo pequeños cambios en las fechas de entrega. Posteriormente a la entrega de dicha revisión, la situación provocada por la COVID-19 y la necesidad de alargar nuevamente la fecha definitiva de entrega de la integración provocaron que diversos *work packages* sufrieran importantes modificaciones. A continuación se muestra la organización de los *work packages* propuestos en la entrega del documento de revisión.

WP	Título	Objetivos / Entregables	Fecha final
1	Integración de Dislib con Hecuba	Entregable: Conjunto de funciones para la comunicación entre ambas librerías.	16/3/20
2	Ejecutar tests de la integración	Objetivo: Conseguir una integración libre de errores.	~~31/3/20~~ 13/4/20
3	Despliegue del proyecto con Docker	Entregable: Una imagen de Docker con todas la librerias instaladas.	~~31/3/20~~ 13/4/20
4	~~Mejora del rendimiento~~	Entregable: Una memoria cache virtual para la mejora del rendimiento de procesos iterativos.	~~25/5/20~~
5	~~Analisis del rendimiento~~	Entregable: Conjunto de mediciones para comparar el rendimiento de la integración con y sin caché.	~~15/6/20~~

Tabla 5. Organización y entregables de los distintos work packages

Finalmente tras estudiar la situación actual y la capacidad de actuación, se decidió de manera consensuada, junto con el director y co-directora de esta tesis, que los *work packages* 4 y 5 no iban a ser llevados a cabo además de alargar de nuevo la entrega de los *work packages* 2 y 3 hasta el 8/5/20. Semanas antes de tomar dicha decisión surgió la

necesidad dentro del proyecto I-BiDaaS de llevar a cabo un caso de uso de dicha integración, en la que se analizaran unos datos de transferencias bancarias de CaixaBank. Teniendo en cuenta todo esto, se tomó la decisión de crear un nuevo *work package* en sustitución de los dos anteriores. En la Tabla 6 mostrada a continuación, se detallan los objetivos y plazos de entrega.

<table>
<tr><td>WP: Análisis transferencias bancarias de CaixaBank</td><td>WP4</td></tr>
<tr><td>Software</td><td rowspan="2">Inicio: 08-05-20

Final: 22-06-20</td></tr>
<tr><td>Descripción:

Para un correcto análisis de un conjunto de datos es necesario llevar a cabo un procesado de los datos teniendo en cuenta cuál es el objetivo, en este caso una clusterización no supervisada con K-means para la detección de posibles anomalías en las transferencias.</td></tr>
<tr><td>Tareas internas:

T1: Investigar sobre cuáles son las técnicas actuales para llevar a cabo un análisis de datos

T2: Prepocesado de los datos para los algoritmos de Dislib

T3: Aplicar algoritmos de Machine Learning de Dislib para clusterizar.

T4: Analizar resultados y extraer conclusiones</td><td>Entregables:

Dos Jupyter notebooks con el preprocesado y procesado de los datos, un archivo de Python con el procesado y un documento escrito con dicho análisis y las conclusiones.</td></tr>
</table>

Tabla 6. Work package análisis de transferencias bancarias de CaixaBank

Anexo 1. Bridge networks e inicialización de contenedores Docker

En un entorno en el que se están utilizando contenedores de Docker una "bridge network" es un enlace virtual en la que los contenedores de una misma máquina local, o que se encuentran dentro de la misma red física, pueden establecer conexión entre ellos si están conectados a la misma "bridge network" de Docker. Como estas redes son virtuales y dentro de una misma máquina podemos encontrar varias, estas son identificadas con un nombre, el cual servirá para indicar a los contenedores a que red deben conectarse.

Gracias a que Docker establece de manera automática toda una serie de normas, estas redes virtuales permiten aislar a los contenedores del exterior, haciendo el sistema más robusto frente a posibles ataques externos.

Para el caso concreto de este proyecto la "bridge network" recibe el nombre de "cassandra-bridge" y esta se inicializa con el siguiente comando:

```
docker network create --attachable --driver bridge cassandra_bridge
```

Debido a que la conexión entre contenedores se efectúa a través del protocolo SSH es necesario que se inicialicen los contenedores con el puerto 22 abierto y conectados a la red "cassandra_bridge".

Primeramente se inicializa el contenedor de almacenamiento, donde estará Cassandra instalado, bajo el nombre de "cassandra_container". Los propios desarrolladores de Apache Cassandra ponen a disposición de los usuarios un contenedor básico con Linux y Cassandra instalado para facilitar su uso con Docker a los desarrolladores. Esta imagen se encuentra en el repositorio online DockerHub, y es a partir de esta imagen que se inicializará el contenedor "cassandra_container" de nuestro sistema.

Con el atributo --network indicamos al contenedor que debe conectarse a la red "cassandra_bridge", previamente creada, y con --expose se abre su puerto 22 para permitir conexiones SSH.

```
docker run --rm --name cassandra_container --expose=22
--network=cassandra_bridge -d cassandra
```

Para la creación de la imagen del contenedor de procesamiento se hará uso de un documento Dockerfile proporcionado por el equipo de desarrollo de COMPSs, el cual

contiene los comandos para la correcta instalación de: PyCOMPSs, Dislib, Hecuba y todas las dependencias. Para la creación de la imagen, la cual recibirá el nombre de "emebe/dislib_hecuba_compss_production:0.6", es necesario ubicarse en la carpeta donde se encuentre el archivo y ejecutar el comando build.

```
docker build --tag emebe/ dislib_hecuba_compss_production:0.6 .
```

Para inicializar el contenedor a partir de la imagen previamente creada, con el puerto 22 abierto y añadirlo a la red "cassandra_bridge" se ejecutará:

```
docker run --expose=22 -network=cassandra_bridge -d --name dislib
emebe/dislib:0.1
```

Por último es necesario configurar los dos contenedores para que puedan efectuar conexiones SSH entre ellos sin necesidad de ninguna contraseña. Para ello se copiará para cada contenedor su key (ubicada en el archivo "id_rsa.pub") en el archivo "authorized_keys" del otro contenedor. De esta manera se habrá habilitado el SSH sin contraseña entre los dos contendores del sistema.

Anexo 2. Configuración Dockerfile del sistema

Para la creación de la imagen Docker que contiene toda la instalación de las distintas herramientas se ha hecho uso del siguiente documento Dockerfile de configuración.

```
FROM ubuntu:18.04

# ================================================================

# COMPSS AND PYCOMPSS INSTALLATION

RUN apt-get update && \

# Install Packages

    apt-get install -y --no-install-recommends && git && vim && wget && openssh-server
sudo && \

# Create Jenkins User

    useradd jenkins -m -s /bin/bash && \
```

```
# Add the jenkins user to sudoers
    echo "jenkins  ALL=(ALL)  NOPASSWD:ALL" >> /etc/sudoers && \
# Enable ssh to localhost for user root & jenkins
    yes yes | ssh-keygen -f /root/.ssh/id_rsa -t rsa -N '' > /dev/null && \
    cat /root/.ssh/id_rsa.pub > /root/.ssh/authorized_keys && \
    cp -r /root/.ssh /home/jenkins && \
# Make sure jenkins owns his files
    chown -R jenkins /home/jenkins/ && \
    chgrp -R jenkins /home/jenkins/ && \
# Enable repo compression
    git config --global core.compression 9 && \

# Dependencies for building COMPSs
# Build dependencies
    apt-get install -y --no-install-recommends maven \
# Runtime dependencies
    openjdk-8-jdk graphviz xdg-utils \
# Bindings-common-dependencies
    libtool automake build-essential \
# C-binding dependencies
    libboost-all-dev libxml2-dev csh \
# Extrae dependencies
    libxml2 gfortran libpapi-dev papi-tools \
# Misc. dependencies
    openmpi-bin openmpi-doc libopenmpi-dev uuid-runtime curl bc \
# Python-binding dependencies
    python3-dev python3-pip python3-setuptools && \
    pip3 install wheel dill decorator coverage numpy==1.15.4 ipython==7.9.0 \
    scipy==1.3.0 jupyter==1.0.0 scikit-learn==0.19.1 pandas==0.23.1 \
    matplotlib==2.2.3 flake8 codecov parameterized && \
# Configure user environment
```

```
# Add environment variables
    echo "JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64/" >> /etc/environment && \
    echo "MPI_HOME=/usr/lib/openmpi" >> /etc/environment && \
    echo "LD_LIBRARY_PATH=/usr/lib/openmpi/lib" >> /etc/environment && \
    mkdir /run/sshd && \
# Clone framework files for installation
    git clone --depth=1 --branch 2.6 https://github.com/bsc-wdc/compss.git framework &&
\
# Install COMPSs
    cd /framework && \
    ./submodules_get.sh && \
    ./submodules_patch.sh && \
    sudo -E /framework/builders/buildlocal -Np /opt/COMPSs && \
    rm -rf /framework /root/.m2 /root/.cache /home/jenkins/.COMPSs /tmp/* && \
    rm -rf /var/lib/apt/lists/*


ENV JAVA_HOME /usr/lib/jvm/java-8-openjdk-amd64/
ENV PATH
$PATH:/opt/COMPSs/Runtime/scripts/user:/opt/COMPSs/Bindings/c/bin:/opt/COMPSs/Runtime/s
cripts/utils
ENV CLASSPATH $CLASSPATH:/opt/COMPSs/Runtime/compss-engine.jar
ENV LD_LIBRARY_PATH /opt/COMPSs/Bindings/bindings-
common/lib:$JAVA_HOME/jre/lib/amd64/server


# ================================================================
# DISLIB INSTALLATION
RUN cd /
RUN git clone --branch test_compss https://github.com/bsc-dd/dislib.git dislib
RUN cd /dislib && python3 setup.py install
# ================================================================
# HECUBA INSTALLATION
# Hecuba requisites
RUN apt-get update && \
    apt-get install -y apt-transport-https ca-certificates gnupg software-properties-
```

```
common wget && \

    wget -O - https://apt.kitware.com/keys/kitware-archive-latest.asc 2>/dev/null |
sudo apt-key add - && \

    apt-add-repository 'deb https://apt.kitware.com/ubuntu/ bionic main' && \

    apt-get update && \

    apt-get install -y cmake

RUN apt-get install libuv1


# Installing Hecuba

RUN cd /

RUN git clone --branch installation_test https://github.com/bsc-dd/hecuba.git hecuba

RUN cd /hecuba && python3 setup.py install

RUN cd /hecuba/storageAPI/storageItf && \

    mvn assembly:assembly


# Bashrc modifications needed

RUN sed -i /'[ -z "$PS1" ] && return'/d ~/.bashrc

RUN echo 'export CONTACT_NAMES="cassandra_container"' >> /root/.bashrc && \

    echo 'export PYCOMPSS_NODES="localhost"' >> /root/.bashrc && \

    echo 'export NODE_PORT=9042' >> /root/.bashrc

RUN cat ~/.bashrc


# Expose SSH port and run SSHD

EXPOSE 22

CMD ["/usr/sbin/sshd","-D"]
```

Anexo 3. Archivos para la ejecución en Travis CI

Para poder ejecutar los tests dentro de la plataforma han sido necesarios tres archivos. Dentro de launch_cassandra.sh se indica que imagen será la utilizada en el contenedor con Cassandra y como se debe configurar la Docker network del sistema. En este caso Cassandra facilita su propia imagen del sistema y la pone a su disposición en Dockerhub de manera que el sistema si no encuentra la imagen "cassandra:latest" en local, la irá a buscar directamente online.

```
# Create Docker network

docker network create --attachable --driver bridge cassandra_bridge

# launch Cassandra and open port 22

CASSANDRA_ID=$(docker run --rm --name cassandra_container --expose=22 --
network=cassandra_bridge -d cassandra:latest)

sleep 30

# add environment variable CONTACT_NAMES needed by Hecuba

export CONTACT_NAMES="cassandra_container"

echo "Using Cassandra host: $CONTACT_NAMES"
```

El script run_tests.sh indica al sistema cual es el comando que deberá ejecutar, teniendo en cuenta cuales son todos los requerimientos de parámetros que necesita COMPSs para poder ejecutar en distribuido.

```
echo "Using Cassandra host $CONTACT_NAMES"

source ~/.bashrc


# Run the test_hecuba.py

runcompss --pythonpath="/usr/local/lib/python3.6/dist-packages/Hecuba-0.1.3.post1-
py3.6-linux-x86_64.egg/" --python_interpreter=python3  \\
--classpath=/hecuba/storageAPI/storageItf/target/StorageItf-1.0-jar-with-
dependencies.jar   --storage_conf="/dislib/storage_conf.cfg"
/dislib/tests/test_hecuba.py &> >(tee output.log)


# Check the unittest output because PyCOMPSs exits with code 0 even if there

# are failed tests (the execution itself is successful)

result=$(cat output.log | egrep "OK|FAILED")
```

```
echo "Tests result: ${result}"


# If word Failed is in the results, exit 1 so the pull request fails
if [[ $result =~ FAILED ]]; then
       exit 1
       fi
```

Por último en el archivo .travis.yml se hace referencia a los dos archivos anteriores para poder configurar el sistema de dos contenedores.

```
language: python3.6
install: skip
sudo: required
branches:
  only:
     - test_compss
     - /^release-.*/
services:
  - docker
env:
  global:
     - REGISTRY_USER=compss
     - TEST_CASSANDRA_VERSION=3.11.6
before_script:
     - source launch_cassandra.sh
     - docker run -it --network cassandra_bridge -d --name dislib
emebemb/dislib_hecuba_compss_production:0.5
script: "docker exec -e CONTACT_NAMES='cassandra_container' -e NODE_PORT=9042 dislib
/dislib/run_tests.sh"
```

Anexo 4. Procesado transferencias bancarias CaixaBank

Se adjunta el archivo Python mediante el cual se ha llevado a cabo el procesado de los datos de CaixaBank. Las funciones export_pca_labeled y export_dataset_labeled devuelven dos archivos csv de los datos clasificados según su clúster. Y la función save_pca_labeled_hecuba permite guardar dichos datos en Cassandra utilizando Hecuba.

```
import gc
import os
import unittest
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
from hecuba import config
from pycompss.api.api import compss_wait_on
import dislib as ds
from dislib.cluster import KMeans
from dislib.decomposition import PCA
from dislib.cluster import DBSCAN

def save_pca_labeled_hecuba(dataset_x_pca,labels):
    col0 = dataset_x_pca.collect()[:, 0]
    col1 = dataset_x_pca.collect()[:, 1]
    col2 = dataset_x_pca.collect()[:, 2]
    num=np.array([col0,col1,col2,labels]).transpose()
    result = ds.array(x=num, block_size=(250, 4))
    result.make_persistent(name="caixa_analysis.pca_labeled")

def export_pca_labeled(dataset_x_pca, labels):
    plot=pd.DataFrame(dataset_x_pca.collect())
    plot["labels"]=labels
    plot.to_csv("pca_labeled.csv", index=False)
```

```
def export_dataset_labeled(encoded, labels):
    encoded['labels']=labels
    encoded.to_csv("original_labeled.csv")

def main():
    #Loading data
    print("Loading data")
    encoded= pd.read_csv('samples.csv')
    #Need it for correct storage on hecuba, it doesn't affect any result
    h=encoded.values.copy('C')

    #StandardScaler
    print("StandardScaler")
    x = StandardScaler().fit_transform(h)

    #Configure Cassandra workspace
    print("Configure Cassandra workspace")
    config.session.execute("TRUNCATE TABLE hecuba.istorage")
    config.session.execute("DROP KEYSPACE IF EXISTS caixa_analysis")
    bn, bm = 250, 167
    dataset = ds.array(x=x, block_size=(bn, bm))
    dataset.make_persistent(name="caixa_analysis.test_array")

    #Data dimension reduction using PCA
    print("Data dimension reduction using PCA")
    pca = PCA(n_components=3)
    dataset_x_pca = pca.fit_transform(dataset)

    #DBSCAN to determine how many clusters
    print("DBSCAN to determine how many clusters")
```

```
    dbscanr2 = DBSCAN(eps=0.9, min_samples=50).fit(dataset_x_pca)
    n_clusters_=dbscanr2.n_clusters


    #Applying kmeans to the dataset
    print("Applying kmeans to the dataset")
    kmeans2 = KMeans(n_clusters=n_clusters_, random_state=500)
    labels=kmeans2.fit_predict(dataset_x_pca).collect()


    #Exporting data
    print("Exporting data")
    export_pca_labeled(dataset_x_pca, labels)
    export_dataset_labeled(encoded, labels)
    save_pca_labeled_hecuba(dataset_x_pca,labels)


if __name__ == '__main__':
    main()
```