UNIVERSIDAD DE VALLADOLID

Escuela de Ingeniería Informática de Valladolid

Trabajo Fin de Master

Master Universitario de Ingeniería Informática - BigData

Despliegue de un clúster Hadoop con Cloudera en un sistema de virtualización basado en Proxmox

Autor: **D. Javier Isaac Ramos López**

Tutor: **Anibal Bregón Bregón**

Tutor: **Miguel Ángel Martinez Prieto**

Valladolid, 25 de Junio de 2019

Primera impresión, Junio 2019

“Pido perdón a mis hijos por el tiempo que les he robado para realizar este trabajo, estoy seguro que lo entenderán en unos años, son una parte muy importante en mi vida, y doy las gracias a mis padres sin su tiempo y apoyo este trabajo no hubiera visto la luz.”

“Anibal, Miguel Ángel gracias por confiar en mí y por obtener los recursos proporcionados por el Departamento de Informática, conseguir poner orden en mi cabeza no es tarea fácil”

“y por último a quién ha estado silenciosamente apoyando mis decisiones, duro y arduo trabajo, no hay agradecimiento que lo compense”

Índice general

Primera Parte: Contexto

1. Introducción

En los últimos años el término Big Data aparece a menudo asociado a una gran cantidad de disciplinas, medicina, deporte, economía, aprendizaje, etc. Realmente, ¿sabemos que es Big Data?. A grandes rasgos Big Data se basa en la gestión y el análisis de grandes cantidades de datos.

Cuando pensamos en Big Data enseguida asociamos la idea a grandes volúmenes de datos heterogéneos. Esa gran variedad de datos se puede representar de muy diversas formas, puede ser de múltiples fuentes distintas e incompatibles entre sí, como por ejemplo sensores, GPS, equipos industriales, anemómetros, móviles, Webs, etc. Cada una se recibirá a una velocidad distinta y requerirá una velocidad de tratamiento diferente, incluso siendo necesario tratarla en tiempo real.

Aunque el término Big Data es relativamente reciente, la recolección de gran cantidad de datos para su posterior análisis es una actividad que se viene realizando desde hace muchos años. El **V**olumen de datos, la **V**ariedad de los mismos y la **V**elocidad de análisis son los términos habituales a los que se está acostumbrado. Con la aparición de Big Data se hablaba de las tres V's del Big Data, volumen, velocidad y variedad, pero hoy en día añadimos dos conceptos más la **V**eracidad y el **V**alor. De ahí que se hable de las cinco V's [30] que caracterizan el concepto de Big Data. Las tres primeras se deducen fácilmente de lo expuesto hasta el momento, volumen de los datos, velocidad a la que se consumen, y variedad de las fuentes de datos. La veracidad se asocia al grado de confianza que se establece sobre los datos a utilizar y determinará la calidad de los resultados y la confianza

en los mismos. Y por último, el valor, que es el fin último del análisis de datos Big Data, nos lo aporta el conocimiento e información útil que sacamos de los datos, permite construir nuevos productos y prestar nuevos servicios.

La necesidad de inferir conocimiento y obtener un rendimiento de él hace que el Big Data sea una tecnología en continuo cambio y demandada ampliamente por diversos ámbitos, ya sea científico, económico, político, social, de salud, de seguridad, etc. Esta es la razón por la cual se habla del nuevo petróleo, los datos. Los datos deben ser almacenados en bruto y refinados, al igual que el petróleo el valor se obtiene del refinamiento.

1.1 Contexto del TFM

En los siguientes apartados daremos una visión de los datos, cómo se clasifican en Big Data, la necesidad de una herramienta para poder manejarlos y como toda esta revolución ha cambiado el perfil profesional del ingeniero informático.

1.1.1 Los Datos

Manejar gran cantidad de datos nos lleva a pensar que los sistemas de almacenamiento habituales de la información, como los sistemas de archivos o bases de datos tradicionales, no dan una solución al problema que supone el tratamiento de tantos datos tan diversos y dispersos. Aunque está claro que estos sistemas tradicionales forman parte importante en la solución.

Para tomar consciencia del problema debemos estudiar de dónde proviene la información. El ser humano tiende a almacenar toda información que es capaz de generar. Hemos pasado de pensar en bit a bytes, a gigabytes y ahora a terabytes, como la medida habitual de volumen de información en un espacio de tiempo muy breve.

Figura 1.1: Origen de los datos y tipos

En el siglo XXI la unidad de medida que consideramos grande es del orden del petabyte $\mathbf{10^{15}}$ bytes. Pensemos que, "Facebook procesa 2.500 millones de entradas de información y más de 500 terabytes de información cada día" [32]. Esta información es la que generamos nosotros, pero existe más información, la denominada M2M (machine to machine) información digital que se genera entre máquinas, como puede ser los sensores de los aviones o barcos y que es tratada y almacenada al igual que la información que generamos los humanos.

Generamos gran cantidad de información y de diversa índole que es recogida en la nube ya sea privada o pública, ver Figura 1.1, esto hace que aparezcan empresas especializadas en las diversas fuentes y tipos de datos, y por último surgen empresas especializadas en analizar datos y obtener un valor de ellos.

Tipos de datos

Queda claro que cuando hablamos de Big Data lo primero que debemos solventar es de dónde recoger los datos y dónde almacenarlos, pero una vez hecho esto ¿qué hacemos?. La pregunta sería ¿qué problema es el que se quiere resolver?, y con este planteamiento clasificar los tipos de datos que necesitamos para resolver el problema planteado. En Big Data podemos clasificar los datos en 5 grandes tipos:

1. **Web y medios de comunicación social**. Este tipo de información se refiere a contenido web e información que es obtenida de las redes sociales como Facebook, Twitter, LinkedIn, blogs, etc. Por ejemplo:

   - Flujo de datos de click
   - Twitter Feeds
   - Publicaciones en Facebook
   - Contenidos web

2. **M2M**. Se refiere la información que generan dispositivos como sensores o medidores que capturan algún evento en particular (velocidad, temperatura, presión, variables meteorológicas, etc.), los cuales transmiten a otras aplicaciones que traducen estos eventos en información significativa. Por ejemplo:

   - Señales GPS.
   - Programas que leen los sensores de los Smarts
   - Lectores RFID usados en tarjetas, etiquetas, etc.
   - Sensores de plataformas petrolíferas.
   - Sensores de flotas de compañías aéreas.

3. **Grandes transacciones de datos**. Esta información es la que hace referencia a las grandes transacciones que realizan sectores empresariales, incluye registros de facturación en telecomunicaciones, registros detallados de las llamadas (CDR), etc. Por ejemplo:

   - Sistemas médicos
   - Registros de llamadas telefónicas
   - Registros de transacciones económicas

4. **Biométricos**. Este tipo de datos se generan en el área de seguridad e inteligencia. Información biométrica en la que se incluye huellas digitales, escaneo de la retina, reconocimiento facial, genética, etc. Este tipo de datos para las agencias de investigación son una información importante.

   - Reconocimiento facial
   - Datos genéticos
   - Datos de conducta

5. **Generados por el hombre**. Este tipo de información hace referencia a las personas y es generada por ellas, como puede ser:

   - Notas de voz.
   - Correos electrónicos
   - Estudios médicos
   - Documentos electrónicos

El objetivo de la clasificación será tomar consciencia para conseguir una mejor selección, tratamiento, clasificación y filtrado de los datos con lo que podríamos obtener la solución al problema de Big Data planteado.

1.1.2 La herramienta

Llegados a este punto en el que reconocemos cada tipo de dato con el que vamos a trabajar, necesitaremos las herramientas para hacer el proceso de tratamiento, almacenamiento, filtrado, etc. Es aquí cuando toma protagonismo el desarrollo de código abierto Hadoop [1]. Este conjunto de herramientas permitirá sacar un valor añadido a la información. Es a partir de este momento cuando el coste que supone almacenar tanta información pasa a producir un beneficio, ya que podremos tratar y analizar esta información para obtener un rendimiento de ella.

Hadoop está inspirado en el proyecto de Google File Sistem (GFS) y en el paradigma de programación MapReduce. El paradigma de programación MapReduce consiste en

Figura 1.2: DataCenter Google para Big Data

explotar el paralelismo dividiendo los datos distribuidos entre los nodos de un clúster. El concepto innovador en este tipo de clúster reside en que se procura mover la computación a donde están los datos y no al contrario. Mover los datos genera gran cantidad de latencia mientras se mueven de un nodo a otro del clúster. Esta tecnología Big Data es open source, y se considera el framework estándar para el almacenamiento de grandes volúmenes de datos, pero también para analizarlos y procesarlos, siendo utilizado por empresas como Facebook, Yahoo o Google. Uno de los puntos fuertes de esta tecnología es que soporta diferentes sistemas operativos, lo que facilita su despliegue en las principales plataformas en la nube. Las grandes empresas de alojamientos en la nube disponen enormes centros de datos o DataCenters dedicados a los despliegues de Big Data. Como se puede ver en el Figura 1.2, estos pueden contarse del orden de cientos de miles.

Tanto el sector empresarial como el investigador usan Hadoop de las siguientes formas:

- **Almacenamiento y archivo de datos de bajo coste**

  Dado que Hadoop está pensado para desplegarse en equipos de bajo coste, ahora nos podemos plantear almacenar información que no se consideraba crítica para su posterior análisis.

- **Sandbox para descubrimiento y análisis**

  Como Hadoop es un proyecto open source, hace que con una inversión mínima se puedan tener un entorno sandbox de prueba que ofrece la oportunidad de innovar. La analítica de Big Data en Hadoop puede ayudar a una organización a operar de manera más eficiente, descubrir nuevas oportunidades y obtener ventajas competitivas.

- **Data Lake**

  Los Data Lake son un repositorio de almacenamiento que contiene una gran cantidad de datos en su formato original, ya sean estructurados como sin estructurar, y de distintas fuentes, que se mantienen ahí hasta que sea necesario. Son los analistas de datos los que hacen uso de los Data Lakes para descubrir o analizar. Hay que tener claro que un Data Lake no es el reemplazo de los almacenes de datos. En las IT (Tecnologías de la Información) cómo asegurar y gobernar Data Lakes es un tema de enorme importancia hoy en día.

- **Complemento del Data WareHouse**

  En el Data WareHouse disponemos de información jerárquica que almacena datos en ficheros o carpetas. Hadoop tomará datos del Data Lake o del Data WareHouse, el objetivo final de cada organización es contar con una plataforma para almacenar y procesar datos de diferentes esquemas, formatos etc., para soportar diferentes casos de uso que se pueden integrar en diferentes niveles.

- **Internet de las Cosas (IoT) y Hadoop**

  Desde la aparición de IoT, el flujo de datos es constante, las interrelaciones entre las "cosas" conectadas a Internet es mayor. Esto nos permite descubrir y definir patrones que pueden ser monitorizados y generar instrucciones como respuesta. Estas instrucciones pueden mejorar continuamente ya que Hadoop se utiliza constantemente con nuevos datos que no coinciden con los patrones definidos anteriormente. Esta continua iteración entre los datos, el análisis, la toma de decisiones, el aprendizaje que se obtiene de esas decisiones es algo que sin Hadoop no sería posible.

1.1.3 Añadir valor a los datos

Hoy en día el reto no es sólo obtener datos y tratarlos. Como hemos visto hasta ahora, tenemos infinidad de fuentes de datos, sabemos cómo almacenarlas y disponemos de las herramientas necesarias para poder analizarlas. El verdadero reto consiste en saber utilizar estos datos. El nuevo petróleo del Siglo XXI surge de los datos, y el refinado de este petróleo es el que produce el dinero. Con técnicas de Big Data y Machine Learning para recolectar, clasificar, monitorizar, etc, podremos agregar el contenido desestructurado y convertirlo en una solución de contenidos ordenados para mejorar los procesos productivos y por lo tanto obtener un beneficio económico.

El proceso para obtener valor de los datos debe pasar por las 5 etapas de un proyecto Big Data:

- Definir una estrategia de negocio y selección de las fuentes de datos.
- Selección de la arquitectura. Evaluación de las características Big Data de las fuentes seleccionadas.
- Gestión de las fuentes de datos. Se identifican cuestiones como las fuentes de datos, son streaming o para procesar por lotes, se deben procesar en tiempo real o no.
- Técnicas y procesos de análisis.
- Creación de visualizaciones.

Para analizar estos datos surge "Business Intelligence". Business Intelligence [2] se considera que es el conjunto de metodologías y herramientas de software necesarias para la recolección, análisis y selección de un gran volumen datos originados de diversas fuentes y la forma de convertirlos en información útil. Para ello se ha desarrollado una tecnología que abarca un buen número de disciplinas.

1.1.4 Perfil profesional Big Data

La revolución de los datos ha llevado a que el perfil profesional de los Ingenieros Informáticos se vea claramente dirigido por la demanda. Si analizamos los perfiles en Big Data, demandados por las empresas, y los perfiles que ofrecen las diferentes Universidades podemos encontrar ente 5 y 7 perfiles profesionales bien definidos. Parece más razonable una división de 7 perfiles aunque es comprensible que algunos se solapen en parte y requieran especializaciones comunes. Estos 7 perfiles [18] son los siguientes:

1. **Chief Data Officer (CDO)**, es el responsable de asegurar que la organización funciona como una Data Driven Company, esto significa que toma decisiones estratégicas basadas en análisis de datos e interpretación de los mismos. Por lo tanto, es la persona que lidera el resto de perfiles profesionales relacionados con el Big Data que haya en una empresa.

2. **Citizen Data Scientist**, son los encargados de la "ciencia de datos". Su labor es fundamental para poder extraer conocimiento e información valiosa de los datos. Deben tener visión de extremo a extremo de los datos, esto es, de que datos se dispone y que conocimiento se pretende obtener de ellos. De esta forma son capaces de construir modelos analíticos y algoritmos, utilizando sus habilidades relacionadas con las matemáticas, la estadística, la programación y visualización de datos.

3. **Data Scientists**, se encargan de la limpieza de los datos para proporcionarlos de una manera apropiada a los usuarios para el desarrollo de aplicaciones Big Data. Debe estar formado en técnicas de Big Data y conocimientos sobre gestión de

bases de datos, arquitecturas de clusters, lenguajes de programación y sistemas de procesamiento de datos.

4. **Data Engineer**, este perfil engloba a todos los especialistas que se encargan de proporcionar los datos de una manera accesible y apropiada a los usuarios y científicos de datos. Es un perfil especializado en infraestructura Big Data. Su función principal es encargarse de todo el proceso para desarrollo de aplicaciones Big Data y para ello es necesario tener gran conocimiento en gestión de bases de datos, arquitecturas de clúster, lenguajes de programación y sistemas de procesamiento de datos.

5. **Data Steward**, es responsable de mantener la calidad, disponibilidad y seguridad de los datos. Es el responsable de utilizar los procesos de gobierno de datos de una organización para garantizar la idoneidad de los elementos de datos, tanto el contenido como los metadatos. Sus funciones serán, mejorar el almacenamiento y presentación de los datos, la disponibilidad y la seguridad de los datos en toda la empresa.

6. **Business Data Analyst**, encargado de recoger e interpretar las necesidades de los usuarios de negocio para los Data Scientist y encargado de presentar los datos obtenidos.

7. **Data Artist**, son expertos en Business Analytics encargados de hacer comprender datos complejos a los usuarios, para ello emplean herramientas visuales.

Una vez que tenemos claro cuáles son los perfiles del Big Data podemos centrarnos en el perfil en el que se desarrollará este TFM. El perfil principal será el de **Data Engineer**, o *Ingeniero de Datos* en Español. En el TFM se intentará crear la infraestructura necesaria para proporcionar los datos de una manera accesible y apropiada a los científicos de datos.

1.2 Motivación

Instalar algunas herramientas del enorme ecosistema de Hadoop es una tarea relativamente sencilla, podemos encontrar muchos manuales en Internet y una buena cantidad de bibliografía. Bien distinto es instalar y mantener un clúster Big Data, analizar su rendimiento, decidir cuáles son las configuraciones hardware más eficientes etc. Adquirir experiencia en este tipo de entornos, metodologías y herramientas requiere tiempo, este proyecto pretende ser el comienzo.

Como se expone en la introducción, el TFM se enmarca dentro del perfil profesional de **Data Engineer (Ingeniero de Datos)**. Dada mi trayectoria profesional, es un complemento

y mejora en mi carrera profesional, esta es una razón de peso para obtener la motivación necesaria para formarme en la gran cantidad de tecnologías que habrá que integrar.

Por otra parte, en el puesto laboral que desempeño hay una demanda de sistemas de cómputo para Big Data debido a las asignaturas que se imparten y los proyectos de investigación que se desarrollan.

1.3 Objetivos

El objetivo principal de este TFM consiste en encontrar una metodología para poder desplegar de manera semi-automática un clúster Big Data basado en la tecnología Cloudera[1] en un entorno de virtualización de Proxmox VE[2]. Este proceso nos ayudará a adquirir experiencia en gestión de un clúster Cloudera y en desplegar máquinas virtuales para asignarlas roles dentro del clúster de Cloudera. Además deberemos cumplir los siguientes subobjetivos:

1. Establecer una metodología para poder desplegar las máquinas virtuales necesarias para integrarlas en el clúster de Cloudera. Este objetivo consistirá en analizar las tecnologías de despliegue automático, seleccionar la tecnología y utilizarla para ajustar el tamaño del clúster a los recursos disponibles.

2. Entender y familiarizarse con las distintas herramientas Hadoop para poderlas desplegar en un clúster de Cloudera. Dado que el clúster pretende emplearse en diferentes ámbitos de Big Data, deberemos familiarizarnos con herramientas de aprendizaje automático, análisis de datos, base de datos NoSQL.

3. Analizar los diferentes entornos Big Data existentes, comparando con Cloudera.

4. Estudiar el proceso de instalación de Cloudera. Este objetivo consistirá en analizar las tecnologías a desplegar, configuraciones de las mismas y aprender a usar el sistema de despliegue de Cloudera Manager, comprender las funciones de Cloudera Manager y las siguientes herramientas, como mínimo, que se incluyen en el CDH de Cloudera: ZooKeeper, HDFS, Flume, HBase, Hive, Hue, Oozie, Solr, Spark, YARN.

5. Evaluar las herramientas de monitorización del rendimiento del clúster y obtener la experiencia necesaria para poder optimizar los procesos.

6. Ejecutar pruebas de diversas características, para ver el funcionamiento del clúster desplegado.

[1] https://www.cloudera.com/

[2] https://www.proxmox.com/en/proxmox-ve

1.4 Organización de la Memoria

La organización del TFM se ha dividido en cuatro partes. La primera parte donde hablaremos del contexto del TFM. Esta parte consta de dos secciones, la introducción, donde hablaremos de la importancia del Big Data. Los datos, cómo se obtiene valor de ellos y de la influencia en el perfil profesional informático debida a la revolución del Big Data. En la segunda sección, de la primera parte, hablaremos del marco tecnológico, Apache Hadoop, las soluciones más relevantes, existentes en la actualizad basadas en la tecnología de Apache Hadoop y una comparativa entre de ellas. Hablaremos brevemente de la virtualización, que es, y cuales son sus ventajas.

En la segunda parte del TFM, trataremos como configurar el clúster de Cloudera, el hardware disponible, que es fundamentar conocer cuantos recursos disponemos para poder crear los nodos del clúster Cloudera bien dimensionados. El sistema de virtualización sobre el que se desplegará el clúster, para poder diseñar el sistema de despliegue. En la segunda sección hablaremos de la metodología seguida y la instalación del clúster de Cloudera. Cerraremos esta parte del TFM hablado de Cloudera Manager, el gestor de clústeres de Cloudera.

La tercera parte de la memoria estará compuesta por una única sección pruebas. En ella se realizarán test con una serie de conjuntos de datos para probar el clúster desplegado. Usaremos varias tecnologías en las pruebas como Flume, HDFS, Hive, Ozzie y trataremos un gran conjunto de datos recogidos de OpenSky Network, una red receptora de datos de tráfico aéreo, para extraer las trayectorias de los aviones.

2. Marco Tecnológico

En este capítulo veremos unos conceptos básicos de Hadoop y de Big Data, alguno de los componentes principales de Apache Hadoop y las tecnologías que usan la mayoría de las instalaciones basadas en este Framework. Apache™ Hadoop® es un framework diseñado para procesar de forma eficaz conjuntos de datos de gran tamaño, pero en lugar de utilizar un equipo grande para procesar y almacenar los datos, Hadoop facilita la creación de clústeres con hardware de consumo para analizar conjuntos de datos masivos en paralelo [23]. Esta idea unida a la tecnología de virtualización ha permitido realizar proyectos Big Data a sectores empresariales y de investigación que antes sería impensable por el coste económico.

El ecosistema de Big Data en los últimos años se ha disparado, las tecnologías que dan soporte a Big Data, Data Science, Machine Learning, AI (Artificial Intelligence) continúan avanzando, apareciendo otras nuevas y cada vez más eficientes. En la Figura 2.1 Se puede ver el panorama de todas las tecnologías que existen alrededor de Big Data en el 2018. Entender todas ellas requiere mucho tiempo y experiencia y no es posible abarcarlas todas.

Realizaremos una descripción de las tecnologías que hemos visto en las diferentes arquitecturas de Hadoop y haremos analizaremos con más hincapié en las que se desenvolverá el proyecto. Veremos los modelos que existen en la actualidad, basados en Apache Hadoop. Realizaremos una breve descripción de cada uno de ellos viendo las herramientas que instalan. Con esta información realizaremos una comparativa que nos hará entender por qué seleccionamos Cloudera como la distribución para crear el clúster de Hadoop.

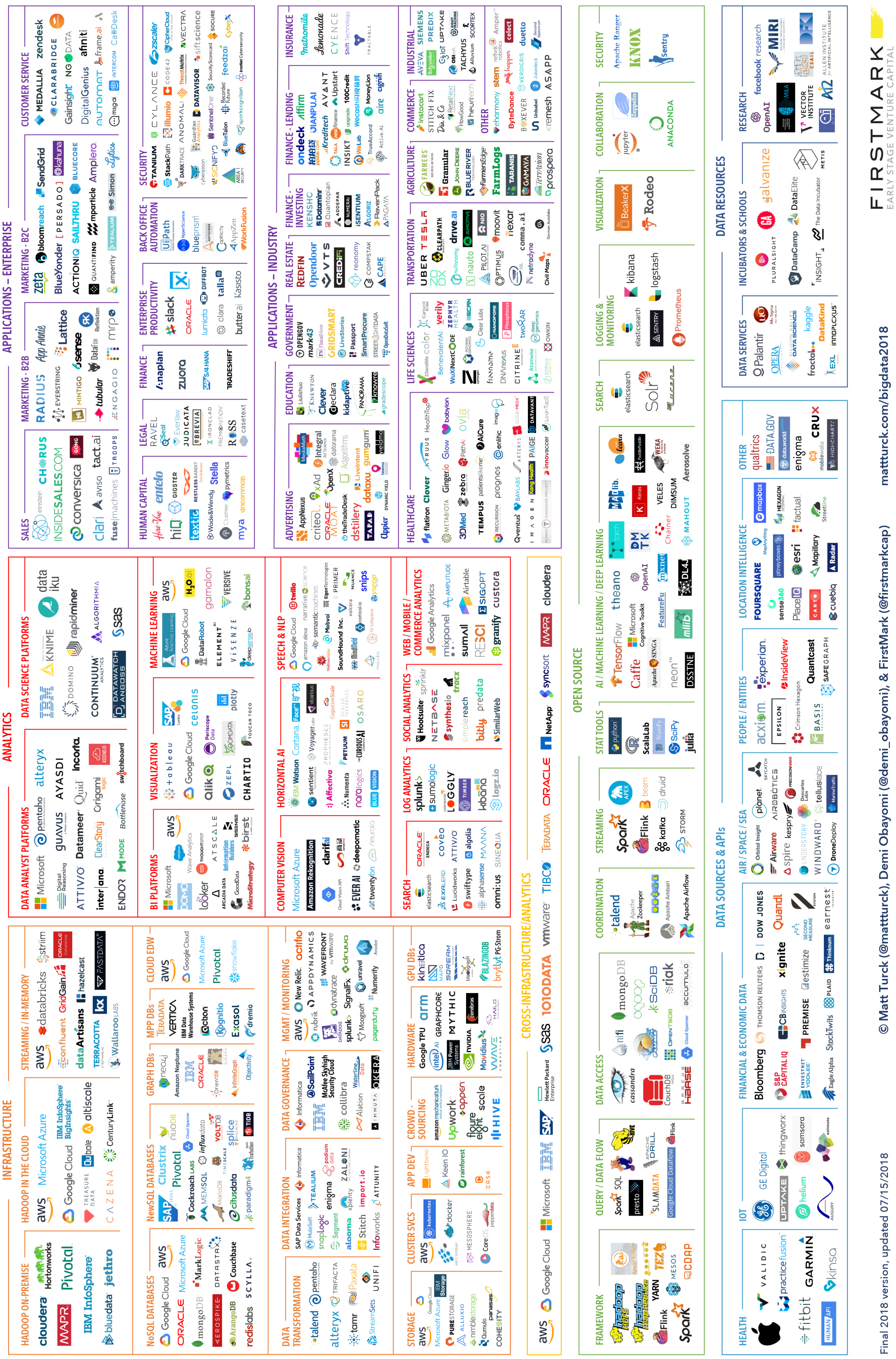

Figura 2.1: Landscape Big Data 2018

También analizaremos brevemente la tecnología de virtualización, otro sector tecnológico en el que se desenvuelve el Big Data hoy en día y con el que deberemos trabajar en este proyecto. De esta forma habremos descrito el marco tecnológico en el que nos desenvolveremos.

2.1 Contexto

Para procesar los datos de un proyecto en Big Data se requiere de un proceso ETL (Extract, Transform, Load), los datos deben ser almacenados mientras pasan de una fase a otra del proyecto Big Data. Para tener éxito en el proceso de transformación de los datos se necesita garantizar la trazabilidad del proceso ETL, los pasos intermedios deben ser almacenados de forma temporal o permanente. Es aquí donde aparece el Data Lake, en este repositorio es donde entran, se transforman y se sacan los datos en un proyecto de Big Data. Las herramientas de Hadoop se nutren del Data Lake y nutren al Data Lake, cada una da una solución diferente de cómo se tratan esos datos, ya sea en modo batch, tiempo real o streaming, pero todas se sirven del Data Lake.

Un Data Lake es un repositorio de almacenamiento que puede almacenar una gran cantidad de datos estructurados y no estructurados. Al igual que en un lago real en el que entran múltiples afluentes, un lago de datos tiene múltiples fuentes y de registros que fluyen en tiempo real. El objetivo de un Data Lake es ofrecer una visión de los datos sin refinar a los analistas de datos. Para comprender la arquitectura de un Data Lake hay que conocer los siguientes conceptos, que representamos en la Figura 2.2:

- **Ingestión de Datos**. Permite a los conectores que obtengan datos de diferentes fuentes y se carguen en el Data Lake. Esta ingesta de datos aporta tipos de datos estructurados o no, múltiples fuentes en tiempo real, de tipo Batch o de una sola carga. Así como muchos tipos de bases de datos, servidores web, correos electrónicos, IoT, FTP, NFS, log, etc.
- **Almacenamiento de datos**. Es un almacenamiento escalable, que permite un acceso rápido a la exploración de los datos con múltiples formatos de datos.
- **Gobernanza de Datos**. La gobernanza de los datos consiste en el proceso de gestión de la disponibilidad, la facilidad de uso, la seguridad y la integridad de los datos utilizados en una organización.
- **Seguridad**. La seguridad es un factor clave hoy en día en un Data Lake. Debe comenzar con el almacenamiento y continuar en la búsqueda y el consumo de los datos. La GUI (Graphical User Interface) de acceso a los datos del Data Lake debe

Figura 2.2: Conceptos de un Data Lake [24]

controlar el acceso a los usuarios y que estos sólo puedan gestionar los datos suyos o públicos del Data Lake.

- **Calidad de los datos**. La calidad de los datos es esencial en la arquitectura de un Data Lake. Hay que tener siempre presente que los datos se utilizan para determinar el valor comercial, por lo tanto, la extracción de información a partir de datos de baja calidad conducirá a información de mala calidad y por lo tanto a un rendimiento económico deficitario.
- **Descubrimiento de datos**. Es otra de las etapas importantes en un Data Lake, los datos deben ser etiquetados para poder ser organizados e interpretados por los ingenieros de datos del Data Lake. El proceso de etiquetado ayuda a la comprensión de los datos.
- **Auditoría de datos**. Es importante auditar el seguimiento de cambios en los elementos importante del conjunto de datos y también capturar cómo / cuándo / y quién cambia estos elementos. De esta forma seremos capaces de evaluar el riesgo de las decisiones tomadas.
- **Linaje de datos**. Este proceso consiste en saber lo que sucede con los datos a lo largo del tiempo. Facilita las correcciones de errores en un proceso de análisis de datos desde el origen hasta el destino.
- **Exploración de datos**. Ayuda a identificar el conjunto de datos correcto, es vital antes de comenzar a utilizar los datos y es la fase inicial del análisis de datos.

La arquitectura de un Data Lake está estructurada por niveles [21]. Como vemos en la Figura 2.3, en los niveles inferiores residen los datos que están en su mayoría en reposo, en los niveles superiores muestran datos transaccionales, incluso en tiempo real. Estos datos fluyen a través del sistema con poca o ninguna latencia. Los niveles descritos son los siguientes:

1. **Nivel de ingestión**. Los niveles del lado izquierdo representan las distintas fuentes de datos, que se pueden cargar en el Data Lake por lotes, grandes o pequeños y en tiempo real.

2. **Nivel de ideas**. Los niveles a la derecha representan el resultado del análisis y la investigación, para la representación de los datos se puede usar SQL, consultas NoSQL, o incluso herramientas como Excel o Qlik para el análisis de datos.

3. **Nivel de almacenamiento**. HDFS sería como la zona de aterrizaje para todos los datos que están en reposo en el sistema.

4. **Nivel de destilación**. En este nivel se toman los datos del almacenamiento y los convierte en datos estructurados para un análisis más sencillo. Los datos no estructurados se pasan de una base de datos MPP (proceso paralelo masivo) y de ahí a memoria, donde son procesados para volver a la base de datos MPP y se almacenan ya de forma estructurada en HDFS. En esta fase es donde se realiza el proceso ETL (Extract, Transform and Load), extraer, transformar y cargar.

5. **Nivel de procesamiento**. Los algoritmos analíticos y las consultas de los usuarios se realizan en tiempo real para generar datos estructurados para un análisis más comprensible. Es la fase en la que obtenemos valor a los datos.

6. **El nivel de operaciones**. En este nivel es donde se lleva la administración y la monitorización del sistema. Incluye auditoría, gestión de permisos, gestión de datos y gestión de flujo de trabajo.

El Data Lake es la pieza angular de un proceso Big Data, Hadoop ofrece diversas soluciones para inyectar, sacar, procesar, transformar, analizar y mantener los datos del Data Lake. Cada fabricante de una solución basada Apache Hadoop, se apoyará en diferentes herramientas de Apache Hadoop o desarrollará algunas propias para el producto que ofrece para Big Data.

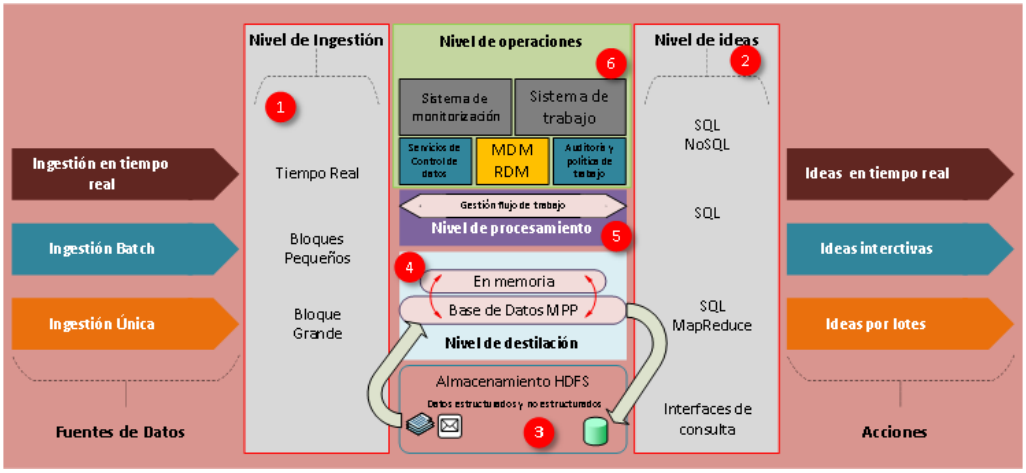

Figura 2.3: Arquitectura Data Lake

2.2 Apache Hadoop

Para poder comprender las arquitecturas que proponen los fabricantes de distribuciones de clúster para Big Data debemos familiarizarnos con ellas. En esta sección describiremos todas las que nos hemos encontrado que implementan las soluciones que hemos comparado.

2.2.1 Conceptos Principales de Hadoop

En Hadoop hay tres piezas básicas que pasaremos a describir, Hadoop Distributed File System (HDFS), Hadoop MapReduce y Hadoop Common.

Hadoop Distributed File System (HDFS)

HDFS [6] es un sistema de ficheros distribuido, escalable, escrito en Java y creado especialmente para trabajar con ficheros de gran tamaño. Es un sistema de archivos que sigue el patron “Write once read many” (escribe una vez y lee muchas) con un tamaño de bloque grande (64MB) frente a los 4 KB de un sistema de archivos tradicional. Estas dos características hacen que sea ideal para bases de datos orientadas al almacenamiento masivo de datos como por ejemplo, Hive o HBase. Que sea un sistema de archivos distribuido quiere decir que los ficheros serán divididos en bloques de un mismo tamaño y distribuidos entre los nodos que forman el clúster de datos, pero teniendo en cuenta que los bloques de un mismo fichero pueden caer en nodos distintos.

Un clúster HDFS tiene dos tipos de nodos, diferenciados completamente según el rol o la función que vayan a desempeñar a la hora de ser usados. Los dos tipos de nodos HDFS son los siguientes:

1. **Namenode (JobTracker)**. Este tipo de nodo es el más importante ya que es responsable de la topología de todos los demás nodos y, por consiguiente, de gestionar el

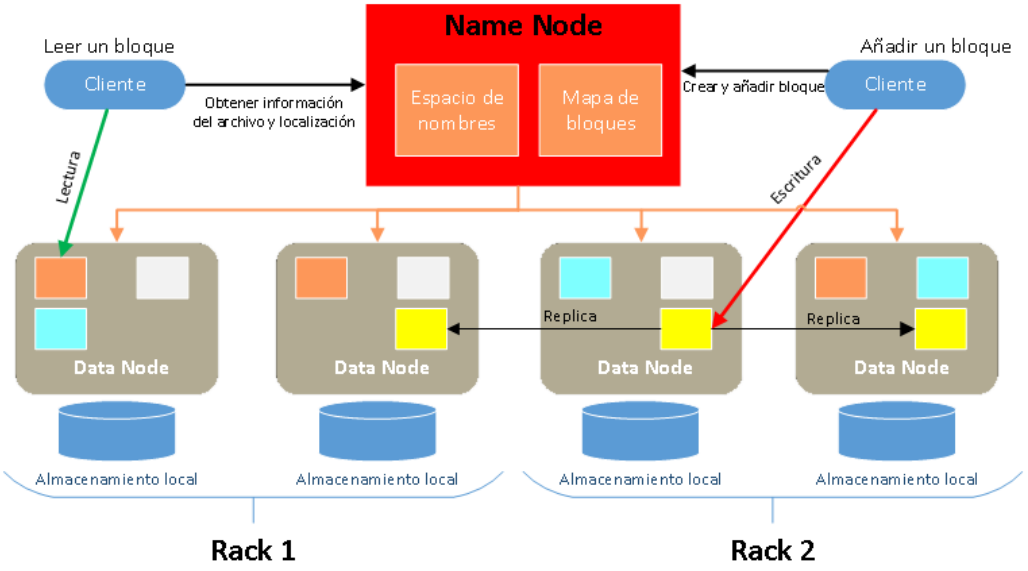

Figura 2.4: HDFS

espacio de nombres. Inicialmente HDFS estaba concebido para que sólo existiera uno NameNode aunque ahora puede haber más de uno si realizamos una instalación HA (High Ability). Cualquier cambio en el espacio de nombres del sistema de archivos o sus propiedades es registrado por el NameNode. Se puede especificar el número de réplicas que HDFS debe mantener de un archivo, por defecto es tres, en próximas versiones se podrá especificar el número de réplicas de un directorio, pero esta característica no está disponible por el momento para sistemas en producción. La cantidad de copias de un archivo se denomina factor de replicación de ese archivo.

2. **Datanodes (TaskTracker)**. Este tipo de nodos son los que realizan el acceso a los datos, almacenan los bloques de información y los recuperan bajo demanda.

Es importante comprender el funcionamiento básico del HDFS, puesto que es el substrato principal de un clúster Hadoop. Este sistema fue diseñado para aprovechar hardware de bajo coste, es tolerante a fallos y los nodos pueden estar distribuidos en chasis distintos y localizaciones distantes. El sistema intenta ubicar la computación donde se encuentran las porciones de los archivos, para evitar el trasiego de información entre Datanodes de distinto chasis o localizaciones lejanas. Las operaciones básicas son:

- **Operación de lectura**. Una consulta de lectura de un cliente será formulada al Namenode y éste responderá con la información y la localización del archivo. La secuencia de pasos concretos, ver Figura 2.5 serían:

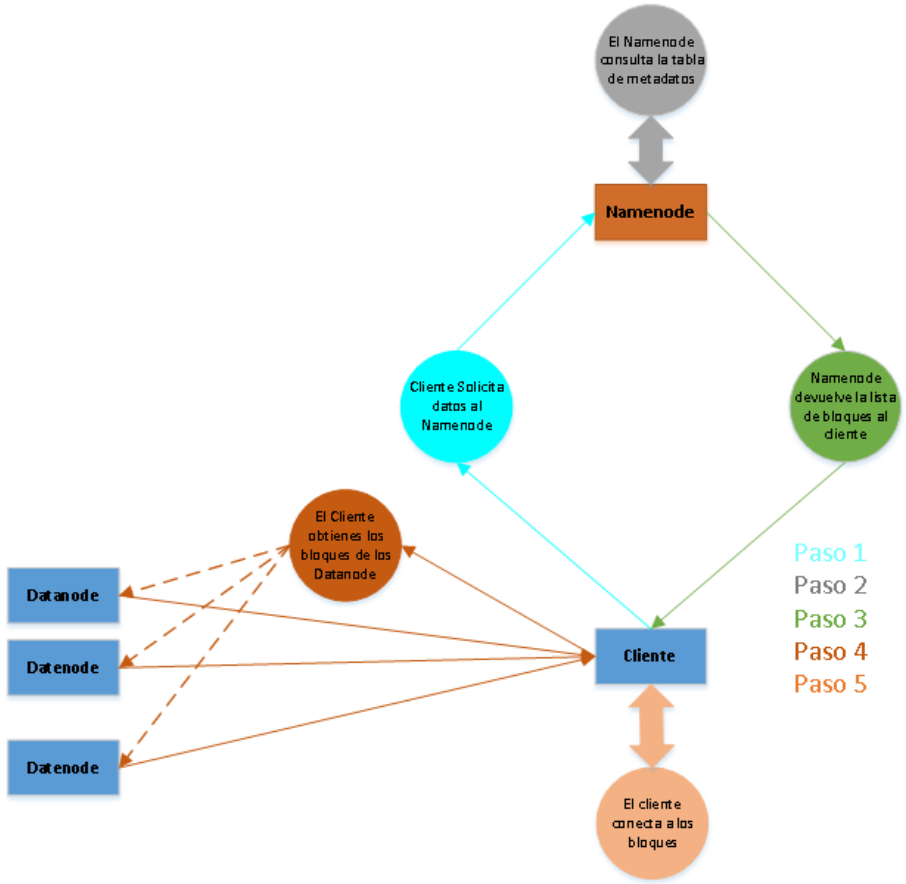

Figura 2.5: Lectura en HDFS

1. El cliente solicita al Namenode que le proporcione un fichero.
2. El Namenode consulta en la tabla de metadatos los bloques que componen el fichero y su localización en el clúster.
3. El Namenode devuelve al cliente una lista de bloques, así como los equipos en los que puede encontrar cada uno de ellos.
4. El cliente contacta con los Datanodes que le indicó el Namenode para obtener los bloques.
5. El cliente compone el archivo de bloques.

- **Operación de escritura**. Una operación de escritura igualmente se realizará al Namenode y este responderá con los Datanode donde se creará cada bloque. La secuencia es más complicada que en la lectura y se ilustra en la Figura 2.6:

1. El cliente solicita al Namenode realizar una escritura de un archivo.
2. El Namenode realiza algunas comprobaciones previas, para comprobar que se puede escribir el fichero y que el cliente tiene permisos para hacerlo.

3. El cliente divide el fichero en bloques, que escribirá de forma secuencial. Para cada bloque, el Namenode le proporciona una lista de Datanodes en los que se escribirá.
4. El cliente contacta con el primer Datanode para pedirle que escriba el bloque. Este Datanode es el encargado de propagar ese bloque al resto de Datanodes que deben contener las réplicas.
5. El último Datanode en escribir el bloque devolverá una confirmación (ACK) hacia atrás, hasta que el primer Datanode mandará la confirmación al cliente.
6. Una vez que el cliente ha escrito todos los bloques, manda una confirmación al Namenode de que el proceso se ha completado.

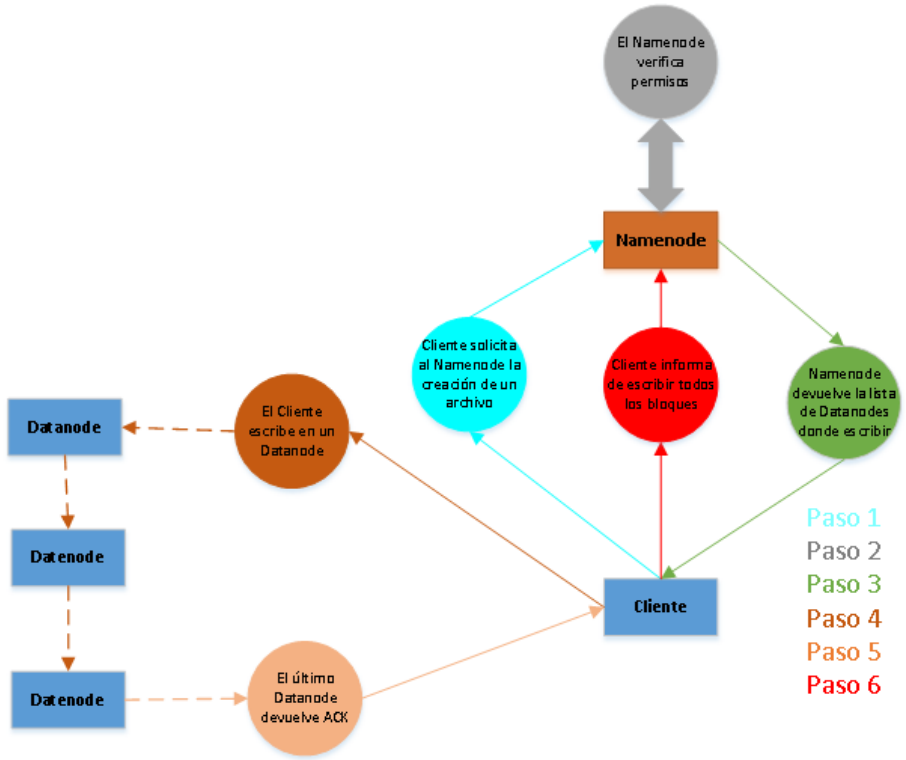

Figura 2.6: Escritura en HDFS

Es importante tener en cuenta que cada clúster tiene un solo NameNode, y si ese host o proceso no está disponible, el clúster en su totalidad no estará disponible hasta que el NameNode se reinicie o se inicie el roll en un nuevo host. Es un error común pensar que el SNN (Secundary NameNode) entrará a funcionar cuando caiga el primario, pero esto no es así. Es sólo una copia del NameNode primario, no proporciona la capacidad de conmutación por error. Por lo tanto, el NameNode es un punto de fallo único, SPO (single

point of failure). A parte de que el nodo que mantiene el roll de NN (NameNode) tiene mucha sobrecarga, ya que es preguntado continuamente por el resto de nodos, servicios y clientes. La alta disponibilidad en HDFS permite tener tan solo dos NameNodes en modo activo/pasivo y con esto solventamos los dos problemas que puede presentar el NN:

1. En caso de que el NameNode deje de dar servicio por un bloqueo del host.

2. Por tener que realizar mantenimiento en el NameNode, ya sea por actualizaciones software o hardware.

Hadoop MapReduce

Hadoop MapReduce [26] es un sistema basado en YARN para el procesamiento en paralelo de grandes conjuntos de datos. Hadoop YARN (Yet Another Resource Negotiator), es una tecnología de administración de clústeres. Cuando se separó MapReduce de HDFS, gracias a YARN se consiguió que Hadoop fuera más adecuado para las aplicaciones que no pueden esperar a que terminen los trabajos por lotes. Este conjunto también es llamado MapReduce 2.0, su arquitectura la podemos ver en la Figura 2.7.

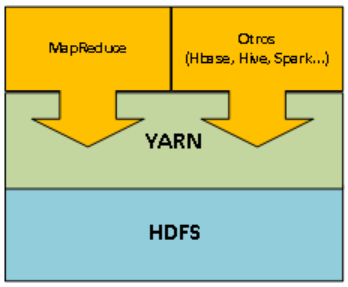

Figura 2.7: Pila YARN

El paradigma de MapReduce se basa en enviar el proceso computacional al sitio donde residen los datos que se van a tratar, los datos residen distribuidos en el clúster Hadoop. Cuando se lanza un proceso de MapReduce, se distribuyen las tareas entre los diferentes nodos del clúster, es el propio framework Hadoop quien gestiona el envío y recepción de datos entre nodos. La mayoría de la carga computacional sucede en los nodos que tienen los datos en local, para minimizar el tráfico de red. En la Figura 2.8 podemos ver este flujo de información, los datos entran en el clúster, se distribuyen en los nodos del clúster, en cada nodo se combinan, se mezclan y ordenan y posteriormente pasan a reducirse para enviarse a la salida. Una vez se han procesado todos los datos, el usuario recibe el resultado del clúster. El proceso consiste en dos fases:

- **Map**. Proceso de distribución de las tareas.
- **Reduce**. proceso de mezclado de datos y reducción.

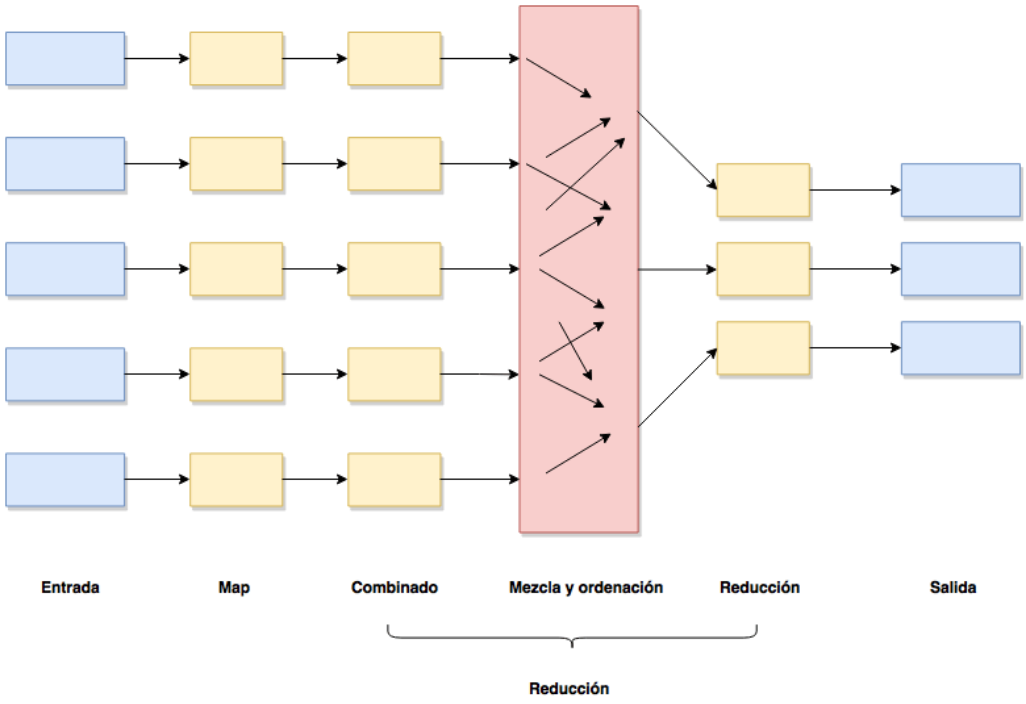

Figura 2.8: Flujo MapReduce

En la Figura 2.9, podemos ver el proceso MapReduce con un ejemplo sencillo. En el ejemplo se quiere contar palabras, y se dispone de dos nodos, cada nodo cuenta las suyas y en el proceso de reducir la tarea consiste en que cada nodo se quede con un conjunto ordenado de palabras y su contador.

Hadoop Common

Hadoop Common contiene los scripts necesarios para inicial Hadoop y las utilidades comunes en las que se apoyan los otros módulos. Proporciona abstracciones a nivel de sistema de archivos o sistema operativo.

2.2.2 Herramientas de Hadoop

En esta sección veremos las diferentes herramientas de Apache Hadoop, profundizando en aquellas que desplegaremos, como ZooKeeper, Hive, HBase, Flume, Hue, Oozie, Spark, Solr que es parte de Lucene, Kafka, Pig y Sqoop, y serán las que describiremos primero.

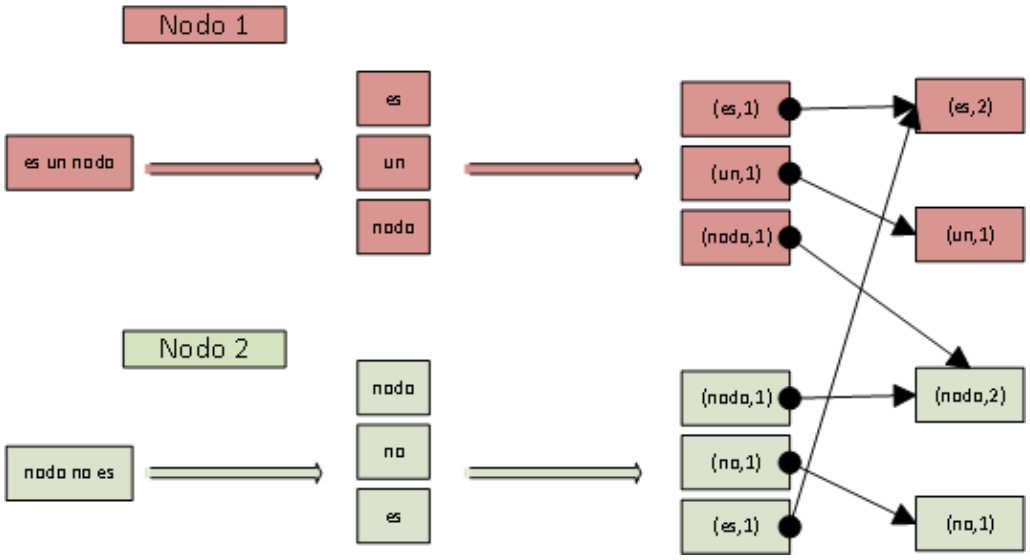

Figura 2.9: Ejemplo MapReduce

Apache ZooKeeper

Apache ZooKeeper [19], Figura 2.10, es un proyecto de software libre de la Apache Software Foundation[1] que ofrece un servicio para la coordinación de procesos distribuido y altamente confiable, que resuelve varios problemas de coordinación para grandes sistemas distribuidos. Este tipo de servicios deben garantizar los siguientes aspectos:

Figura 2.10: ZooKeeper

- **Secuencialidad:** Asegurar que las operaciones de actualización se aplican en el mismo orden en el que se envían.
- **Atomicidad:** No existen los resultados parciales, las actualizaciones fallan o son un éxito. La atomicidad desde el punto de vista del cliente caracteriza al servicio como:
  - **Único endpoint:** Los clientes verán siempre el mismo servicio, independientemente del servidor que lo contenga.
  - **Seguridad:** En el momento en el que una actualización se ha aplicado, dicha modificación se mantendrá hasta el momento en que un cliente la sobrescriba.
  - **Actualización:** El cliente tiene garantizado que, en un determinado intervalo temporal, los servicios estarán actualizados.

Las principales características que tiene Zookeeper [14] son:

- **Sencillo:** La coordinación entre los procesos distribuidos se organiza mediante un namespace jerárquico muy similar a un sistema de archivos. El namespace consiste

[1] https://www.apache.org/

en registros (Znodos) similares a ficheros o directorios. ZooKeeper mantiene la información en memoria permitiendo obtener latencias bajas y rendimientos altos y cada nodo mantiene una copia.

- **Replicable:** Permite las réplicas instaladas en múltiples hosts, llamados conjuntos o agrupaciones, para ello todos los nodos que forman parte del servicio Zookeeper se deben conocer entre ellos. Cada nodo mantiene una imagen en memoria con el estado y un log de transacciones, así como los snapshots almacenados en un disco. Por lo tanto, mientras la mayoría de los nodos que mantienen ZooKeeper estén activos el servicio se mantendrá disponible.
- **Ordenado:** ZooKeeper se encarga de mantener la traza de todas las operaciones realizadas, marcando cada petición con un número que refleja su orden entre todas las transacciones.
- **Rápido:** dado que mantiene la copia en memoria, si no hay muchas operaciones de escritura el servicio es muy rápido.

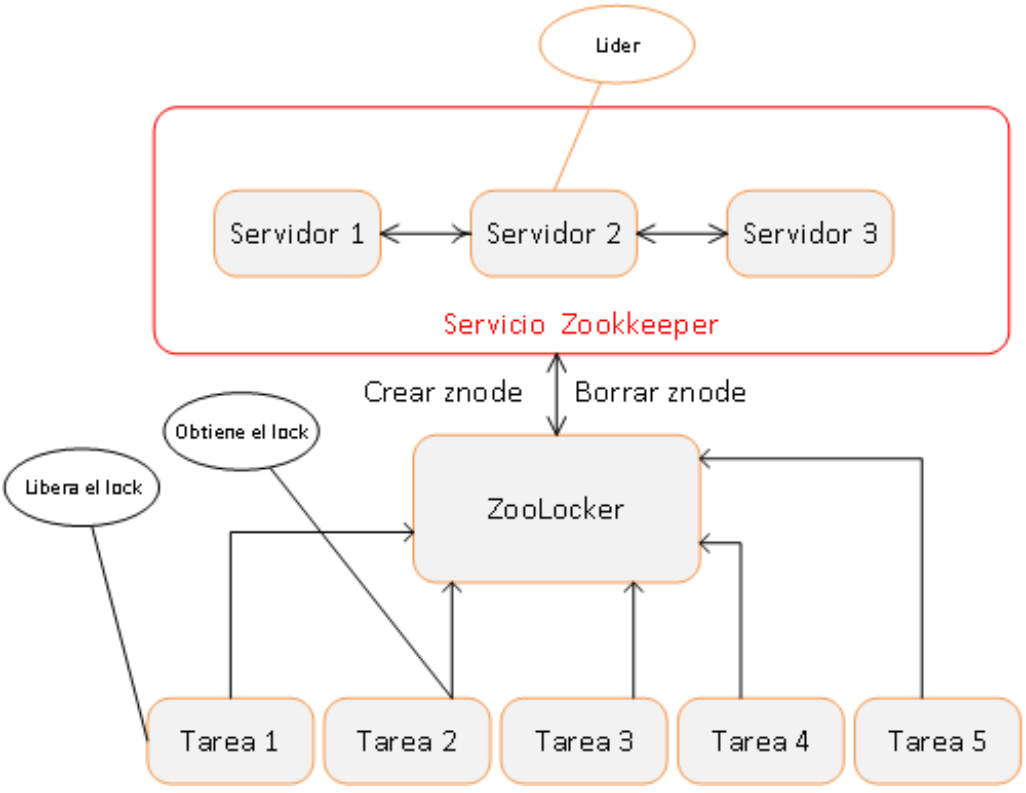

Figura 2.11: Zookeeper

Los nodos que mantienen el servicio Zookeeper (Znodos) mantienen información estadística que incluye el número de versión de la base de datos, cambios ACL (Access Control List) y timestamps que permitirán crear validaciones de la caché que reside en

memoria y permite que las actualizaciones sean coordinadas. Cada vez que la información de un Znodo se modifica, su número de versión aumenta y así se sabe qué Znodos están actualizados. La base de datos que se encuentra replicada en cada servidor, contiene el árbol completo de Znodos. Cuando se reciben actualizaciones en primer lugar se almacenan serializadas en disco para poder recuperar el estado, en caso necesario y, con posterioridad, se aplican sobre la base de datos. Todos los servidores ZooKeeper proveen a los clientes las tareas, pero un cliente se conecta únicamente a un servidor para realizar las peticiones. Las operaciones que se pueden realizar son dos:

- **Lectura**. Las peticiones de lectura se pueden responder desde cualquier Znodo ya que disponen de una réplica local de la base de datos.
- **Escritura**. Estas operaciones se procesan usando el protocolo de aceptación. Estas operaciones son enviadas al Znodo "líder", el resto de los servidores se denominan "seguidores" reciben las peticiones de mensajes del líder y aceptan acordar la entrega de mensajes. La capa de mensajería de Zookeeper se encarga de reemplazar al líder en caso de error y sincronizar los seguidores con el líder en las operaciones de escritura. Cómo se ve en la Figura 2.11 las tareas de escritura hacen que el resto sean encoladas en el log y cuando una libera el bloque se pasa a la siguiente, si esta es de escritura vuelve a obtener el bloqueo y si es de lectura lo libera.

Apache Hive

Apache Hive [7], Figura 2.12, es un infraestructura originalmente creada por Facebook que sirve para trabajar con HDFS. Su funcionamiento es sencillo, ya que podemos lanzar consultas similares a SQL (HiveQL) que serán traducidas a trabajos MapReduce. Hive traduce consultas tipo SQL a trabajos MapReduce paralelizables contra el dataset en un entorno completamente distribuido. La tarea de traducir la sentencia HQL (lenguaje de la interfaz HiveQL) a lenguaje Java para crear un trabajo MapReduce, hace que la latencia sea alta, quizás sea su mayor pega junto con que no soporta todo el lenguaje SQL, transacciones ni índices. En el fondo Hive no es una base de datos, lo cierto es que se comporta como si lo fuera. Para entender qué es Hive es importante que conozcamos que este sistema cuenta con tres formatos diferentes para la organización de los datos, tablas, particiones y los cubos.

Figura 2.12: Hive

- **Tablas**. Las tablas de Hive son muy parecidas a los sistemas de bases de datos relacionales clásicos que presentan tablas y filas. Para trabajar con tablas lo que hacemos es asignar cada una de estas tablas a los directorios que contienen los

sistemas de archivos, un proceso que se lleva a cabo de forma directa. Hive es compatible con otros sistemas, no solo con HDFS.

- **Particiones**. Las particiones se realizan en las propias tablas, las tablas Hive pueden tener más de un fraccionamiento. Cada partición determina como se guardan los datos. Aparte de ser unidades de almacenamiento, también permite a los usuarios identificar eficazmente las filas que necesita.
- **Cubos**. Los cubos, o buckets, es otra técnica para descomponer conjuntos de datos en partes más manejables. Cuando se tiene un campo con una alta cardinalidad podemos crear un clúster para agruparla y de esta forma manejarla de forma más eficiente. Un ejemplo típico son los pedidos de una empresa. Si particionamos por el identificador de pedido tendríamos millones de particiones. Pero si las agrupados por sectores la información la podremos manejar mejor. En resumen, los datos de una partición pueden ser divididos en paquetes utilizando una función hash sobre el valor de alguna columna.

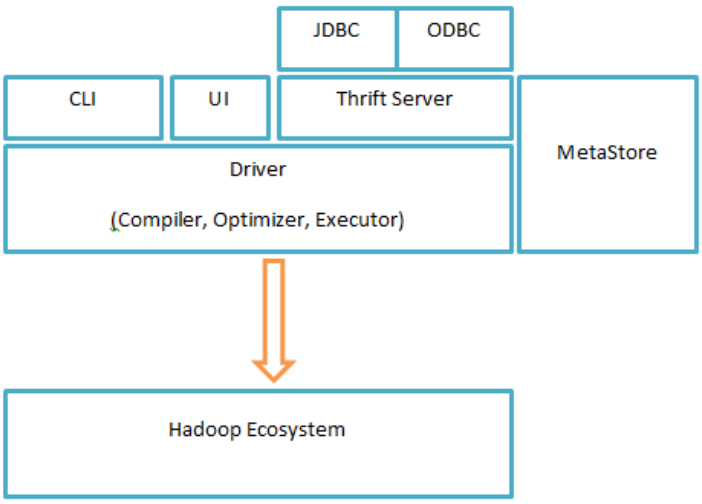

Figura 2.13: Arquitectura Hive

Por lo tanto, antes de poder hacer consultas con Hive hay que crear esquemas sobre los datos que trabajaremos, definir la tabla con sus columnas y tipo de dato de cada una de ellas, hacer la carga de los datos inicial y de esta forma podremos realizar consultas y obtener los resultados. Ya que Hive se comporta casi como un sistema de bases de datos relacional, se considera que es un componente Data Warehouse del ecosistema Hadoop.

En la Figura 2.13 podemos ver los componentes principales de Hive que son los siguientes:

- **Compiler**, realiza la compilación de la consulta HiveQL dando como resultado un plan de ejecución. Es el proceso que más carga lleva. El compilador convierte la consulta a un árbol de sintaxis abstracta (AST: Abstract syntax tree). Después de comprobar la compatibilidad y errores de compilación del AST lo convierte en un grafo dirigido acíclico (DAG: Directed Acyclic Graph). El DAG divide los operadores para la etapas MapReduce basándose en la consulta de entrada y en los datos.
- **Optimizer**, encargado de optimizar el DAG. Puede dividir las tareas o aplicar una transformación a los datos antes de que se envíen a las operaciones de Reduce para mejorar en rendimiento y escalabilidad.
- **Driver**, recibe y coordina las sentencias HiveQL. Es el encargado de comenzar la ejecución de las sentencias creando sesiones, controla el ciclo de vida y progreso de la ejecución. Otra tarea que realiza es almacenar los metadatos necesarios generados durante la realización de la instrucción HiveQL y recoger los datos después de la realización de una tarea.
- **Metastore**, guarda los metadatos para cada una de las tablas, así como sus esquemas y localización. También incluye una partición de metadatos que ayuda al Driver a rastrear los conjuntos de datos distribuidos sobre el clúster. Los metadatos son almacenados en un sistema de base de datos relacional tradicional. Se realiza una copia de seguridad regularmente en un servidor remoto para que pueda ser utilizado en caso de que haya perdida de datos.
- **Motor de ejecución**, realiza las tareas acordes al DAG, se ocupa de que el orden sea correcto y si alguna tarea depende del resultado de otra esta no se lance antes de tiempo, para ello interactúa con el Resource Manager.
- **CLI**, Hive dispone de su propio CLI (Command Line Interface) llamado Beeline, es un cliente JDBC basado en SQLLine.

Apache HBase

HBase [15], Figura 2.14, es un proyecto Open Source mantenido por la Apache Foundation que proporciona una base de datos columnar, distribuida, no relacional, de código abierto escrita en Java, creada sobre el sistema de ficheros de Hadoop, que puede escalar horizontalmente. Se creó a partir de Google BigTable que es una base de datos tolerante a fallos, distribuida y propietaria de Google. HBase guarda las tablas almacenando los datos por pares clave-valor manejando así millones de columnas por billones de filas. Buscar por clave en HBase es muy rápido y la escritura también ya que se realiza en memoria. La arquitectura de HBase, Figura 2.15, consta de dos componentes

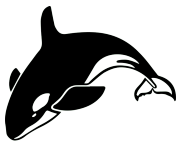

Figura 2.14: HBase

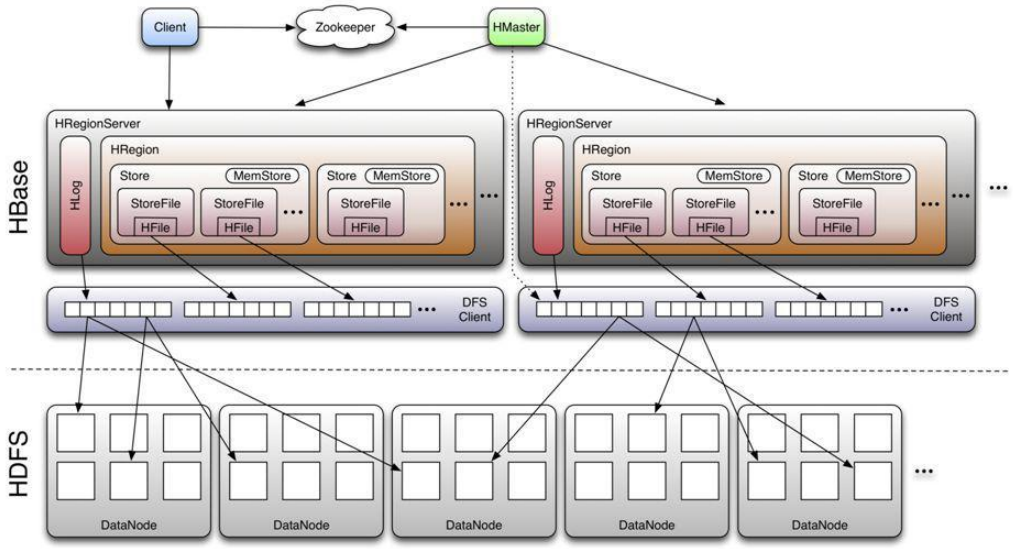

Figura 2.15: Arquitectura HBase

principales HMaster Server , HBase Region Server, la coordinación de tareas las realiza el HMaster Server con ayuda de Zookeeper [31].

- **HMaster Server**, realiza operaciones de DDL (crear y eliminar tablas) y asigna regiones a los Regions Servers. Se encarga de asignar regiones a los Regions Server y los coordina de una forma similar a como el Namenode administra los DataNode en HDFS. Dada la enorme carga de trabajo y supervisión que debe realizar, se ayuda de Zookeeper para realizar las tareas.

- **Region Server**, mantiene varias regiones que se ejecutan en la parte superior de HDFS. Está compuesto por cuatro componentes, Wal, Block Cache, MemStore y HFile. **WAL** almacena los nuevos datos que no se han conservado o comprometido con el almacenamiento permanente. **Block Cache**, almacena los datos de lectura frecuente en la memoria. **MemStore**, almacena todos los datos entrantes antes de enviarlos al disco o a la memoria permanente, es la caché de escritura. **HFile** está almacenado en HDFS. Por lo tanto, almacena las celdas reales en el disco. MemStore confirma los datos que almacena a HFile para mantener la coherencia entre los datos de HFile y MemStore.

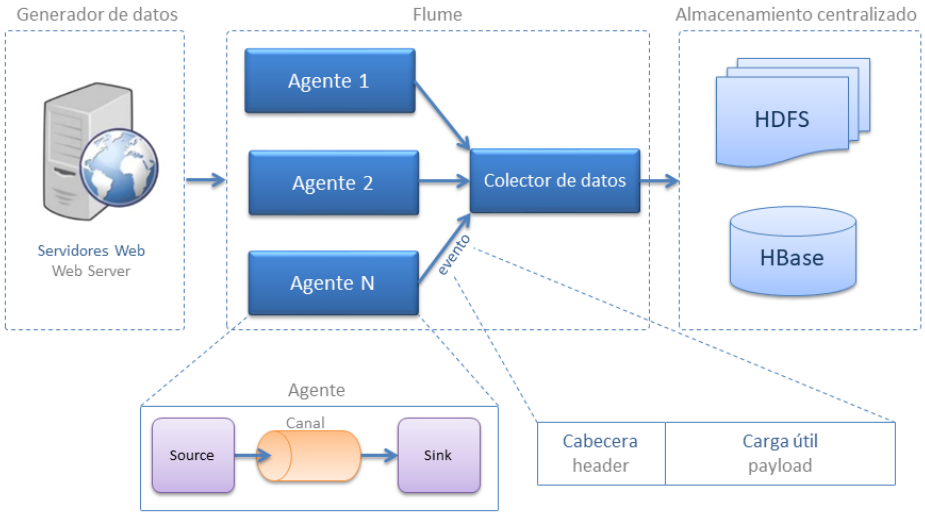

Figura 2.17: Arquitectura Flume

Apache Flume

Apache Flume [5], Figura 2.16, es un servicio distribuido, con una arquitectura sencilla y flexible basada en flujos de datos en streaming. Permite construir múltiples flujos coordinados por los que viajan los eventos a través de diferentes agentes hasta que alcanzan el destino. Es el encargado de recopilar, agregar y mover de forma eficiente grandes cantidades de datos.

Figura 2.16: Flume

Para trabajar con Flume hay que entender los tres conceptos sencillos:

- **Orígenes**, cualquier origen de datos que de soporte a Flume, por ejemplo, stream de vídeo o un stream de sensores, etc.
- **Canales**, es un repositorio donde se guardan los datos temporalmente.
- **Receptores**, el destino de los datos a los que Flume envía lo almacenado en el canal.

Para entender la arquitectura Flume utilizaremos el ejemplo de la Figura 2.17, el caso típico de recoger los log de un servidor o una granja de servidores Web. Los elementos de la arquitectura Flume son [3].

- **Agente**, es un proceso independiente, se encarga de recibir, guardar y enviar eventos. Hay uno en cada nodo del clúster que disponga del servicio Flume. El agente está compuesto de dos partes:
  - **Source**, es una interfaz que consume eventos de entrada que le son enviados por un mecanismo, este puede ser un flujo web, o stream de vídeo, y los pone en el canal. Apache Avro es un ejemplo de origen.

  - **Canal**, es un almacén transitorio que dispone el agente donde almacena los eventos hasta que son entregados a un receptor.
  - **Sink**, es una interfaz que permite consumir eventos de un canal y transmitirlos al siguiente agente o directamente a un destino final como puede ser Flume HDFS.

- **Evento unidad de datos Flume**, cuenta con una cabecera que es opcional y un conjunto de bytes.
- **Flujo de datos**, describe el movimiento de eventos del origen al destino.
- **Cliente**, interfaz que opera en el origen de los eventos y los entrega a los agentes.

Apache Hue

Figura 2.18: Hue

Hue [22] (Hadoop User Experience), Figura 2.18, es un proyecto de Cloudera [8] que se abrió como proyecto Open Source bajo licencia Apache License, version 2.0. Es una interfaz de usuario web para la gestión de Hadoop, además de una plataforma para construir aplicaciones a medida sobre esta librería UI (Interfaz de Usuario). Esta interfaz de usuario se dirige al desarrollador de aplicaciones de datos ofrece varios intérpretes para conectarse a las bases de datos más comunes, como Apache Hive, Apache Impala, Apache Presto y base de datos SQL como MySQL, SparkSQL, Oracle, Apache Phoenix, Apache Drill, Apache Kylin, PostgreSQL, Redshift, BigQuery ...

Hue es una interfaz visual que se proporciona acceso a una gran cantidad de almacenes de datos, herramientas avanzadas de BI y algoritmos de aprendizaje automático. En la Figura 2.19 se puede ver el aspecto de la interfaz de usuario. Hue es una plataforma de desarrollo para Expertos TI (Tecnología de la Información), además de ser una herramienta para que más personas aprovechen la plataforma de datos para responder preguntas empresariales básicas, por ejemplo, cuántas visitas recibimos cada día, sin la necesidad de Expertos de TI.

Esta herramienta dispone de un Smart Query Editor y un asistente, que facilita la escritura de programas, emite alertas de autocompletar y corrige convirtiéndose en editor eficiente de consultas.

Permite la programación de flujos de datos y tareas, así como almacenar y compartir consultas que pueden ser usadas en herramientas BI (Business Intelligence) o para la ingesta de datos de un flujo. Otra de sus características es que permite el etiquetado y búsqueda de datos, así como la importar tablas externas. Permite catalogar la información para poder realizar búsquedas entre miles de tablas la documentación y el código generado.

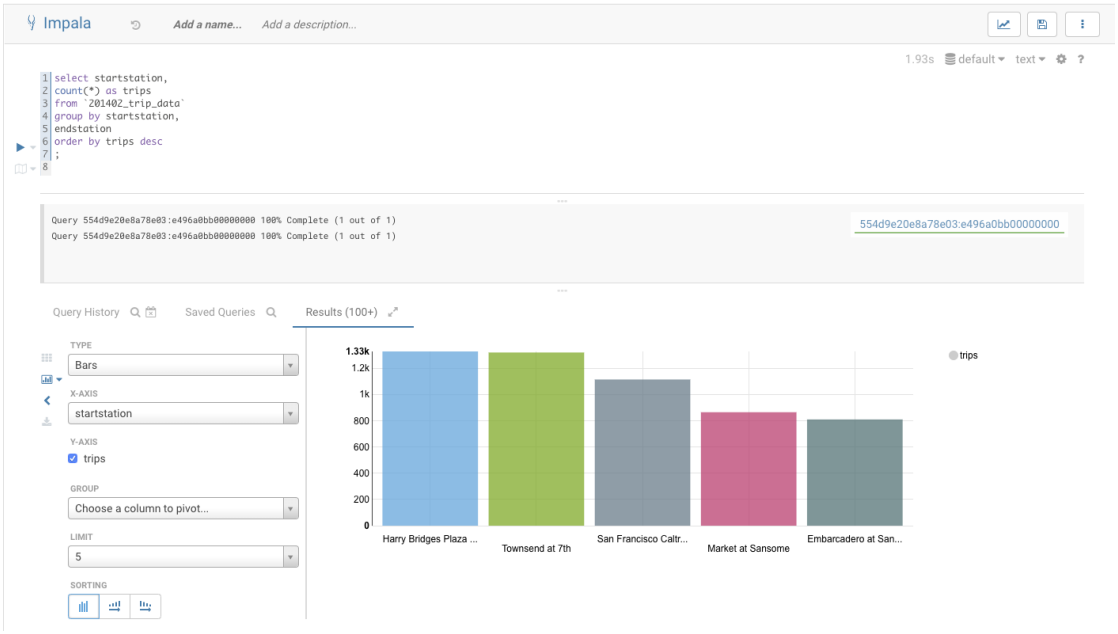

Figura 2.19: Interfaz de Usuario de Hue

Apache Oozie

Oozie [10], Figura 2.20, es el planificador de flujos de trabajo para Hadoop. Oozie define una unidad lógica que llama workflow en la que podemos incluir cualquier tipo de tarea que necesitemos.

Figura 2.20: Oozie

All estar completamente integrado con los demás proyectos de Apache Hadoop, nos permite incluir en los workflow tareas MapReduce, Streaming MapReduce, Pig, Hive o Sqoop, así como programas en Java o scripts shell.

Oozie se ejecuta como un grafo acíclico dirigido DAG (Directed Acyclic Graph), lo que significa que no existen bucles dentro del grafo del workflow, por lo cual, todas las tareas y dependencias se ejecutan de inicio a fin en un solo camino permitiendo una ejecución más limpia de los procesos.

Apache Spark

Apache Spark [13], Figura 2.21, es un motor de análisis para el procesamiento de datos en memoria a gran escala y de propósito general. Apache Spark se está posicionado para reemplazar a MapReduce como el motor de procesamiento de datos predeterminado en el ecosistema de Hadoop, pero por el momento no

Figura 2.21: Spark

se ha implantado a nivel empresarial. Al ser un lenguaje de propósito general dispone de multitud de librerías para procesamiento en batch y streaming, algoritmos iterativos, consultas interactivas, etc. Incluye APIs de Python, Java, Scala, SQL y R, con funciones

integradas y en general con un rendimiento razonablemente bueno en todas ellas. Spark puede ejecutarse en clúster Hadoop y acceder a los datos almacenados en HDFS y otras fuentes de datos de Hadoop, como Cassandra, Hbase, Kafka, etc. Spark trabaja con RDDs, que se define como una colección de elementos que es tolerante a fallos y que es capaz de operar en paralelo. Es la principal abstracción para trabajar con Spark. Los RDDs están particionados en distintos nodos del clúster y se pueden crear a partir de ficheros HDFS.

Apache Solr

Solr [12], Figura 2.22, es un motor de búsqueda de código abierto basado en la biblioteca Java del proyecto Lucene. Es un potente motor de indexado y búsqueda de contenido. Solr permite definir la estructura de los índices de forma declarativa, indicando los campos, tipos de los campos y análisis a utilizar para estos tipos de campos mediante un archivo de esquema. Las características principales de Solr son, que es extensible mediante plugins y componentes, es escalable y distribuido y tolerante a fallos gracias a Zookeeper.

Figura 2.22: Solr

Apache Kafka

Kafka [9] es un servicio que proporciona una plataforma unificada, de alto rendimiento y de baja latencia para la manipulación en tiempo real de fuentes de datos. Esta plataforma de streaming tiene tres capacidades clave:

Figura 2.23: Kafka

- Publicar y suscribirse a flujos de registros, similar a una cola de mensajes o un sistema de mensajería.
- Almacenar flujos de registros de forma duradera y tolerante a fallos.
- Procesar flujos de registros a medida que se producen.

Kafka se usa generalmente para dos clases de aplicaciones:

- Construcción de flujos de datos en tiempo real para obtener datos de manera fiable entre sistemas o aplicaciones.
- Creación de aplicaciones de transmisión en tiempo real que transforman o reaccionan a los flujos de datos.

Apache Pig

Apache Pig, Figura 2.24 es una plataforma de alto nivel para crear programas MapReduce utilizados en Hadoop bajo la licencia Apache 2.0. Aunque la programación de aplicaciones MapReduce no es excesivamente compleja, se necesita experiencia en desarrollo de software para hacer los programas eficientemente. Apache Pig modifica esta forma de programar ya que crea una abstracción de lenguaje de procedimental que es más simple que MapReduce. Esta abstracción es más parecida al SQL para sistemas de gestión de bases de datos relacionales que a las aplicaciones Hadoop, sin llegar a ser un lenguaje declarativo como el SQL. Entonces, en vez de escribir una aplicación de MapReduce, se puede escribir un único script en Pig Latin que se paraleliza automáticamente paralelizado y distribuido a través de un clúster. El lenguaje de esta plataforma es llamado Pig Latin.


Figura 2.24: Pig

Apache Sqoop

Sqoop, Figura 2.25, es un proyecto que tiene gran éxito en el mundo empresarial. Permite transmitir grandes volumenes de datos de manera eficiente entre Hadoop y sistemas de bases de datos relacionales o Data Warehouse, y desde sistemas de bases de datos relacionales o Warehouse a Hadoop. Sqoop dispone de múltiples conectores para integrar Hadoop con otros sistemas como puede ser Db2, Oracle, MariaDB, Oraculo, SQL Server, Postgress, etc.


Figura 2.25: Sqoop

Apache Impala

Impala, Figura 2.26 es un motor de consulta de procesamiento masivo en paralelo. Con esta definición podíamos pensar que similar a HBase, pero ofrece características casi de tiempo real en sus consultas gracias a su baja latencia. Para evitar la latencia, Impala evita que MapReduce acceda directamente a los datos y lo hace a través de un motor de consulta distribuido especializado que es muy similar a los que se encuentran en los RDBMS paralelos comerciales. El resultado es un rendimiento de orden de magnitud más rápido que el de Hive, según el tipo de consulta y configuración.


Figura 2.26: Impala

Trabaja directamente sobre los datos almacenados en HDFS o en HBase. Impala se integra con Hive para el procesamiento o para crear tablas en un solo sistema de archivos compartidos HDFS sin ningún cambio en la definición de la tabla. El lenguaje se llama ImpalaQL que es un subconjunto de HiveQL, con algunas limitaciones funcionales como

las transformaciones. Apache Impala es un motor SQL de procesamiento masivo en paralelo para análisis interactivo e inteligencia de negocios. Su arquitectura está muy optimizada, esto hace que sea ideal para consultas de estilo BI tradicional, con uniones, agregaciones y subconsultas.

HCatalog

Apache HCatalog es una capa de gestión de almacenamiento para Hadoop que ayuda a los usuarios de diferentes herramientas de procesamiento de datos en el ecosistema de Hadoop, como Hive, Pig y MapReduce, a leer y escribir fácilmente datos del clúster. HCatalog permite una vista relacional de los datos de formato RCFile (Record Columnar File), ORCFile (Optimized Row Columnar), Parquet o archivos secuenciales almacenados en HDFS. También dispone una API REST a sistemas externos para acceder a los metadatos, con este API Rest podemos crear tablas o realizar otras operaciones. En resumen, como muestra la Figura 2.27, HCatalog es una interfaz entre Hive, Pig, Mapreduce y diferentes formatos de almacenamiento de datos.

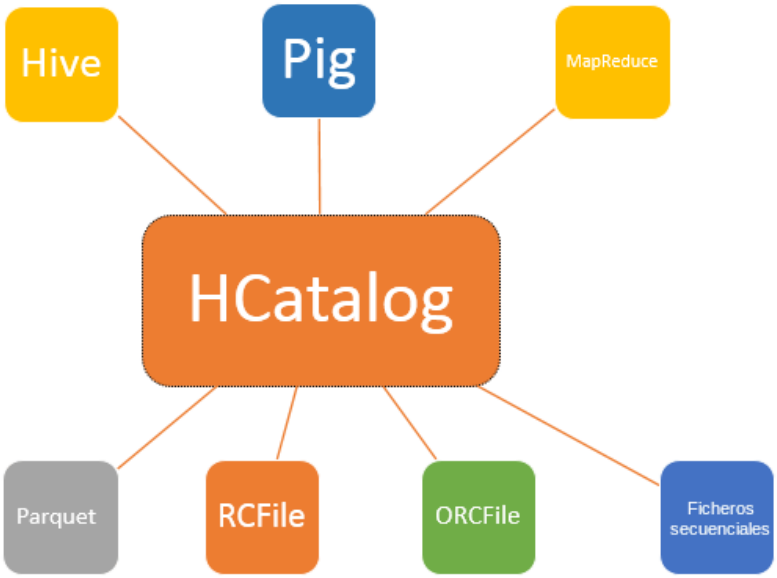

Figura 2.27: HCatalog

Apache Lucene

Apache Lucene, Figura 2.28, es una biblioteca de motores de búsqueda de texto de alto rendimiento, es una tecnología adecuada para casi cualquier aplicación que requiera una búsqueda de texto completo. Lucene ofrece potentes funciones a través de una API simple, necesita pocos recursos de memoria y dispone muchos tipos de consultas potentes, como consultas por

Figura 2.28: Lucene

frases, por proximidad o consultas de rango, todas ellas específicas para búsquedas en texto.

Apache Avro

Avro, Figura 2.29, es un sistema de serialización de datos, el formato que utiliza para serializar es JSON gracias a su portabilidad y fácil lectura. Avro se ocupa de que los distintos formatos que podemos encontrar en Hadoop puedan ser procesados por lenguajes de programación como por ejemplo Java, C, C++, Python, Ruby y C#. Los datos de Avro se almacenan en un archivo, su esquema se almacena con él, por lo que cualquier programa puede procesarlo posteriormente. Si el programa que lee los datos espera un esquema diferente, esto se puede resolver fácilmente, ya que ambos esquemas están presentes.

Figura 2.29: Avro

Jaql

Jaql, Figura 2.30, es un lenguaje de consulta y procesamiento de datos funcional que se usa comúnmente para el procesamiento de consultas JSON en BigData. Jaql posee una infraestructura flexible para administrar y analizar datos semiestructurados como XML, JSON, archivos CSV o datos relacionales. El motor Jaql transforma la consulta en procesos MapReduce.

Figura 2.30: Jaql

Cascading

Cascading, Figura 2.31, proporciona una capa de abstracción sobre Hadoop y permite aprovechar las habilidades y los recursos existentes para crear aplicaciones de procesamiento de datos en Apache Hadoop, sin habilidades de Hadoop especializadas. Cascading es una API de procesamiento de datos y un planificador de consultas que se utiliza para definir, compartir y ejecutar flujos de trabajo de procesamiento de datos en un solo nodo informático o clúster distribuido. En un clúster de computación distribuida que utiliza la plataforma Apache Hadoop, Cascading agrega una capa de abstracción sobre la API de Hadoop, lo que simplifica enormemente el desarrollo de la aplicación Hadoop, la creación de trabajos y la programación de trabajos.

Figura 2.31: Cascading

Apache Drill

Drill, Figura 2.32, es un motor de consulta SQL de código abierto de Apache para la exploración de Big Data. Drill proporciona integración plug-and-play con las implementaciones existentes de Apache Hive y Apache HBase. Apache Drill admite una varie-

Figura 2.32: Drill

dad de bases de datos y sistemas de archivos NoSQL, como HBase, MongoDB, MapR-DB, HDFS, MapR-FS, Amazon S3, Azure Blob Storage, Google Cloud Storage, Swift, NAS y archivos locales. Una sola consulta puede unir datos de múltiples almacenes de datos. La ventaja de Drill es que no es necesario cargar los datos, crear y mantener esquemas, o transformar los datos antes de que puedan procesarse. En su lugar, simplemente se incluye la ruta a un directorio de Hadoop, una colección MongoDB o un grupo de S3 en la consulta SQL. Soporta SQL estándar por lo que se puede conectar a la mayoría de las herramientas BI del mercado.

Apache Vaidya

Vaidya es una herramienta de diagnóstico de rendimiento basada en reglas para los trabajos de MapReduce. Realiza un análisis de ejecución posterior del MapReduce mediante la recopilación de estadísticas de ejecución a través del historial de trabajo y los archivos de configuración del trabajo. Sirve para diagnosticar problemas de rendimiento. Cada regla de prueba que se define detecta un problema de rendimiento específico con el trabajo de MapReduce y proporciona un consejo al usuario.

Apache Mahout

Apache Mahout™, Figura 2.33, es un marco de álgebra lineal distribuida y Scala DSL (Domain Specific Languages) diseñado para permitir que matemáticos, estadísticos y científicos de datos implementen fácilmente algoritmos. Permite soluciones modulares para la aceleración de CPU/GPU/CUDA. Apache Mahout es un proyecto de Apache Software Foundation para producir implementaciones de algoritmos de aprendizaje automático distribuidos o escalables enfocados principalmente en las áreas de filtrado colaborativo, agrupación y clasificación. Es una herramienta para crear sistemas de aprendizaje automático (Machine Learning) y Data Mining escalables.

Figura 2.33: Mahout

Apache Tez

El proyecto Apache TEZ, Figura 2.34, tiene como objetivo facilitar la construcción de un grafo acíclico de tareas para procesar datos que actualmente en Apache Hadoop se construye sobre YARN. Los objetivos de Tez son, proporcionar una API sencilla para definir flujos de datos, simplificar el despliegue, gestión óptima de recursos y poder tomar decisiones sobre la gobernanza de los datos. Algunas distribuciones de Hadoop implementan Apache Tez en lugar de Apache Oozie.

Figura 2.34: Tez

Ganglia

Ganglia [20], Figura 2.35, es un sistema de monitorización distribuido escalable para sistemas de computación de alto rendimiento como clústeres. Emplea tecnologías ampliamente utilizadas, como XML para la representación de datos, XDR para el transporte de datos compacto y portátil, y RRDtool para el almacenamiento y visualización de datos.

Figura 2.35: Ganglia

Apache Falcon

Falcon, Figura 2.36, simplifica el desarrollo y la administración de las tuberías de procesamiento de datos con un nivel más alto de abstracción, eliminando la compleja codificación de las aplicaciones de procesamiento de datos al proporcionar servicios de administración de datos listos para usar. Esto simplifica la configuración y la organización del movimiento de datos, la recuperación de desastres y los flujos de trabajo de retención de datos.

Figura 2.36: Falcon

Falcon esencialmente transforma las configuraciones de alimentación y proceso del usuario en acciones repetidas a través de un motor de flujo de trabajo estándar. Falcon por sí mismo no hace ningún trabajo pesado. Todas las funciones y los requisitos de administración del estado del flujo de trabajo se delegan al programador del flujo de trabajo. Lo único que mantiene Falcon son las dependencias y la relación entre estas entidades.

Apache Uima

Apache Uima, Figura 2.37, proporciona herramientas para el análisis de contenido no estructurado, como texto, audio y vídeo, datos de audio, imágenes, etc... y obtener conocimiento que sea relevante para el usuario final. Por ejemplo, a partir de un fichero plano, poder descubrir qué entidades son personas, lugares, organizaciones, etc...

Figura 2.37: Uima

2.3 Comparativa de Soluciones Hadoop

Los cincos gigantes del Big Data, Amazon, Apple, Google, Microsoft y Facebook disponen de miles e incluso millones de ordenadores para el tratamiento de sus datos. Estas empresas tienen recursos y herramientas desarrolladas por ellos mismos para la compleja tarea de gestión de sus clústeres.

Comprender la complejidad de la infraestructura de un clúster de Big Data, y la complicación del tratamiento de los datos y su análisis para obtener rendimiento monetario de estos procesos es importante para poder entender mejor los modelos de negocio existentes en el mercado.

Existen dos modelos de negocio. El primer modelo sería las empresas que ofrecen servicios de Hadoop integrales que engloban todos los ámbitos, como son Pivotal, MapR, Microsoft Azure, Amazon AWS, etc. El segundo modelo se especializa en integrar todas las herramientas de Hadoop proporcionando un sandbox que facilite su instalación y manejo, como son, Cloudera Inc., MapR Technologies Inc., IBM InfoSphere Insights, HD Microsoft Insight.

2.3.1 Análisis de los modelos integrales

Si deseamos un servicio integral, desde la obtención de datos, la instalación, programación, análisis de resultados, etc, tenemos muchas opciones. A continuación describiremos las que consideramos más relevantes.

Pivotal **Pivotal Software Inc.**[2]. es una compañía de software y servicios con sede en San Francisco y Palo Alto, California que nació de la escisión de DELL EMC y VMware[3]. Es especialista en varios sectores, entre ellos Cloud Computing y el desarrollo de productos de Big Data. Si deseáramos centrarnos en el problema de Big Data y no tener que hacer inversión en infraestructura ni personal, esta empresa está especializada en realizar el estudio, desarrollo y despliegue Big Data. Entre sus clientes se encuentran empresas de todos los sectores, como telecomunicaciones con TMobile, aeroespacial con Thales, seguros con Liberty Mutual, automoción, deportes, etc.

MAPR **MapR** [4] al igual que Pivotal Software Inc. MapR proporciona un servicio integral con acceso a una variedad de fuentes de datos desde un solo gestor de clústeres. Integra Apache Hadoop y Apache Spark, un sistema de archivos distribuido propio MapRFS, múltiples modelos de sistema de gestión de bases de datos y procesamiento de eventos y flujo de datos. Su tecnología se ejecuta tanto en hardware propio como en servicios públicos de computación en la nube. Lo que la diferencia de todas las demás empresas evaluadas es que en lugar del HDFS, ofrece MapRFS para una gestión de datos más eficiente, más confiable y fácil de usar. Por ello, suelen decir que está más orientada a la "producción empresarial" que el resto. MapR se ha convertido en una de las

[2]https://pivotal.io/
[3]https://www.vmware.com/es.html
[4]https://mapr.com/

empresas con más crecimiento en el sector por encima de Hortonworks [5]. También ofrece una solución sandbox que veremos más adelante.

DELL EMC[6], es una empresa fabricante de software y sistemas para administración y almacenamiento de información, está especializada en almacenamiento y en los últimos años fabrica sistemas específicos para el almacenamiento y el cómputo en Big Data empresarial. Ofrece servicios de consultoría para Big Data y dispone de sistemas específicos para Hadoop como Elastic Cloud Storage. Es una empresa centrada más en las soluciones hardware y software necesarias para el Big Data y no en la tecnología de análisis o inteligencia de negocio.

Amazon AWS [7], se considera el servicio más integrado de todos. Es el pionero en este campo y ofrece una colección de servicios de computación en la nube pública que en conjunto forman una plataforma de computación en la nube. En Amazon AWS podemos obtener desde sólo en alquiler de cómputo por horas a servicios especializados como por ejemplo, Amazon Rekognition, herramientas específicas para el análisis de gran cantidad de imágenes en tiempo real, etc.

IBM[8] probablemente IBM sea la empresa con más patentes tecnológicas en el sector TI (Tecnologías de la Información) y con gran número de laboratorios de investigación. Al igual que el resto de competidores ofrece hardware específico para Big Data así como soluciones tecnológicas, cómo IBM Db2 Big SQL la mítica base de datos de IBM con un motor SQL híbrido para Hadoop, IBM Big Replicate, que proporciona réplica empresarial para Apache Hadoop o IBM Analytics para Apache Spark, su solución de análisis para grandes volúmenes de datos.

Microsoft Azure[9] es el principal competidor de Amazon AWD y Google Cloud Platform. Con su producto HDInsight ofrece ejecutar y gestionar macros de código abierto para Apache Hadoop, Spark y Kafka, etc. Está integrado con Data Factory, para crear procesos ETL (Extract, Transform and Load) y ELT, y Data Lake Storage la solución de almacenamiento escalable de Microsoft, así como con los sistemas de despliegue en la nube de Microsoft Azure, Virtual Network y Azure Active Directory. Ofrece sus servicios de inteligencia artificial y machine learning con Azure AI.

Google Cloud DataProc [10] el sistema de Google para ejecutar en la nube clústeres de Apache Spark y Apache Hadoop. Al igual que Amazon AWS

[5] https://es.hortonworks.com/
[6] https://www.dellemc.com/es-es/big-data/index.htm
[7] https://aws.amazon.com/es/big-data/datalakes-and-analytics/
[8] https://www.ibm.com/es-es/it-infrastructure/solutions/big-data
[9] https://azure.microsoft.com/es-es/
[10] https://cloud.google.com/dataproc/

se paga por el cómputo gastado o por poder disponer de la infraestructura desplegada continuamente con Google Cloud Platform un servicio pensado para grandes corporaciones dado su coste.

Estas soluciones expuestas son las más usadas en el entorno empresarial, ninguna de ellas las hemos podido probar en este proyecto como posibilidad de despliegue de un clúster de Apache Hadoop dado el coste económico que llevan asociado.

2.3.2 Análisis de los modelos sandbox

Una vez estudiadas las grandes soluciones, sólo nos queda analizar las soluciones tipo sandbox que es en las que se desarrollará este proyecto. Es evidente que podemos resolver el problema instalando componente a componente de las distintas tecnologías que se demandan en los objetivos de la Sección 1.3 del proyecto. Pero esto choca con la idea inicial de intentar proporcionar un sistema sencillo y eficiente de despliegue de un clúster Hadoop.

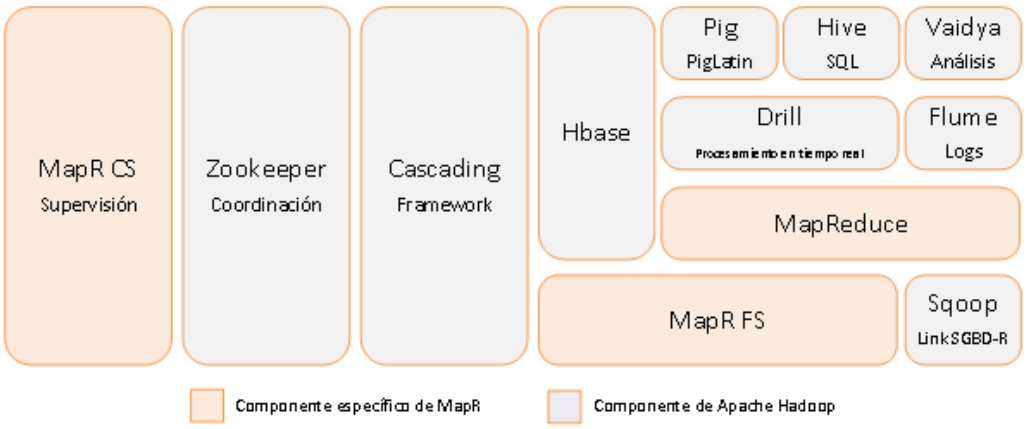

Figura 2.38: Componentes MapR

MapR dispone de tres versiones, una de código abierto que denomina M3, la M5 que añade características de alta disponibilidad a la versión anterior y la M7 que incluye un entorno optimizado para HBase. La estructura de componente de MapR se puede ver en la Figura 2.38. MapR dispone de componentes de Apache Hadoop habituales como ZooKeeper, HBase, Pig, Hive, Flume, Sqoop. Y otros cómo Apache Cascading que permite a los desarrolladores de Java manejar conceptos de Hadoop con un lenguaje de alto nivel sin tener que conocer la API. Hadoop Vaidya es una herramienta de análisis de rendimiento de operaciones MapReduce. Apache Drill es una API para facilitar la creación de consultas basadas en el modelo SQL para MapReduce. Han desarrollado una serie de servicios propios como MapR FS sustituto de HDFS y también un MapReduce

propio. El sistema de gestión del clúster MCS que consiste en una herramienta web que facilita la gestión de servicios y de puestos de trabajo.

HortonWorks es un gran contribuyente al proyecto de Apache Hadoop, su filosofía es simplificar lo más posible el ecosistema Apache Hadoop. HortonWorks ofrece una plataforma híbrida para la gestión global de datos que incluye dos componentes desarrollados por HortonWorks, Hortonworks Data Platform (HDP®) y Hortonworks Data Flow (HDF). Estos sistemas se integran con Hadoop, son rápidos y seguros, proporcionan procesamiento y análisis en tiempo real. La arquitectura del sandbox que proporciona se puede observar en la Figura 2.39, se divide en cuatro grandes bloques, integración y gobierno de datos, acceso a los datos, seguridad y operaciones. En cada bloque integra aplicaciones de Apache Hadoop para dar la mejor solución. Desde la fusión con Cloudera Hortonworks se ha especializado en la optimización de Data Warehouse (EDW), descubrimiento de datos y análisis predictivo. Para ello ha desarrollado varios productos. Hortonworks Data Platform (HDP) ofrece capacidades para desplegar aplicaciones ágiles, sistemas de aprendizaje automático/profundo, almacenamiento de datos en tiempo real, seguridad y gobernanza y mantener los datos seguros en la nube o in situ.

DataPlane es una plataforma de servicios que permite descubrir, gestionar y controlar la distribución de sus datos en entornos híbridos. Proporciona una visión centralizada de los clústeres y administración basada en funciones, así como una plataforma para ejecutar aplicaciones DataPlane en cualquier clúster registrado. Las aplicaciones disponibles incluyen Data Lifecycle Manager (DLM), Data Steward Studio (DSS), Data Analytics Studio (DAS) y Streams Messaging Manager (SMM).

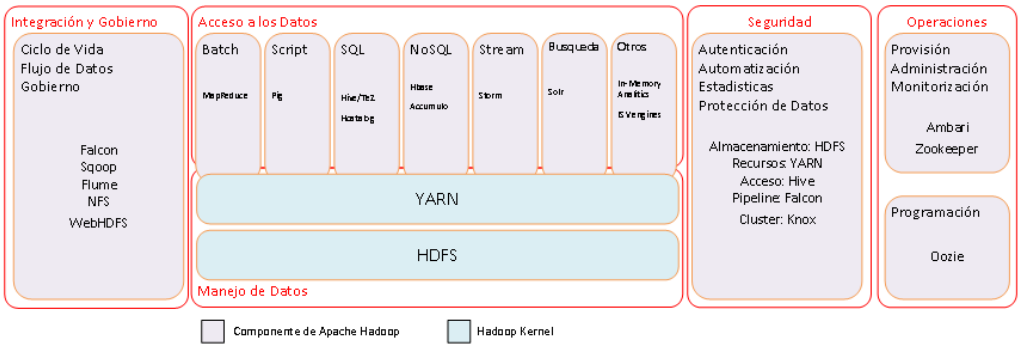

Figura 2.39: Arquitectura HortonWorks

Cloudera proporciona Cloudera CDH (Cloudera Distribution Hadoop), es un sandbox muy completa basada en Apache Hadoop con licencia Open-Source. CDH ofrece los elementos centrales de Hadoop (almacenamiento escalable y computación distribuida) junto con una interfaz de usuario basada en la Web. CDH ofrece procesamiento

por lotes unificado, búsqueda interactiva basada en SQL o controles de acceso basados en roles. La arquitectura de los componentes de CDH de Cloudera se puede ver en la Figura 2.40, en ella se puede observar los componentes que ha desarrollado Cloudera que integra con los de Apache Hadoop, hay que destacar el desarrollo de Hue la interface web para desarrollo, gestión de roles y usuarios.

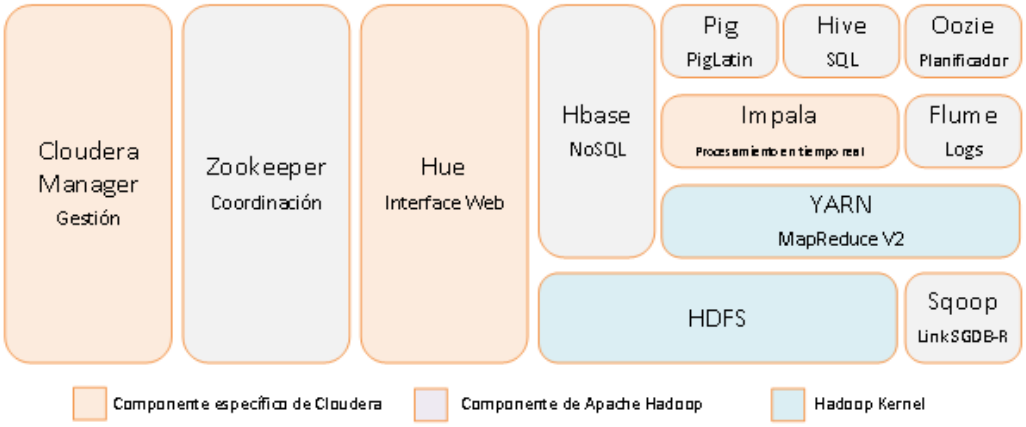

Figura 2.40: Componentes Cloudera

Otra de las soluciones que proporciona Cloudera es Cloudera Manager, que facilita la administración de implementaciones de Cloudera en producción. Permite implementar, configurar y supervisar clústeres de nodos Cloudera CDH a través de una interfaz de usuario intuitiva, completa con actualizaciones continuas, copia de seguridad y recuperación de desastres, y alertas personalizables.

IBM proporciona su propia distribución BigInsights®. Es una plataforma de software flexible que proporciona capacidades para descubrir y analizar perspectivas de negocios que están ocultas en grandes volúmenes de datos estructurados y no estructurados, dando valor a los datos previamente inactivos. En la Figura 2.41 podemos entender la infraestructura que propone IBM, en ella podemos ver las soluciones propias que ha desarrollado, como un sistema de archivos propios GPFS (General Parallel File System).

2.3.3 Comparativa de Distribuciones Hadoop

En la sección anterior hemos realizado una descripción de diversas distribuciones de arquitecturas Hadoop para gestión de grandes volúmenes de datos. Nuestro objetivo principal era hacer una evaluación entre las distribuciones y por lo tanto proporcionar las ventajas y los inconvenientes de los cinco principales proveedores de distribución de Hadoop: Cloudera, Hortonworks, IBM BigInsights®, MAPR y Pivotal HD. De los estudios

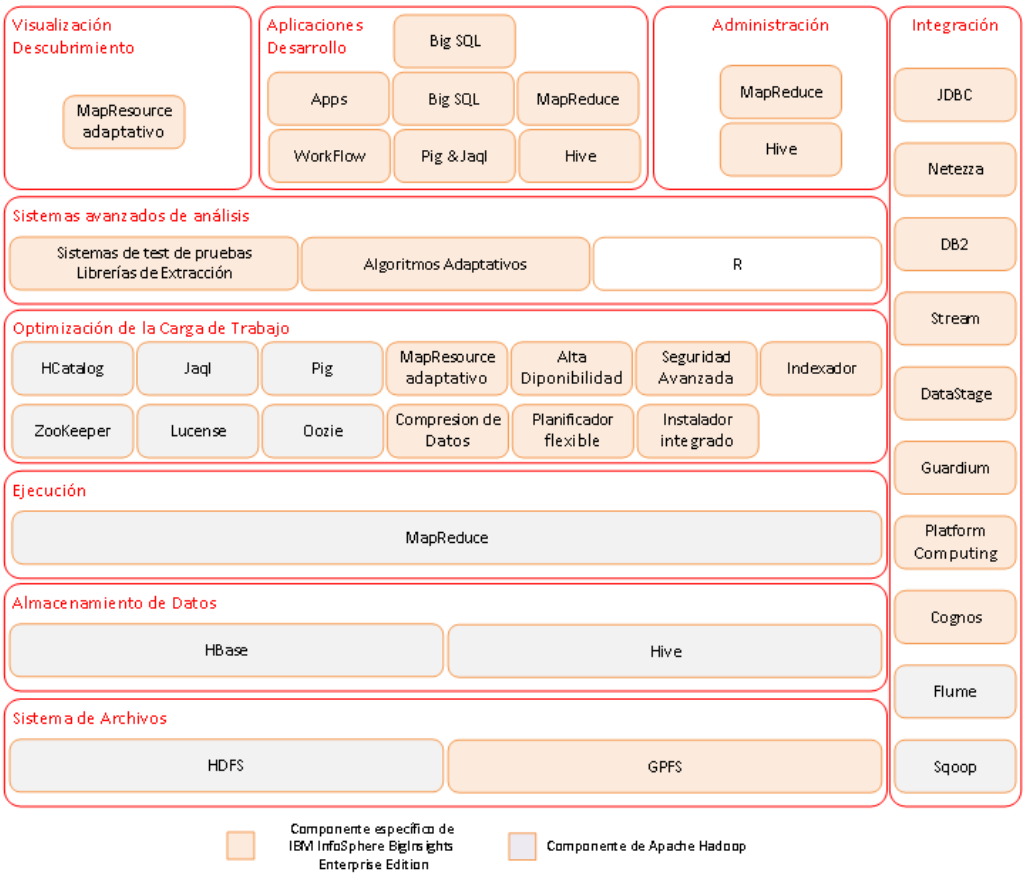

Figura 2.41: Componentes IBM InfoSphere BigInsights Enterprise Edition

comparativos de Allae Erraissi [17] [16], y los realizados por [35], R. D. Schneider [29] y V. Starostenkov [33] de de las distribuciones Hadoop proponen diferentes criterios para comparar las distribuciones. Allae Erraissi [16] propone 34 criterios para tratar de diferencias las distribuciones.

1. **Recuperación de desastres**. Se puede prevenir la pérdida de datos en caso de fallo de algún nodo de cómputo. Tiene la capacidad de reconstruir la infraestructura y reiniciar aplicaciones.

2. **Replicación**. La replicación de datos hace referencia a si los datos se duplican en múltiples lugares de almacenamiento, para mejorar la fiabilidad, la tolerancia y la disponibilidad.

3. **Replicación de metadatos**. Hace referencia a la replicación de los metadatos del clúster.

4. **Herramientas administrativas**. Se refiere a las consolas de gestión utilizados por los diferentes proveedores de soluciones Hadoop para gestionar una distribución del

clúster Hadoop.

5. **Control de los datos y distribución de roles**. Permite controlar la distribución de datos y roles de trabajo en un clúster Hadoop y, por lo tanto, permite elegir los nodos donde ejecutar los trabajos por los diferentes usuarios y grupos.

6. **Alarmas y alertas**. Estas herramientas y sus notificaciones permiten mantener una visión global del sistema de grandes volúmenes de datos. Disponen de representación gráfica de estado de los datos y de los nodos del clúster.

7. **DFS**. Sistema de archivos distribuido para su almacenamiento.

8. **ACL de seguridad**. El sistema asigna ACLs (Access Control List) a los objetos del sistema y controla los permisos de los usuarios a estos objetos.

9. **Ingestión de Datos**. La ingestión de datos es el proceso de importación y la obtención de datos para su uso inmediato o almacenamiento en una base de datos o HDFS.

10. **Tipo de arquitectura de los metadatos**. Se entiende que hay dos tipos de arquitectura de los metadatos del clúster descentralizada y centralizada.

11. **Map Reduce**. Es el modelo de programación dedicado para hacer cálculos paralelos y distribuidos de datos potencialmente muy grandes.

12. **Apache Hadoop YARN**. hace referencia al tipo de tecnología para el procesamiento de las operaciones que no sean MapReduce.

13. **Bases de datos no relacional**. Dispone de tecnologías NoSQL.

14. **Servicios de meta datos**. Es una tecnología de referencia orientado a objetos.

15. **Sistemas de Scripting**. Proporcionan lenguajes de programación que utilizan construcciones de alto nivel para interpretar y ejecutar un comando a la vez.

16. **Acceso a datos y consultas**. Los usuarios utilizan consultas para expresar sus necesidades de información y de acceso a datos.

17. **Programador de flujos de trabajo**. Es una representación de una secuencia de operaciones o tareas llevadas a cabo por una persona, un grupo de personas o una organización.

18. **Coordinación del clúster**. Tiene como objetivo garantizar un enfoque coherente para trabajar con el clúster.

19. **Transferencia de datos entre RDB (Relational Data Base) y Hadoop**. Dispone de herramientas diseñadas para transferir eficientemente datos entre Apache Hadoop y almacenes de datos estructurados como bases de datos relacionales.

20. **Servicios de gestión de logs distribuidos**. Estos servicios están pensados para manejar grandes volúmenes de mensajes de logs de manera distribuida.

21. **Machine learnning**. Se refiere a las herramientas de análisis, diseño, desarrollo y aplicación de métodos de aprendizaje automático.

22. **Análisis de datos**. Permite procesar una gran cantidad de datos e identificar los aspectos más interesantes de la estructura de estos datos. Además, eso finalmente proporciona gráfica representaciones, que pueden revelar las relaciones que son difíciles de captar por el análisis de datos directa.

23. **Servicios en la nube**. Dispone de servicios en la nube, aprovechan la potencia de cálculo y almacenamiento de los servidores de equipos remoto a través de Internet.

24. **Motor de computación paralela de consultas**. Dispone de motores para optimizar la ejecución de consultas e índices.

25. **Búsqueda de texto completo**. Dispone de técnicas de búsqueda de palabras en documentos o base de datos.

26. **Almacenamiento de datos, Data Warehouse**. Significa una base de datos utilizada para recoger, ordenar, registrar y almacenar información de bases de datos operacionales. También proporciona una base para apoyar las decisiones de negocio.

27. **Extracción, Transformación y Carga (ETL)**. Permite la sincronización masiva de información de una fuente de datos a otro.

28. **Análisis de Datos e interacción**. Permite analizar los datos e interactuar con los resultados.

29. **Autenticación**. Es la capacidad para validar la legitimidad del acceso al sistema.

30. **Autorización**. Permite determinar si el sujeto autenticado puede ejecutar la acción deseada en el objeto especificado.

31. **Contabilidad y trazabilidad**. Es la obligación de informar y explicar, de forma transparente, la trazabilidad de las operaciones.

32. **Protección de Datos**. En qué medida se adoptan normas internas y para aplicar las medidas adecuadas para garantizar y demostrar el tratamiento de datos personales, que se lleva a cabo en cumplimiento de la normativa y las libertades de TI.

33. **Monitorización, provisión y manejo**. Hace referencia a las herramientas para la gestión de configuración, gestión y seguimiento del sistema informático. Aprovisionamiento de nodos en el clúster, configuración del software, asignar espacio en disco, la memoria, etc. Debe disponer de un sistema para poder monitorizar permanentemente el sistema informático y ser alertado en caso de detectar de operaciones anormales.

34. **Programación**. Hace referencia a la capacidad de planificar tareas en de procesamiento en una o más máquinas de computación.

Como se puede observar en la Tabla 2.1 todas las distribuciones cumplen con casi todos los criterios establecidos. No hay en el mercado una distribución claramente dominante respecto a las otras, tiene lógica ya que todas se basan el Apache Hadoop. Algunas desarrollan o mejoran alguna característica, pero los servicios que proporcionan son en casi todas los mismos. Algunas de ellas aportan recursos a Apache Hadoop para que continúe evolucionando el producto. Por desgracia no disponemos de recursos suficientes para comparar instalaciones equivalentes y comparar su rendimiento en especial en aquellas que proporcionan sistemas de almacenamiento propio como puede ser GPFS o MapR FS.

Selección de Cloudera

El 3 de octubre de 2018 [4] Cloudera y Hortonworks anuncian su fusión. Esto nos llevó a preguntarnos qué distribución sería la que quedaría como vigente en el mercado y qué roles tomaría cada sección de la empresa. Tom Reilly, CEO de Cloudera, comentó en el anuncio de la fusión "*Hoy iniciamos un nuevo y excitante capítulo para Cloudera al convertirnos en la compañía líder proveedora de data cloud empresarial. Esta combinación de equipo y tecnologías sitúa a la nueva Cloudera como un claro líder del mercado, con las dimensiones y los recursos necesarios para impulsar la innovación y el crecimiento continuos. Facilitaremos a los clientes un paquete integral de soluciones con el que podrán llevar la analítica de sus datos a cualquier área en la que necesiten trabajar, desde el Edge hasta la Inteligencia Artificial, con el primer Enterprise Data Cloud de la industria*". En su presentación parece que HortonWorks se centraría en plataformas híbridas y que

Característica	Cloudera	Hortonworks	MapR	IBM BigInsights	Pivotal HD
Recuperación de desastres	✓	✷	✓	✓	✓
Replicación	✓	✓	✓	✓	✓
Replicación de meta datos	✷	✷	✓	✓	✓
Herramientas administrativas	✓	✓	✓	✓	✓
Control de los datos y distribución de roles	✷	✷	✓	✓	✷
Alarmas, alertas	✓	✓	✓	✓	✓
DFS	✓	✓	✓	✓	✓
ACL de seguridad	✓	✓	✓	✓	✓
Ingestión de Datos tipo Batch	✓	✓	✓	✓	✓
Ingestión de Datos tipo Stream	✓	✷	✓	✓	✓
Tipo de arquitectura de los metadatos Centralizada	✓	✓	✷	✷	✷
Tipo de arquitectura de los metadatos Distribuida	✷	✷	✓	✓	✓
Map Reduce	✓	✓	✓	✓	✓
Apache Hadoop YARN	✓	✓	✓	✓	✓
Bases de datos no relacional	✓	✓	✓	✓	✓
Servicios de meta datos	✓	✓	✓	✓	✓
Sistemas de Scripting	✓	✓	✓	✓	✓
Acceso a datos y consultas	✓	✓	✓	✓	✓
Programador de flujos de trabajo	✓	✓	✓	✓	✓
Coordinación del clúster	✓	✓	✓	✓	✓
Transferencia de datos entre RDB y Hadoop	✓	✓	✓	✓	✓
Servicios de gestión de logs distribuidos	✓	✓	✓	✓	✓
Machine learnning	✓	✓	✓	✓	✓
Análisis de datos	✓	✓	✓	✓	✓
Servicios en la nube	✓	✓	✓	✓	✓
Motor de computación paralela de consultas	✓	✓	✓	✓	✓
Búsqueda de texto completo	✓	✓	✓	✓	✓
Almacenamiento de datos, Data Warehouse	✓	✓	✓	✓	✓
Extracción, Transformación y Carga (ETL)	✓	✓	✓	✓	✓
Análisis de Datos e interacción	✓	✓	✓	✓	✓
Autenticación	✓	✓	✓	✓	✓
Autorización	✓	✓	✓	✓	✓
Contabilidad y trazabilidad	✓	✓	✓	✓	✓
Protección de Datos	✓	✓	✓	✓	✓
Monitorización, provisión y manejo	✓	✓	✓	✓	✓
Programación	✓	✓	✓	✓	✓

✓= Cumple la Característica
✷= No Cumple la Característica

Tabla 2.1: Comparativa de distribuciones Hadoop para Big Data

Cloudera mantendría el sandbox. Hortonworks ofrece soluciones integrales en plataformas de datos híbridas. De esta forma entra a competir con MapR o Pivotal.

Debido a esta fusión empresarial, nos redujo la selección de distribuciones de cinco a cuatro. Pivotal e IBM las descartamos ya que no las pudimos probar debido al coste económico que suponen. Entre las dos que nos quedaban Cloudera y MapR la propia licencia de distribución nos llevó a descartar MapR. Las características que proporciona Cloudera Express, la licencia gratuita para prueba y aprendizaje dispone de más características que la que proporciona MapR. Por otra parte, la documentación abierta que dispone Cloudera es significativamente mejor que la de MapR. Por lo tanto, nos declinamos por la instalación Cloudeda Manager Express.

Licencias Cloudera

Ahora que sabemos que distribución vamos a instalar es interesante explicar que tipo de licencias ofrece para conocer de qué características dispondremos. Al instalar Cloudera Manager, se puede seleccionar entre las siguientes ediciones:

- **Cloudera Express**. No necesita licencia, incluye la licencia de CDH y las funciones de Core Cloudera Manager, que consisten en las funcionalidades básicas de Cloudera Manager. Admite hasta 100 nodos de CDH en total, ya sea en un clúster o en varios.
- **Licencia de prueba de 60 días de Cloudera Enterprise**. A las funciones de Cloudera Express se añaden las funciones avanzadas de Cloudera Manager y Cloudera Navigator, tan solo durante 60 días. Las funciones avanzadas de Cloudera Manager son:

  - Autenticación LDAP y SAML.
  - Historial de configuración.
  - Alertas entregadas como capturas SNMP y scripts de alerta personalizados.
  - Copia de seguridad y recuperación de desastres.
  - Informes operacionales.
  - Cloudera Navigator.
  - Comandos como reinicio rotativo, Historial y Rollback, y sistema de envío de diagnósticos.
  - Informes de uso de clúster.
  - Control de acceso basado en roles y diseño de roles específicos para el clúster.

- **Cloudera Enterprise**, Es una subscripción anual. A las funcionalidades anteriores se añade, Key Trustee de Cloudera Manager, que es un sistema de autenticación y cifrado, y el servicio de soporte. Cloudera Enterprise está disponible por suscripción en cinco versiones, cada una pensada a en un tipo de uso de la plataforma Cloudera Manager:

- **Essentials Edition**. Proporciona soporte superior y administración avanzada para el núcleo de Apache Hadoop. El precio por licencia está en 4.000$ por licencia y admite pago por uso en la nube, ya sea en la nube propia de Cloudera o en multi-cloud.
- **Data Science and Engineering Edition**. Para preparación de datos programática y creación de modelado. El precio por licencia está en 4.000$ por licencia y admite pago por uso en la nube propia de Cloudera.
- **Operational Database Edition**. Para aplicaciones en línea con necesidades de servicio en tiempo real. El precio por licencia está en 6.000$ por licencia y admite pago por uso en la nube, propia de Cloudera.
- **Data Warehouse Edition** para análisis BI (Business Intelligence) y análisis con SQL. El precio por licencia está en 8.000$ por licencia y admite pago por uso en la nube, ya sea en la nube propia de Cloudera o en multi-cloud.
- **Enterprise Data Hub Edition**. Proporciona un uso completo de la plataforma. El precio por licencia está en 10.000$ por licencia y no admite pago por uso en la nube.

2.4 Cloud Computing y Virtualización

Actualmente, la mayoría de las organizaciones no disponen de los medios humanos con el conocimiento necesario para discernir sobre las diferentes opciones en el mercado en todos los ámbitos de las TI (Tecnologías de la Información), e incorporar perfiles en la organización es una solución costosa cuando normalmente las estrategias se marcan a corto plazo, 3 o 4 años vista. Para dar solución a este problema aparecen el Cloud Computing y los sistemas de Virtualización, que ponen a disposición de las organizaciones los medios necesarios para que, sin afrontar grandes costos de propiedad, dispongan de las actividades propias de estrategia en soluciones TI.

Cloud Computing

Cuando se habla de Cloud Computing, todavía muchos interpretan que se trata de complejas soluciones únicamente para grandes empresas con un enorme volumen de datos y una considerable capacidad financiera. Pero nada más lejos de la realidad, la nube es especialmente útil para pequeñas o grandes instalaciones. Estas son algunas de las grandes ventajas de trabajar en este entorno ya sea con un Cloud propio realizando una inversión en componentes y personal TIC, o comprando cómputo en la nube de algún proveedor:

1. **Reducción de costes**. Adquirir cómputo en empresas cómo Google, AWS, Azure, etc, hace que la empresa prescinda de realizar inversiones en infraestructura TI

propia y licencias de software. Adquirir una infraestructura de virtualización hace que la inversión en hardware se amortice en el tiempo mucho más rápido y que la inversión en personal TI sea inferior, con menos personal se puede mantener mucha más infraestructura con menos personal. Adquirir una infraestructura tecnológica propia, en muchas ocasiones, es mucho más costoso y no todos los negocios pueden permitírselo. ¿Por qué? porque no se trata solamente de la adquisición tecnológica, sino de todo lo que esto conlleva: gastos de mantenimiento, energéticos, personal técnico etc. Está demostrado que con la gestión desde la nube un negocio puede ahorrar entre un 20 % y 30 %. Por otro lado, la empresa se puede centrar exclusivamente en su actividad sin necesidad de tener que dedicar tiempo y recursos a mantenimiento tecnológico, que a veces puede salir tremendamente caro.

2. **Movilidad**. Acceso desde cualquier dispositivo y lugar. Estés donde estés tienes acceso a toda la información de la empresa. La movilidad se ha convertido en una gran ventaja competitiva, tanto para trabajar o atender clientes desde cualquier lugar, como para favorecer la flexibilidad laboral de un trabajador que puede organizar toda su actividad sin depender de una oficina. Simplemente basta con tener conexión a Internet para acceder a las aplicaciones o a la información. Además, los documentos están alojados en la nube, no en equipos individuales, por lo tanto, distintos usuarios pueden compartirlos o trabajar sobre ellos sin necesidad de estar físicamente juntos.

3. **Tecnología siempre actualizada**. Poder disfrutar siempre de las últimas versiones del software y las más modernas aplicaciones hasta hace poco tiempo era un "lujo" solo al alcance de las grandes compañías. Con la nube, el cliente se asegura una tecnología siempre actualizada y optimizada. Las actualizaciones desde el punto de vista del usuario, se hacen automáticamente y simplemente estarán disponibles la próxima vez que inicie su sesión en la nube.

4. **Capacidad de almacenamiento ilimitada**. Hoy en día manejamos grandes cantidades de datos y la nube ofrece un almacenamiento prácticamente ilimitado ya que se optimiza al ser común a la empresa, no se disponen de cientos o miles de discos personales a medio o bajo uso.

5. **No es necesario "lo último" en ordenadores**. Se pueden utilizar ordenadores con discos duros más pequeños, menos memoria y procesadores. Además, con menos programas acaparando la memoria el ordenador, el PC rendirá mucho mejor porque tiene menos procesos cargados.

6. **Seguridad**. A diferencia de la informática tradicional, en la que cualquier fallo en el PC puede destruir todos los datos, en este entorno, si el disco del ordenador deja de funcionar eso no afectaría a tus datos. Si se da el caso en el que el ordenador se bloquea, todos los datos seguirán en la nube y totalmente accesibles desde otro dispositivo. En cuanto a las copias de seguridad de la información, está claro que son vitales para cualquier empresa. En el entorno Cloud Computing el usuario se despreocupa de las copias de seguridad porque la nube las realiza automáticamente y con posibilidad de un cifrado seguro, a prueba de cualquier ataque de hackers.

7. **Dejar de estar atado a un PC**. Se puede cambiar sin miedo los ordenadores, porque las aplicaciones y los datos seguirán estando en la nube. Los documentos, datos y aplicaciones son los mismos sin importar qué ordenador o dispositivo se esté utilizando para acceder a ellos.

8. **Igualdad**. Por último, y sin duda una de las ventajas más importantes, es que el servicio en la nube permite que todos los usuarios estén en las mismas condiciones en el acceso a la tecnología. Pudiendo ajustarla a las necesidades de cada usuario.

Con el sistema que disponemos, Proxmox, podemos explotar todas estas ventajas del Cloud Computing, aprender a gestionar una pequeña infraestructura de virtualización y formarnos en la gestión de almacenamiento, cómputo y gestión de red.

Virtualización

La virtualización se está convirtiendo en un recurso muy popular entre las empresas, ya que esta permite la ejecución simultánea de múltiples sistemas operativos y aplicaciones en un único dispositivo físico o hardware. Para ello, es necesario un programa o software que nos permita crear lo que se denominan máquinas virtuales, las cuales pueden ser servicios, redes o servidores que se montan en un PC o servidor físico.

En los sistemas de virtualización una máquina virtual es un programa que crea un equipo virtual en su sistema, haciendo creer que es un equipo de verdad. Existen muchas distribuciones de software para virtualizar y gestionar sistemas virtualizados. Algunos de los más notables son:

- **VMware vSphere Enterprise**. VMWare se trata de una de las principales compañías en el ámbito de la virtualización. Sus sistemas de virtualización sirven tanto para ordenadores de escritorio como para sistemas de servidores y es el software de virtualización más utilizado por las empresas y de los más avanzado.
- **Microsoft Hyper-V Server**. Hyper-V permite crear y administrar un entorno informático virtualizado mediante la tecnología de virtualización integrada en Windows

Server. Es un sistema pensado para entornos Microsoft, aunque en admite la virtualización de algunas distribuciones de Linux y FreeBSD.

- **Citrix XenServer**. Citrix, junto con VMWare, es una de las grandes compañías de virtualización y de las pioneras. Citrix está basado en software de código abierto y dispone de dos versiones: una de pago y otra libre. Citrix XenServer es una plataforma de virtualización de servidores administrada, completa e integrada en el potente hipervisor Xen. La tecnología Xen proporciona aislamiento seguro, control de recursos, garantías de calidad de servicio y migración de máquinas virtuales en caliente. XenServer está diseñado para una gestión eficiente de los servidores virtuales de Windows y Linux.
- **VirtualBox**. VirtualBox se caracteriza por ser muy fácil de usar, disponible para Windows, Linux, Solaris y OS. Es un sistema de virtualización para uso personal, no para crear un clúster de virtualización, pero es de los más usados para uso personal.
- **KVM**. Kernel-based Virtual Machine (KVM) es un software de virtualización libre y de código abierto para Linux que se basa en las extensiones de virtualización de hardware Intel VT-X y AMD-V y una versión modificada QEMU. QEMU es un emulador de procesadores basado en la traducción dinámica de binarios. KVM permite ejecutar máquinas virtuales utilizando imágenes de disco que contienen sistemas operativos sin modificar. Cada máquina virtual tiene su propio hardware virtualizado: una tarjeta de red, discos duros, tarjeta gráfica, etc. Muchos sistemas se han desarrollado alrededor de este proyecto de código abierto, como pueden ser Opennebula, Openstack o Proxmox que es el sistema que disponemos.

Licencia Proxmox

Al igual que vimos con Cloudera, es necesario conocer la licencia del software que usaremos en el sistema de virtualización. El código fuente de Proxmox Virtual Environment se publica bajo la licencia de software libre GNU AGPL, v3 y, por lo tanto, está disponible gratuitamente para descargar, usar y compartir. Las subscripciones de licencias de Proxmox VE están enfocadas a dar un servicio adicional para ayudar a los profesionales de TI y las empresas a mantener sus implementaciones de Proxmox VE actualizadas. Proporciona acceso al repositorio empresarial estable de Proxmox VE, que proporciona actualizaciones de software y mejoras de seguridad confiables, y ayuda técnica y de soporte.

Se ofrecen cuatro planes de suscripción basado en la cantidad de servidores físicos y sockets de CPU. El período de suscripción es de un año desde el momento de la compra y según la licencia subscrita obtendremos un tipo de soporte diferente. Los tipos de suscripción disponibles son:

- **COMUNIDAD**. Este tipo de licencia ofrece acceso al repositorio empresarial, conjunto completo de características, y soporte comunitario, por un precio de 79,9 € por año y CPU socket.
- **BASIC**. A la licencia anterior se le añade, soporte a través del portal del cliente, 3 tickets de soporte / año y tiempo de respuesta de 1 dia hábil. El coste de esta licencia es de 259.90 € por año y CPU socket.
- **ESTANDAR**. A la licencia anterior se cambia a 10 tickets de soporte / año, tiempo de respuesta de 4h dentro de un día hábil y soporte en línea por SSH. El coste es de 398 € por año y CPU socket.
- **PRIMA**. A la licencia anterior se disponen de tickets ilimitados, tiempo de respuesta de 2h dentro de un día hábil y soporte en línea por SSH. El coste es de 796 € por año y CPU socket.

La instalación que disponemos no tiene ningún tipo de suscripción, trabajaremos sin licencia y por lo tanto sin soporte.

2.5 Automatización de arquitectura TI

Automatizar los procesos de implementación y administración de la infraestructura TI, es fundamental para obtener un rendimiento apropiado en las instalaciones TI. Cuando se trabaja con una nube como puede ser, Azure o AWS. En estos clouds la automatización de la infraestructura viene dada con la interface del sistema y proporcionan una API para los clientes, pero no se conoce cómo está desarrollada o como se lleva a cabo. Dentro del marco de este proyecto se buscará una herramienta para automatizar el despliegue en el sistema de virtualización. Los objetivos que pretenden cumplir este tipo de sistemas son:

- Mejorar la frecuencia de despliegue, poder repetir las tareas.
- Reducir la tasa de errores en las tareas habituales.
- Disminuir los tiempos de entrega de la infraestructura.
- Disminuir los tiempos de recuperación en caso de fallo.

II Segunda Parte: Tecnología

3. Configuración del Clúster

Vamos a explicar las características concretas del despliegue del clúster Cloudera en un entorno de vitualización Proxmox, esto es, la metodología a la que hemos llegado para minimizar los mantenimientos y los tiempos de despliegue de un clúster Cloudera. Es importante saber de los recursos que se disponen para poder dimensionar adecuadamente el clúster que si quiere desplegar. Si la situación fuera el despliegue en una nube como Amazon, el sistema hardware es más transparente ya que de eso se encarga el sistema de Amazon, nosotros sólo nos preocuparíamos del coste de los recursos que desplegamos. Por lo cual, lo primero que describiremos será el hardware disponible, para luego explicar brevemente el sistema de virtualización de Proxmox y sus capacidades.

Para el sistema de despliegue usaremos una tecnología muy usada en administración de sistemas, Ansible. Una vez descritas estas tres características nos centraremos en la instalación de Cloudera Manager y los nodos explicando cada paso y las razones por las cuales se tomaron.

3.1 Hardware de despliegue

El hardware que se dispone es heterogéneo dado que se utiliza para dar un servicio multidisciplinar. Las tecnologías hardware que se disponen son de diversas generaciones, debido a que no se ha adquirido en una única compra.

3.1.1 Placas de computación

El hardware de computación consta de siete placas con características diversas. Cuatro de ellas forman parte de un chasis de blades. Un chasis de blades es un equipo integral que consta de hasta seis hojas de cálculo sin disco, una red de almacenamiento (SAN) integrada y de tres a cinco módulos de servicio. El resto son equipos de propósito general. Los equipos disponibles se detallan a continuación:

1. **Intel Modular Server**, es un sistema blade fabricado por Intel que utiliza sus propias placas base y procesadores (ver Figura 3.1). El sistema consta de un chasis de servidor modular Intel, hasta seis blade de cálculo sin disco, una SAN integrada y de tres a cinco módulos de servicio. Este servidor está dirigido a pequeñas y medianas empresas, apareció en el mercado en el 2008.

Figura 3.1: Intel Modular Server

2. Equipo **Dell PowerEdge R715**, cuenta con tecnología de procesador AMD Opteron™ (ver Figura 3.2) diseñada para manejar sus cargas de trabajo exigentes. Este sistema dispone almacenamiento local.

Figura 3.2: Dell PowerEdge R715, Figura

3. **Intel ECS IXION** Equipo basado en tecnología Intel con almacenamiento local (ver Figura 3.3).

4. **SuperMicro CSE-828TQ** Equipo para cálculo intensivo, no dispone de almacenamiento propio, ver Figura 3.4.

En la Tabla 3.1 podemos ver las características técnicas de cada sistema de computación. Observamos los distintos saltos generacionales en los procesadores y las memorias, la

Figura 3.3: Intel ECS IXION

Figura 3.4: SuperMicro CSE

tabla está ordenada de menor a mayor capacidad de procesamiento. Estas características nos ayudarán a alojar los nodos más grandes en los equipos más potentes.

La estructura del almacenamiento está estratificada en tres tipos de almacenamiento:

1. El propio de cada nodo. Es almacenamiento local de tecnología SAS formando un RAID 6. Dentro de este tipo de almacenamiento tenemos también el que proporciona Intel Modular server que dispone de almacenamiento local de tipo NAS compartido entre todos los nodos, lo cual permite tener volúmenes locales a cada nodo y otros compartidos entre todos los nodos, algo que es ideal para un sistema de virtualización y no tanto para sistemas Hadoop.

2. Almacenamiento tipo NAS de 3Gb/s, está basado en tres cabinas SAS de doble controladora que permite conectarlas a los sistemas de cómputo por tarjetas SAS externas. Con unos Switch SAS podemos compartir este almacenamiento entre todos los nodos o darle a un nodo un almacenamiento dedicado.

3. Almacenamiento ISCSI a 10Gb/s es un almacenamiento compartido por todos los equipos basado en tecnología de HUAWEI. Disponemos de un equipo SuperMicro CSE-828TQ, ver en sección, que no dispone de almacenamiento local, este nodo usará este tipo almacenamiento, el resto de nodos dispone de acceso a este almacenamiento con anchos de banda de 10Gb/s o 1Gb/s.

La Figura 3.5 muestra la distribución física del almacenamiento, con los tres tipos de tecnologías de conexión. Algunos de estos sistemas de almacenamiento disponen de caches, por niveles, ya sea cache RAM, cache SSD o las dos, como por ejemplo la cabina

Servidor	Procesador	Nº de Procesadores	Frecuencia Núcleo	Núcleos Totales	Hilos Totales	Memoria	Tipo Memoria
IMS Blade 1	Xeon E5630	2	2.53GHz	8	16	48GB	DDR3 800Mhz
IMS Blade 2	Xeon E5630	2	2.53GHz	8	16	64GB	DDR3 1067Mhz
IMS Blade 3	Xeon X5675	2	3.07GHz	12	24	144GB	DDR3 1333Mhz
IMS Blade 4	Xeon E5-2620	2	2.30GHz	12	24	96GB	DDR3 1333Mhz
Dell PowerEdge R715	Xeon Opteron 6376	2	2.00GHz	32	32	256GB	DDR3 1600Mhz
Intel ECS IXION	Xeon Xeon E5-2630	2	2.20GHz	20	40	256GB	DDR4 2400Mhz
SuperMicro CSE	Xeon Gold 6128 E5630	2	3.40GHz	12	24	256GB	DDR4 2666Mhz

Tabla 3.1: Recursos de Cómputo Disponibles

de almacenamiento Huawei y el equipo Intel ECS IXION. Este tipo de almacenamientos mejoran mucho el rendimiento de lectura escritura. De esta forma, podemos distribuir las máquinas del clúster en los distintos tipos de almacenamiento para que se adapte a las necesidades que pretendemos dar al a esa máquina virtual. En el caso de un clúster HADOOP procuramos que las máquinas virtuales del clúster usen el almacenamiento local del nodo de virtualización ya que estas pueden causar sobrecargas debido a la cantidad enorme de operaciones de entrada salida a disco que realizan.

El sistema de virtualización soporta servicios diversos como son:

1. **Servicios principales del departamento**: web, correo, servidores de dominio, servidores de archivos, servidores de aplicaciones.

2. **Equipos de uso personal**:escritorios windows, linux para uso personal.

3. **Servicios de investigación**: equipos dedicados a cálculo intensivo con Mathlab, , servidores web, servidores de archivos, servidores de dominio, clústers de Hadoop, etc.

3.2 Proxmox

Proxmox VE “Virtual Environment” [25] (ver Figura 3.6) es una potente plataforma de virtualización de nivel empresarial 100 % libre y sin límites en su uso. El modelo de negocio de PROXMOX se basa en capacitación, certificaciones y soporte. Proxmox VE es un entorno de virtualización basado en Debian con KVM y OpenVZ que nos permiten disponer de virtualización completa, paravirtualización y contenedores.

Figura 3.6: Proxmox VE

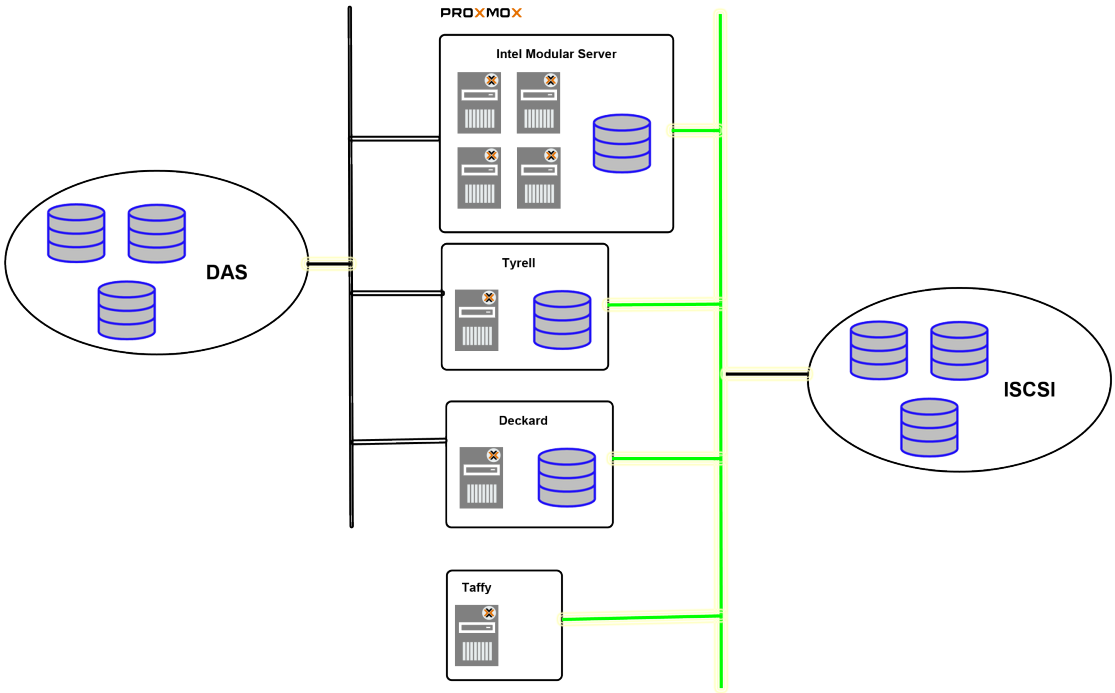

Figura 3.5: Estructura del sistema de almacenamiento

Principales características de PROXMOX

1. **Administrador Web HTML5**. PROXMOX proporciona un interfaz Web para configurar los servidores físicos, clúster, máquinas virtuales, políticas de backups, restauración de backups, snapshots. No es necesario instalar aplicaciones clientes en su máquina para administrar y siendo HTML5 le permite conectarse y gestionar el entorno virtualizado desde cualquier tipo de plataforma.

2. **Virtualización para la mayoría de Sistemas Operativos**, en sus versiones 32/64bits: Linux en todas sus versiones, Microsoft Windows 10 / 2016 / 2012 / 7 / 8/ 2003 / XP, Solaris, AIX, entre otros.

3. **KVM**, es una solución para implementar virtualización sobre Linux. Puede funcionar en hardware x86/x86_64 y es necesario que el microprocesador tenga soporte de virtualización Intel VT (Virtualization Technology) y en AMD SVM (Secure Virtual Machine).

4. **Container-based Virtualization** (LXC), es una alternativa para ejecutar máquina "Linux" en espacios separados. A diferencia de la virtualización este funciona como un módulo agregado al servidor físico y hace uso directo del hardware (también conocido como paravirtualización).

5. **Backup & Restore de máquinas virtuales**. Realizar estas tareas en Proxmox es sencillo y se administra a través de su interfaz Web. Puede efectuar una copia de seguridad de forma inmediata o dejarlo programado. La restauración es simple, y sólo debe seleccionar la copia de seguridad a restaurar.

6. **Snapshot Live**. Le permite hacer copias instantáneas de máquinas virtuales incluyendo el contenido de la RAM, su configuración y el estado de los discos virtuales. Permite retroceder en tiempo la máquina virtual restaurando snapshots.

7. **Migración en caliente**. La administración de los nodos es centralizada a través de un interfaz Web, permitiéndole movilizar máquinas virtuales entre cada servidor físico (nodo) sin tener que apagar la máquina virtual.

8. **Clúster Alta Disponibilidad**. Esta característica le permite definir reglas de alta disponibilidad en el clúster, por ejemplo: Si uno de los servidores físicos (nodo) esta sobrecargado, este transfiere automáticamente a otro servidor físico con menos carga la máquina virtual.

9. **Administración centralizada**. En un clúster Proxmox se debe definir uno de los nodos como orquestador con el objetivo de centralizar el trabajo, sin embargo, cada nodo cuenta con su propio administrador Web.

10. **Clúster sin SPOF** (Single Point Of Failure). Cada servidor físico Proxmox cuenta con su propio interfaz Web permitiendo acceso a la administración de las máquinas virtuales. Si el nodo orquestador llega a fallar, cada nodo tiene replicado la información del orquestador y desde cualquiera de los nodos puede tomar control del clúster.

11. **Puentes de red**. Proxmox administra las tarjetas físicas a través de Bridges de red virtuales que comparte a las máquinas virtuales.

12. **NAS & SAN**. Se pueden disponer de ilimitadas NAS o SAN ya sea a través de canal de fibra (fiber channel), iSCSI over ethernet, NFS, ZFS, ZFS over ethernet, samba CIFS.

13. **Autenticación**. Puede configurar la autenticación de acceso al área de administración a los nodos a través de cuentas propias con Proxmox o utilizando LDAP/Active Directory.

14. **Firewall**. Proxmox VE Firewall proporciona una manera fácil de proteger su infraestructura en un entorno virtualizado. Puede definir reglas de firewall para todas las máquinas virtuales o definir reglas precisas a una máquina virtual, e incluso definir reglas para cada nodo.

3.3 Ansible

La necesidad que existe hoy en día de escalar los servicios en el menor tiempo posible, realizar tareas de mantenimiento en un gran número de servidores físicos o virtuales ha llevado a desarrollar nuevas formas de desplegar, configurar y actualizar las máquinas, de forma que las tareas mecánicas que llegan a ser idénticas para muchos grupos de equipos no hagan perder tiempo y se realicen el en menor tiempo posible.

Figura 3.7: Ansible

```
TASK [Borramos particion y creamos otra con el tamaño corregido] ****************************
ok: [nodo-05]
ok: [nodo-02]
ok: [nodo-01]
ok: [nodo-04]
ok: [nodo-03]
ok: [nodo-06]
ok: [nodo-07]

TASK [Releemos la partcion] *****************************************************************
changed: [nodo-02]
changed: [nodo-01]
changed: [nodo-05]
changed: [nodo-04]
changed: [nodo-03]
changed: [nodo-06]
changed: [nodo-07]

TASK [Aumentamos el espacio del sistema de ficheros] ****************************************
changed: [nodo-05]
changed: [nodo-02]
changed: [nodo-04]
changed: [nodo-03]
changed: [nodo-01]
changed: [nodo-06]
changed: [nodo-07]

PLAY RECAP **********************************************************************************
c1blade3                   : ok=2    changed=1    unreachable=0    failed=0
c1blade4                   : ok=2    changed=1    unreachable=0    failed=0
deckard                    : ok=2    changed=1    unreachable=0    failed=0
nodo-01                    : ok=5    changed=2    unreachable=0    failed=0
nodo-02                    : ok=5    changed=2    unreachable=0    failed=0
nodo-03                    : ok=5    changed=2    unreachable=0    failed=0
nodo-04                    : ok=5    changed=2    unreachable=0    failed=0
nodo-05                    : ok=5    changed=2    unreachable=0    failed=0
nodo-06                    : ok=5    changed=2    unreachable=0    failed=0
nodo-07                    : ok=5    changed=2    unreachable=0    failed=0
taffy                      : ok=2    changed=1    unreachable=0    failed=0
tyrell                     : ok=3    changed=2    unreachable=0    failed=0

[root@orchestrator ~]#
```

Figura 3.8: Salida Playbook de Ansible

Para dar respuesta a esta demanda han surgido diferentes herramientas que permiten automatizar muchas tareas que son repetitivas en los equipos. Algunas de las herramientas son Puppet, Chef, Salt y de la que hemos seleccionado para la gestión del despliegue del clúster, Ansible.

Ansible, Figura 3.7, es una herramienta Open Source desarrollada en Python y comercialmente ofrecida por AnsibleWorks que la definen como un motor de orquestación muy simple que automatiza las tareas necesarias en el campo de las TI [27]. La principal característica de Ansible y por la que tiene un gran éxito es su sencillez, es simple de utilizar y junto su amplia documentación hacen esta herramienta realmente atractiva para automatizar tareas repetitivas.

Ansible es una herramienta que funciona sin agente, lo que quiere decir que no necesita instalación en el equipo cliente, y es multiplataforma, el único requisito necesario es que el sistema operativo cliente tenga Python 2.4 o superior. El sistema de autenticación lo hace por ssh, en paralelo y usando pares de claves privada/publica. Permite el uso de comandos básicos en los clientes y no se necesita acceso como root. Las tareas complejas utilizan YAML (Playbooks). YAML es un formato de serialización de datos sencillo de entender inspirado en lenguajes como XML, C, Python, Perl.

Ansible permite diferentes formas de configuración: una es mediante un solo fichero llamado playbook, que contendría todos los parámetros para hacer una determinada tarea sobre un determinado grupo de servidores o mediante una estructura de directorios por cada proyecto separando los parámetros en ficheros, que luego nos permitirán importarlos desde otros playbooks.

Ansible lo usaremos para orquestar los despliegues de los nodos tanto en el clúster de Proxmox como en el clúster de Cloudera, para ello hemos creado una serie de playbooks que realizan las tareas repetitivas. Este tipo de tareas críticas se llevan a cabo desde un nodo especial que se llama “orchestrator” basado en Centos 7.6 y Ansible 2.8, ver Figura 4.1. Este nodo es crítico ya que desde él se puede acceder a ambas redes y a todos los equipos de los dos clústers por lo tanto está protegido para sólo poder acceder desde una máquina concreta y nunca desde máquinas del clúster Cloudera. En la Figura 3.8 podemos ver la salida de un playbook que realiza la tarea de redimensionar los discos de todos los nodos sin necesidad de reiniciarlos o pararlos, las tareas se realizan en paralelo en todos los nodos.

4. Infraestructura

Ahora que conocemos la infraestructura de la que disponemos y de la tecnología que usaremos para el despliegue tendremos que definir la infraestructura del clúster de Cloudera. Un clúster de Cloudera requiere poder acceder a muchos servicios, cada uno en un puerto distinto por ello descartamos una pasarela dado que esta sería demasiado compleja. A continuación, describiremos cada uno de los elementos de la infraestructura que hemos creado y que se puede observar en la Figura 4.1.

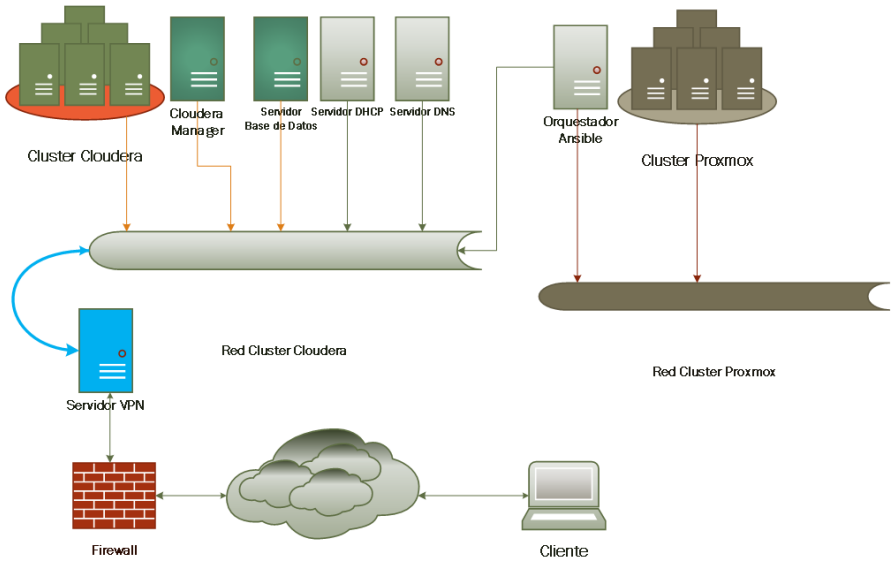

Figura 4.1: Infraestructura del Clúster de Cloudera

Orquestador

Este nodo es uno de los más importantes ya que es el encargado de desplegar las máquinas virtuales en el clúster de Proxmox y de configurar los nodos del clúster de Cloudera. Este servidor es el nodo de despliegue basado en Ansible y Centos 7.6. Entre sus funciones están todas las relacionadas con él despliegue en Proxmox. Estas son:

1. Clonar la máquina modelo.
2. Configuración de la máquina virtual, fijará el número de núcleos, tamaño de disco, cantidad de memoria, configuración de red y nodo de cómputo en el que se alojará.
3. Configurar el DHCP.
4. Configurar el Firewall si es necesario.
5. Redimensionar el disco de la máquina virtual.
6. Arrancar la máquina virtual.
7. Redimensionar el sistema de archivos de la máquina.

Servidor VPN

Debido al gran número de servicios que un clúster Cloudera necesita es necesario crear un acceso por VPN. Esto simplificará el acceso al clúster y asegurará el acceso al mismo. Buscar una solución basada en un firewall y NAT (Network Address Translation) haría que la gestión fuera compleja, ya que tendríamos un gran número de reglas en el firewall y estas habría que cambiarlas si se reasignan roles en el clúster. Utilizar direccionamiento IP público tiene el mismo problema por el número de filtros necesarios para proteger el sistema y no es recomendable emplear el direccionamiento público para este tipo de sistemas.

Ahora tendremos que decidir el tipo de VPN que usaremos. La premisa principal era no usar ningún tipo de software y que fuera fácil de instalar en cualquier plataforma. Para ello utilizaremos el protocolo L2TP.

L2TP es un protocolo de túnel seguro para transportar tráfico IP utilizando PPP. L2TP encapsula PPP en líneas virtuales que se ejecutan a través de IP, Frame Relay y otros protocolos. El estándar L2TP dice que la forma más segura de cifrar datos es usar L2TP sobre IPsec. Hay que tener en cuenta que es el modo predeterminado para el cliente Microsoft L2TP, ya que todos los paquetes de control y datos L2TP para un túnel particular aparecen como paquetes de datos UDP / IP homogéneos lo cual hace que la encriptación sea muy rápida. El tráfico L2TP usa el protocolo UDP para los paquetes de control y

de datos. El puerto UDP 1701 se usa solo para el establecimiento del enlace, el tráfico adicional está utilizando cualquier puerto UDP disponible (que puede o no ser 1701). Esto significa que L2TP se puede usar con la mayoría de los firewalls y enrutadores (incluso con NAT) al permitir que el tráfico UDP se enrute a través del firewall o enrutador. En nuestro caso es uno de los servicios reconocidos en EDUROAM y es por lo que decidimos crear esta infraestructura. Por lo tanto, usaremos L2TP sobre Ipsec con clave precompartida. La tecnología que hemos utilizado para el servidor VPN que hará las funciones de firewall, DHCP y DNS es de MikroTik con sistema operativo RouterOS 6.44, es una máquina virtual del sistema de virtualización.

La Red Interna

En la red interna desplegaremos el clúster de Cloudera, el direccionamiento en será privado con su propio servidor de nombres (DNS), servidor de DHCP y todos los nodos del clúster. Entendemos que la forma mejor de asegurar el sistema es poner un firewall antes del servidor VPN para poder dar acceso a la infraestructura. De esta forma un cliente puede trabajar desde cualquier punto con el clúster de una forma sencilla, y podemos controlar quien se conecta, desde donde lo hace y durante cuánto tiempo. Cómo hemos comentado anteriormente, los servicios de DNS, firewall y DHCP están integrados en la misma máquina virtual.

Servidor de Base de Datos

Este equipo será el encargado de mantener las bases de datos necesarias para los diferentes servicios de Hadoop. Se ha separado como un nodo independiente para poder tener la capacidad de escalar el servicio o cambiar de tipo de base de datos. Inicialmente será una un servidor de PostgreSQL, pero podría ser un clúster de MariaDB o un servidor de Oracle. Este nodo se creará como una especialización del modelo base que explicaremos en la siguiente Sección 4.1.

Servidor Cloudera Manager

Este nodo reside Cloudera Manager, que proporciona visibilidad y control sobre cada parte del clúster CDH, lo que permite a los administradores mejorar el rendimiento y mejorar la calidad del servicio. Debido a su importancia será un nodo dentro del clúster, pero sin ningún rol. Hemos decidido separarlo por la misma razón que el nodo de base de datos, esto nos permitirá no sobrecargar un nodo de cómputo con las labores de gestión del clúster.

Clúster Proxmox

Este clúster está formado por todos los nodos de computación y las cabinas de almacenamiento descritas en la Sección 3.1, el software instalado en los nodos de computación es Debian 9.9 y Proxmox 5.4 para el sistema de virtualización.

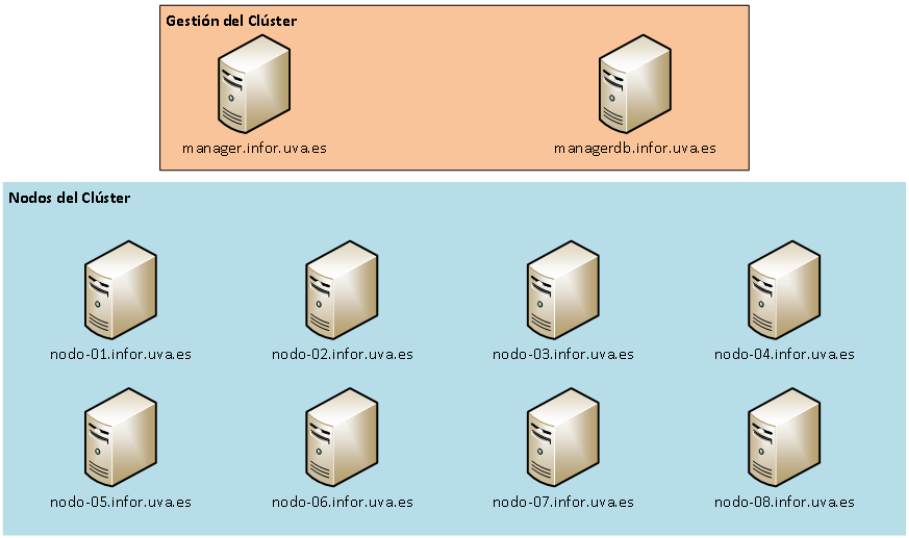

Figura 4.2: Esquema de Nodos del Clúster de Cloudera

Clúster Cloudera

El clúster Cloudera, ver Figura 4.2, está formado por los dos nodos de gestión y 8 nodos que desempeñan distintos roles dentro del clúster como se puede ver en la Tabla 4.8.

4.1 Modelo Base

Una vez que tenemos todos los elementos necesarios, tan sólo nos queda diseñar el procedimiento para desplegar clústeres Cloudera. En la Figura 4.3 podemos ver el diagrama de flujo que hemos seguido para intentar minimizar las operaciones necesarias para desplegar un clúster. Gracias a Ansible definimos operaciones que son específicas para el despliegue en Proxmox, pero que pueden ser adaptadas para desplegar en nodos físicos o en nodos en clouds, como AWS. La idea básica de este flujo de trabajo es disponer de un modelo base, lo más sencillo posible para que el despliegue sea rápido y lo podamos adecuar a distintas funcionalidades, ya sea para gestión de base de datos o un nodo más en el clúster. Este sistema reducirá considerablemente los tiempos de administración y gestión.

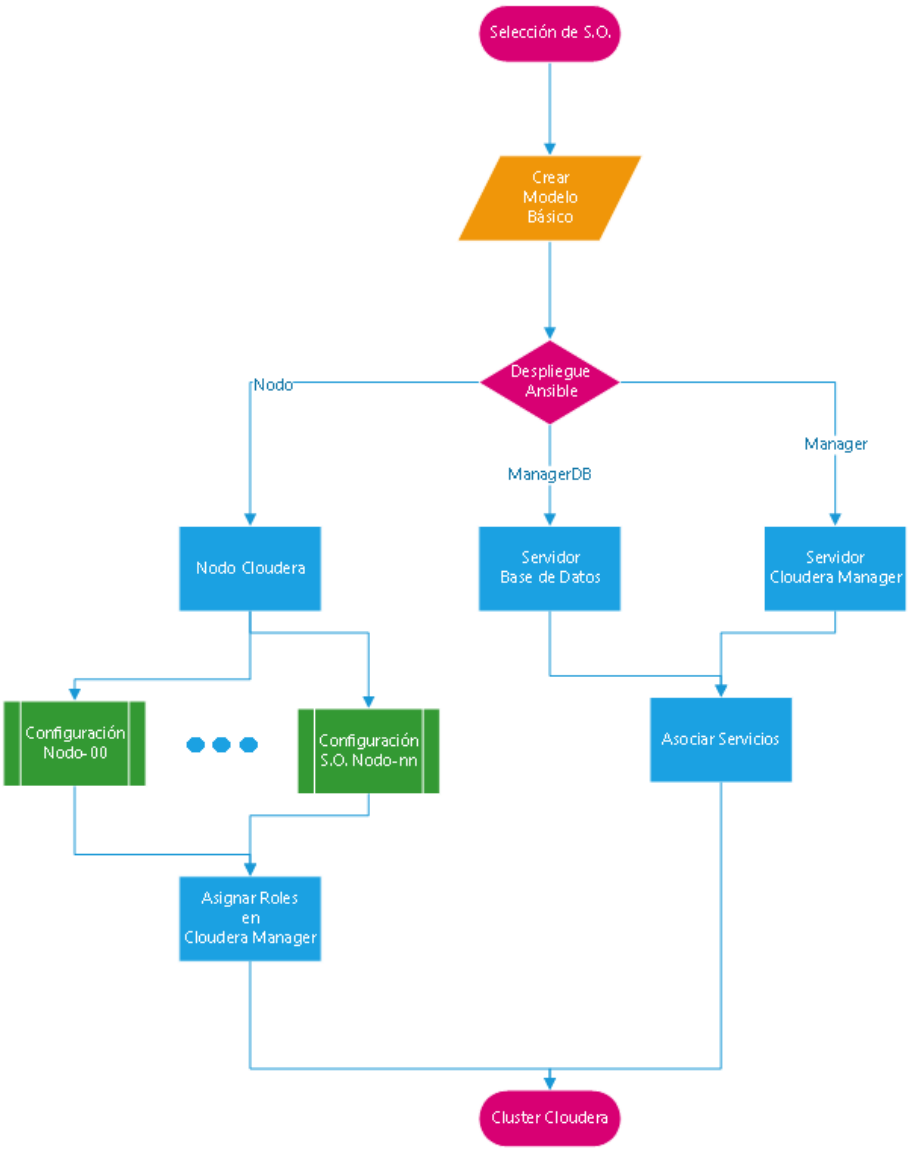

Figura 4.3: Flujo de Despliegue

Selección de S.O.

En esta sección describiremos qué sistema operativo seleccionamos para el modelo base y cómo configuramos el modelo que desplegaremos para crear los nodos. Cloudera Manager es compatible con los sistemas de la Tabla 4.1. Por el tipo de licencias de RedHat Enterprise Linux conlleva un coste desde 349$ a 18.000$ por subscripción lo cual nos llevó a descartar esta distribución. Probamos la distribución RedHat Entrerprise en su versión de prueba, es robusta y eficiente, muy recomendable para entornos empresariales, de hecho, Red Hat Enterprise Linux es la distribución número uno a nivel empresarial. Oracle Linux es otra distribución que se descartó por el coste económico y el tipo de licencia. Esta distribución dispone de agentes para despliegue en la plataforma Azure y compatible con

Sistema operativo	Versión
RHEL / CentOS	7.6, 7.5, 7.4, 7.3, 7.2, 6.10, 6.9, 6.8
Oracle Linux (OL)	7.6, 7.4, 7.3, 7.2, 6.10
SUSE Linux Enterprise Server	12 SP3, 12 SP2
Ubuntu	18.04 LTS (Bionic), 16.04 LTS (Xenial)

Tabla 4.1: Tabla de S.O. Soportados

todos los sistemas de Cloudera, simplificando en gran medida los despliegues de grandes clústeres de Cloudera, otra de sus ventajas es su compatibilidad con Oracle. Si se va a crear un gran clúster que use una base de datos Oracle, esta es la distribución más adecuada.

Tan solo nos quedan dos distribuciones que por la licencia no generan ningún coste que son Ubuntu y CentOS. Finalmente nos decidimos por CentOS, porque es la más parecida a Oracle Linux y RedHat Enterprise, que son las distribuciones más extendidas en las empresas y con gran cantidad de información en la red. La base conocimiento de RedHat Enterprise, bajo subscripción, inmejorable. CentOS es una distribución muy estable y es la versión de RedHat de código abierto, por esta razón hemos seleccionado CentOS como la distribución base para construir un modelo básico.

Dependencias de Software

Una vez que tenemos el sistema operativo seleccionado, debemos ver cuáles son las dependencias del software que necesita Cloudera Manager 6.2. Cloudera Manager no es compatible con Python 3.x y Hue necesita Python 2.7, con lo cual al momento de instalar la distribución de Centos 7.6 hay que disponer de Python 2.7. Otros paquetes de software necesarios son, Perl e iproute, para el caso de CentOS, es necesario la versión iproute 3.10, utilizado por el agente de Cloudera Manager. Por último, python-psycopg2 necesario para Cloudera Manager 6.2 y Hue.

Sistemas de Archivos

El sistema de archivos del sistema operativo es importante, ya que debe de ser compatible con todas las herramientas que se monten en el clúster. Ext3 es el sistema más probado para HDFS pero es un sistema de archivos un tanto anticuado. Evaluamos Ext4 y XFS decantándonos por XFS como sistema de archivos base. XFS es un sistema de archivos de 64 bits transaccional de alto rendimiento, las razones principales por las que nos decantamos por este sistema de archivos es que soporta archivos muy grandes y sistemas de archivos con un gran número de archivos. Tiene un gran sistema de recuperación frente a fallos, se puede desfragmentar y expandir en caliente. Dispone de sistemas de pre-asignación de espacio, que sirve para evitar la fragmentación, proporciona excelente estabilidad y

<domain>	<type>	<item>	<value>
cloudera-scm	soft	nofile	32768
cloudera-scm	hard	memlock	unlimited
cloudera-scm	soft	nproc	65536
cloudera-scm	hard	nofile	1048576
cloudera-scm	hard	nproc	unlimited
cloudera-scm	hard	memlock	unlimited

Tabla 4.2: Valores de *limits*

escalabilidad de E/S mediante árboles-b para indexar todos los datos y metadatos de usuarios. En modo beta puede comprimir y deduplicar los bloques de almacenamiento lo cual es tremendamente útil para un DataNode. Y por supuesto es compatible con todas las herramientas de Hadoop incluida Kudu que requiere una serie de características especiales.

> **Nota 4.1** Es muy importante la configuración de nproc, donde se establecen los límites del sistema operativo para el usuario *cloudera-scm*, por ejemplo límite de procesos o de archivos. La automatización del despliegue de Cloudera Manager lo establece en /etc/security/limits.d/cloudera-scm.conf en todos los nodos con los valores que se muestran en la Tabla 4.2 y pueden ser variados según las necesidades del clúster.

Los demás de requisitos

Otro de los requisitos necesarios es la versión de Java ya que se debe instalar Oracle JDK 1.8 o OpenJDK 1.8. Se requiere un servidor DNS o incluir todos los nodos en el fichero /etc/hosts. Hay que tener en cuenta que no se admiten alias, por lo tanto, se debe hacer de forma correcta, por ejemplo: *192.168.1.1 cluster-01.example.com cluster-01*.

> **Nota 4.2** En el caso de usar Kudu es altamente recomendable usar nscd para almacenar en caché la resolución de nombres DNS y la resolución de nombres estática para Kudu. Esto es necesario debido al gran número de peticiones que realiza este servicio al servidor DNS.

En nuestro caso, para simplificar la configuración del clúster hemos creado un servidor DHCP para asignar la configuración de red en el arranque así los despliegues son más sencillos.

Los nodos no deben tener firewall y deben tener el puerto SSH abierto. Cloudera Manager y CDH utilizan varias cuentas de usuario y grupos para completar sus tareas. El conjunto de cuentas de usuario y grupos varía según los componentes que se instalan en los nodos. No se debe eliminar estas cuentas o grupos ni modificar los permisos. Hay que

asegurarse de que ningún sistema existente impida que estas cuentas y grupos funcionen ya que Cloudera Manager ni el CDH comprueban si son modificadas.

4.2 Proceso de instalación del modelo base

En esta sección vamos a describir, paso a paso el proceso de instalación del modelo. La memoria del modelo será 8Gb con 4 Cores y 10 GB de disco. Estos datos se han obtenido en las pruebas de los diferentes sistemas operativos. Con 10 GB de disco es suficiente para desplegar todo el software que necesitamos y de esta manera cuando clonamos sólo duplicamos 10Gb que es el espacio mínimo requerido. Los pasos necesarios para la instalación son:

- Particionado del disco. Hacemos una única partición sin swap y activa para que arranque el sistema de ella.
- Instalamos el sistema operativo mínimo CentOS 7.6.1810, con el servicio SSH.
- Eliminamos cualquier cuenta de usuario que creáramos en la instalación. Por lo tanto, sólo existirá la cuenta root en el modelo y todos los comandos que se realicen a continuación se realizarán con el usuario root.
- Creamos un archivo para el swap de sistema. Utilizamos un archivo ya que es más cómodo que una partición y al tener el modelo de una única partición, nos permite escalar el disco de una forma muy sencilla.

  Las instrucciones para crear un archivo de swap son:

  ```
  dd if=/dev/zero of=.swapfile bs=1G count=1
  chmod 600 .swapfile
  mkswap .swapfile
  ```

  Editamos `/etc/fstab` y añadimos:

  ```
  /.swapfile    none    swap    sw 0 2
  ```

- Cambiar el comportamiento de la swap. Esto es muy importante para evitar que se use el swap. La recomendación es que el valor esté por debajo de 10. En nuestro caso vamos a intentar que no se use el swap salvo que sea absolutamente necesario. Esto se hace cambiando unas propiedades del kernel, para ello en el archivo */etc/sysctl.conf*, añadimos *vm.swappiness = 1*. De esta forma podemos monitorizar el uso de swap y en el caso de que empiece a usarse emitir una alarma al administrador del sistema ya que es una situación no desada en un nodo de cómputo.
- Desactivación de SELINUX. Es un módulo de seguridad para el kernel Linux que proporciona el mecanismo para soportar políticas de seguridad para el control de

acceso. Se trata de un conjunto de modificaciones del núcleo y herramientas de usuario. Su arquitectura se enfoca en separar las decisiones de las aplicaciones de seguridad de las políticas de seguridad mismas y racionalizar la cantidad de software encargado de las aplicaciones de seguridad. Es incompatible con Cloudera y se desactiva cambiado en el fichero */etc/selinux/config* la variable *SELINUX=disabled*. Requiere reiniciar la máquina para que tenga efecto.

- Desactivación de IPV6. Se desactiva cambiado unas propiedades del kernel en el arranque. Nosotros no vamos a usar direccionamiento IPV6 por lo tanto es mejor desactivarlo.

  Modificaciones para deshabilitar IPV6.

  Añadir al archivo /etc/sysctl.conf:

  ```
  net.ipv6.conf.all.disable_ipv6 = 1
  net.ipv6.conf.default.disable_ipv6 = 1
  ```

  En el archivo /etc/default/grub modificar variable GRUB_CMDLINE_LINUX con la opción del kernel ipv6.disable=1

  No olvidar generar el archivo de arranque nuevo:

  ```
  grub2-mkconfig -o /boot/grub2/grub.cfg
  ```

- Cómo habíamos comentado, para simplificar el sistema de red usamos un servidor DHCP que nos dará el nombre del nodo y toda la configuración de red, de esta forma todos los nodos del clúster Cloudera son iguales.

  Para configurar el DHCP editamos el archivo

  /etc/sysconfig/network-scripts/ifcfg-eth0

  con el siguiente contenido:

  ```
  TYPE=Ethernet
  PROXY_METHOD=none
  BROWSER_ONLY=no
  BOOTPROTO=dhcp
  DEFROUTE=yes
  IPV4_FAILURE_FATAL=no
  IPV6INIT=no
  IPV6_AUTOCONF=no
  IPV6_DEFROUTE=no
  IPV6_FAILURE_FATAL=no
  #IPV6_ADDR_GEN_MODE=stable-privacy
  NAME=eth0
  DEVICE=eth0
  ONBOOT=yes
  ```

- Transparent Huge Page Compaction debe ser deshabilitada para no afectar al rendi-

miento. Esta configuración del kernel es otra de las recomendaciones para nodos de cómputo.

> Configuración de Transparent Huge Page Compation
>
> Añadir al archivo /etc/rc.local los siguientes comandos:
>
> ```
> echo never > /sys/kernel/mm/transparent_hugepage/defrag
> echo never > /sys/kernel/mm/transparent_hugepage/enabled
> ```
>
> Este fichero se ejecutará al terminar el arranque del sistema pero para ello no hay que olvidar poner permisos de ejecución al archivo.
>
> ```
> chmod +x /etc/rc.local
> ```

- Si no desactivamos el firewall Cloudera Manager no se puede comunicarse con los nodos.

> Desactivación del firewall
>
> ```
> systemctl stop firewalld
> systemctl disable firewalld
> ```
>
> No hay que olvidar comprobar que la política por defecto es ACCEPT:
>
> ```
> iptables -L
> ```

- Los nuevos sistemas ocultan las instrucciones de arranque y limpian el terminal. Para evitar esto y poder analizar el arranque de los nodos lo deshabilitamos.

> Activar el modo verboso.
>
> ```
> mkdir /etc/systemd/system/getty@tty1.service.d
> ```
>
> Crear en ese directorio el fichero noclear.conf con el siguiente contenido:
>
> ```
> [Service]
> TTYVTDisallocate=no
> ```
>
> Suprimir las opciones gráficas rhgb y el modo silencioso quiet del fichero /etc/default/grub definidas en la variable GRUB_CMDLINE_LINUX.
>
> Deberemos rehacer el fichero de configuración de arranque:
>
> ```
> grub2-mkconfig -o /boot/grub2/grub.cfg
> ```

- Un clúster necesita que todos sus nodos tengan una sincronización de tiempo, para ello instalamos el servicio NTP en todos los nodos y sincronizamos con el servidor de tiempos de nuestra red en el arranque. Es igualmente importante que tengan lo

zona horaria adecuada.

Fijar la zona horaria:

```
ln -f -s /usr/share/zoneinfo/Europe/Madrid
/etc/localtime
```

Sincronización de tiempo.

```
yum install ntp
```

Editamos el fichero /etc/sysconfig/ntpdate y la variable OPTIONS añadimos el servidor de tiempos local, en nuestro caso, ns1.infor.uva.es

Configuramos el servicio para que sincronice la hora en el arranque.

```
systemctl enable ntpdate
systemctl start ntpdate
```

- La configuración del servidor saliente de correo es importante para el envío de notificaciones de los nodos del clúster. CentOS suele tener instalado el servicio *postfix* y hay que configurar el relay de correo de la red o no se enviarán los correos fuera del nodo.

Configuración e instalación de postfix :

```
yum install postfix
```

Editar el fichero /etc/postfix/main.cf y establecer la variable relayhost con el nombre del servidor de correo saliente de la red, en nuestro caso:

```
relayhost = [mail-server.infor.uva.es]
```

- Configuración del servicio SSH para la conexión entre nodos mediante clave pública.

Configuración de clave para el servicio SSH

Generamos los ficheros de clave privada/publica:

```
ssh-keygen
```

Autorizamos al nodo a conectarse a si mismo y por lo tanto a sus clones.

```
cd ~/.ssh
cat id\_rsa.pub > authorized\_keys
```

- Instalamos ciertos paquetes para monitorizar el nodo y herramientas del sistema de virtualización, como es el agente qemu, usado por el sistema de virtualización para monitorizar la máquina virtual o SPICE, que son los drivers necesarios para el tipo de consola. Algunos de estos paquetes necesitan el repositorio de software EPEL que no se instala por defecto con CentOS y es necesario.

Instalación del repositorio EPEL.

```
yum install epel-release
```

Instalación de paquetes de monitorización:

```
yum install sysstat net-tools wget bind-utils iotop htop psmisc
```

Herramientas para Poxmox, agente qemu y driver para pantallas SPICE:

```
yum install qemu-guest-agent spice-vdagent
```

- Configuración de los *locales* del sistema. Este punto es importante ya que para la codificación de los archivos y de la base de datos es necesario que sea en UTF-8.

Configuración del locale de la máquina.

```
echo 'LC\_ALL='"en_US.UTF-8"' >> /etc/locale.conf
echo 'LANG="es_ES.UTF-8"' >> /etc/locale.conf
```

- Instalación de *expect*. Facilita el trabajo al hacer posible la automatización de algunas tareas como los scripts desatendidos. Esta librería permite automatizar las respuestas de los scripts interactivos.

Instalación de Expect:

```
yum install expect
```

- Instalación de paquetes necesarios para la base de datos. En nuestro caso al haber selecciona PostgreSQL como base de datos necesitamos instalar *psycopg2*, pero como usaremos Hue este tiene que ser superior a la versión 2.7.5 y en CentOS se instala una versión inferior (la 2.5.1-3). En nuestro modelo base no sabemos en qué nodos desplegaremos el rol de Hue, por lo tanto, lo instalamos en todos ya que ocupa poco y de esta forma hacemos que el modelo valga para cualquier rol.

Instalación de pip, para poder instalar el paquete psycopg2 en la versión deseada.

```
yum install python-pip
```

Instalación de psycopg2

```
pip install psycopg2==2.7.5 --ignore-installed
```

Con el proceso que hemos descrito habremos instalado un modelo que podemos especializar. Como vimos en la Figura 4.1, tenemos dos nodos especializados, el servidor

de base de datos del clúster y el servidor de Cloudera Manager, el modelo base que hemos creado nos servirá para crear este tipo nodos y el resto de nodos del clúster de Cloudera.

4.3 Servidor de Base de Datos

Si se quiere usar Cloudera Manager y CDH en un entorno de producción se debe usar una infraestructura de base de datos externa. Cloudera Manager y CDH vienen empaquetados con una base de datos PostgreSQL, pero esta configuración no es adecuada para un uso intensivo, lo ideal es usar un clúster de bases de datos. Cloudera Manager usa varias bases de datos y almacenes de datos para almacenar información sobre su propia configuración, así como información sobre el estado del sistema o el progreso de la tarea. En nuestro caso, hemos creado un servidor de base de datos externos con una base de datos postgreSQL. La razón fundamental es que postgreSQL admite extensiones GIS, que pueden ser interesantes para ciertas aplicaciones y es una extensión bastante conocida. Utilizar Oracle queda fuera de nuestro alcance por un tema de licencias y dado que tampoco deseábamos crear un clúster de bases de datos, descartamos también MariaDB, que es más apropiada para un clúster de tamaño medio o grande que no es nuestro caso. En las pruebas que realizamos con MariaDB con un sólo nodo de base de datos, pudimos observar que funcionaba perfectamente al igual que con postgreSQL, pero sin aportar mayores ventajas.

4.3.1 Proceso de instalación

El proceso de instalación es sencillo ya que partimos del modelo base y tan solo hay que instalar unos pocos paquetes software y configurarlos. La base de datos la usan varios componentes de Cloudera, como son:

- Servidor Cloudera Manager, encargado de contener toda la información sobre los servicios que se han configurado en el clúster. Es una base de datos pequeña inferior a 100MB pero es muy importante hacer copia de seguridad.
- Servidor Oozie, encargado de los flujos de trabajo y coordinación de los mismos. Puede llegar a ser una base de datos grande.
- Servidor Sqoop, es una base de datos pequeña que contiene entidades como el conector, el controlador, los enlaces y los trabajos.
- Monitor de actividad, contiene información sobre las actividades pasadas, es una base de datos necesaria para MapReduce. Puede llegar a ser una base de datos muy grande.
- Administrador de informes, una base de datos de tamaños medio, se utiliza para realizar el seguimiento de la utilización del disco y las actividades de procesamiento

a lo largo del tiempo.

- Servidor Metastore Hive, una base de datos pequeña donde se almacenan los metadatos de Hive.
- Servidor Hue, mantiene la información de Hue, cuentas de usuario, envío de trabajos y consultas Hive, no suele ser una base de datos grande.
- Servidor Sentry, una base de datos pequeña con los metadatos de seguridad.
- Servidor de auditoría Cloudera Navigator, esta base de datos puede ser muy grande si los grupos de trabajo son grandes, contiene información de auditoría.
- Servidor de metadatos Cloudera Navigator, contiene políticas, autorizaciones y metadatos de informes de auditoría, es relativamente pequeña.

Lo que haremos será clonar el modelo base y crearemos una máquina virtual nueva con las siguientes características:

- Nombre: managerdb.infor.uva.es
- Memoria: 2GB
- CPU: 4 Cores con NUMA
- Disco: 25 GB

Dado que el nombre y la configuración de red es automática ya que la proporciona el servidor DHCP, tan solo nos queda realizar las siguientes operaciones.

Instalación de PostgreSQL

```
yum install postgresql-server
```

Las instrucciones de configuración del servidor PostgreSQL son igualmente simples, es importante tener bien los *locales* del sistema, pero esto ya lo definimos en el modelo base descrito en la sección 4.2.

Inicialización de la base de datos

```
sudo su -l postgres -c "postgresql-setup initdb"
```

Una vez inicializada la base de datos hay que configurar la autenticación MD5. Dado que estamos en una red privada y tenemos que conectarnos a la base de datos desde los nodos, habilitamos cualquier red para conectarse al servidor.

Ajustes de autenticación

Editar /var/lib/pgsql/data/pg_hba.conf y cambiar la linea

```
host all all 127.0.0.1/32 ident
```

por

Servicio	Base de Datos	Usuario
Cloudera Manager	scm	scm
Activity Monitor	amon	amon
Reports Manager	rman	rman
Hue	hue	hue
Hive Metastore Server	metastore	hive
Sentry Server	sentry	sentry
Cloudera Navigator Audit Server	nav	nav
Cloudera Navigator Metadata Server	navms	navms
Oozier	oozie	oozie
Sqoop2	sqoop	sqoop

Tabla 4.3: Bases de Datos para Cloudera

```
host all all 127.0.0.1/32 md5
```

Ajustes de conexión.

Editar */var/lib/pgsql/data/postgresql.conf* y definir la variable siguiente:

```
listen_addresses = '*'
```

Ajustamos los parámetros de la base de datos para un clúster de tamaño pequeño / medio. Necesitamos ajustar el número máximo de conexiones, el tamaño de memoria compartida,

Ajustes del servicio PostgreSQL

```
\lstset{basicstyle=\scriptsize ,language=sh}
max_connection=100
```

shared_buffers = 256MB Memoria compartida.

checkpoint_completion_target = 0.9 fracción de tiempo entre puntos de control.

wal_buffers = 16 MB 3 % de shared_buffers

```
checkpoint_segments = 16
```

Si la versión de postgreSQL es superior a la 9.5 se usa min_wal_size y max_wal_size según la siguiente fórmula: max_wal_size = (3 ∗ checkpoint_segments) ∗ 16MB

La instalación termina habilitando el servicio en el arranque y creando las bases de datos para los diferentes servicios como se muestran en la Tabla 4.3.

Habilitar el servicio PostgreSQL y reiniciarlo.

```
systemctl enable postgresql
systemctl restart postgresql
```

Creamos los usuarios y las bases de datos según la Tabla 4.3:

```
sudo -u postgres psql
CREATE ROLE <usuario> LOGIN PASSWORD '<clave>'
CREATE DATABASE <base de datos> OWNER <usuario> ENCODING 'UTF8';
```

Para las bases de datos de Metastore y Oozie si instalamos PostgreSQL 8.4 y superiores se necesita establecer standard_conforming_strings =off:

```
ALTER DATABASE metastore SET standard_conforming_strings=off;
ALTER DATABASE oozie SET standard_conforming_strings=off;
```

Crear base de datos externa para Sqoop2 es distinto:

```
CREATE ROLE sqoop LOGIN ENCRYPTED PASSWORD '<clave>' NOSUPERUSER INHERIT CREATEDB
    NOCREATEROLE;
CREATE DATABASE "sqoop" WITH OWNER = sqoop ENCODING = 'UTF8' TABLESPACE =
    pg_default CONNECTION LIMIT = -1;
```

Con este proceso tenemos configurado y listo para pasar a producción el servidor de base de datos del clúster Cloudera.

4.4 Servidor de Cloudera Manager

Configurar el servidor de Cloudera Manager es un proceso bastante automatizado. Lo primero que necesitamos es crear una máquina virtual del modelo base con las siguientes características:

- Nombre: manager.infor.uva.es
- Memoria: 5GB
- CPU: 4 Cores con NUMA
- Disco: 10 GB

Para instalar Cloudera Manager necesitaremos añadir un repositorio de software que proporciona Cloudera para la instalación y el JDK de Java.

Repositorio de Cloudera

```
wget https://archive.cloudera.com/cm6/6.2.0/redhat7/yum/cloudera-manager.repo -P /
    etc/yum.repos.d/
```

La instalación de Java Development Kit es otro de los requisitos. Podemos instalar Oracle JDK o OpenJDK . La mayoría de las distribuciones de Linux compatibles con Cloudera incluyen OpenJDK. Cloudera Manager no es compatible con la versión de 32 bits, es necesario instalar la versión de JDK 64bits que deseemos. En nuestro caso, hemos optado por la versión OpenJDK por el tipo de licencia.

Opción	Descripción
-? | –help	Ayuda
–config-path	La ruta a los archivos de configuración de Cloudera Manager Server. El valor predeterminado es /etc/cloudera-scm-server.
-f | –force	Si se especifica, la secuencia de comandos no se detiene si se produce un error.
-h | –host	La dirección IP o el nombre de host del servidor de base de datos. El valor predeterminado es utilizar localhost.
-p | –password	Contraseña del usuario de la base de datos
-P | –port	CEl número de puerto que se utilizará para conectarse a la base de datos. El puerto predeterminado es 3306 para MariaDB, 3306 para MySQL, 5432 para PostgreSQL y 1521 para Oracle. Esta opción se usa solo para una conexión remota.
–scm-host	El nombre de host donde está instalado el Servidor de Cloudera Manager. Si el servidor de Cloudera Manager y la base de datos están instalados en el mismo host, no use esta opción o la-h opción.
–scm-password-script	Un script para ejecutar cuyo stdout proporciona la contraseña para el usuario SCM (para la base de datos).
-u | –user	El nombre de usuario de administración para la base de datos. Utilizar con la opción -p. No pongas un espacio entre -u y el nombre de usuario (por ejemplo, -uroot). Si se proporciona esta opción, el script crea un usuario y una base de datos para el servidor de Cloudera Manager. Si ya ha creado la base de datos, no use esta opción.

Tabla 4.4: opciones de scm_prepare_database.sh

Instalación de JDK.

OpenJDK

```
yum install java-1.8.0-openjdk-devel
```

OracleJDK

```
yum install oracle-j2sdk1.8
```

Tan solo con queda instalar los paquetes de Cloudera Manager en el host del servidor manager.infor.uva.es que es nuestro nodo de Cloudera Manager.

Instalación de Cloudera Manager.

```
yum instalar cloudera-manager-daemons cloudera-manager-agent cloudera-manager-
    server
```

Para finalizar la instalación tan solo queda configurar la base de datos que usará Clou-

Parámetro (**negrita obligatorio**)	Descripción
Tipo Base de Datos	mysql, oracle o postgresql
Nombre Base de Datos	El nombre de la base de datos que usará de Cloudera Manager.
Usuario Base de Datos	El nombre de usuario de la base de datos de Cloudera Manager Server
contraseña	Contraseña del usuario de la base de datos

Tabla 4.5: Parámetros de scm_prepare_database.sh

dera Manager. Para ello usaremos el script scm_prepare_database.sh que configura el servidor de base de datos para el servicio. El comando se encuentra en /opt/cloudera/cm/schema/ y tiene el siguiente la sintaxis:

scm_prepare_database.sh [opciones] <Tipo Base de Datos> <Nombre Base de Datos> <Usuario Base de Datos> <clave>

La descripción de los parámetros se puede ver en la Tabla 4.5 y la de las opciones en la Tabla 4.4.

Instalación de Cloudera Manager.

```
scm_prepare_database.sh postgresql -h managerdb.infor.uva.es --scm-host manager.
    infor.uva.es scm scm -p cloudera-manager-server
```

Ya sólo resta arrancar el servicio de Cloudera Manager. Este servicio arrancará la aplicación en http://<nombre-del-servidor>:7180. En el primer arranque hay que esperar unos minutos hasta que el servicio está preparado.

Arranque de Cloudera Manager.

```
systemctl start cloudera-scm-server
```

A partir de este momento el proceso consistirá en seguir el asistente de instalación, al cual accederemos http://manager.infor.uva.es:7180, el usuario y clave iniciales son admin, admin. Previamente nos habremos conectado a la VPN.

Con el playbook de Ansible que hemos creado clonamos 8 nodos que repartimos por el clúster de Proxmox según la Tabla 4.6. Los equipos se llamarán nodo-01.infor.uva.es a nodo-08.infor.uva.es. La distribución de los nodos viene dada por las capacidades de las placas de cómputo, estos pueden cambiar según las necesidades del cluster de vitualización. Las características de cada nodo se han calculado según los roles que van a alojar.

Las fases del asistente son:

1. **Bienvenida**, Figura 4.4

Nombre	Memoria	Núcleos	Disco	Placa de Cómputo
nodo-01	80GB	16	50GB	Dell PowerEdge R71, Tabla 3.1
nodo-02	64GB	24	1TB	IMS Blade 4, Tabla 3.1
nodo-03	32GB	8	1TB	SuperMicro CSE, Tabla 3.1
nodo-04	40GB	16	1TB	IMS Blade 4, Tabla 3.1
nodo-05	32GB	8	1TB	Intel ECS IXION, Tabla 3.1
nodo-06	64GB	12	1TB	SuperMicro CSE, Tabla 3.1
nodo-07	64GB	12	1TB	Dell PowerEdge R71, Tabla 3.1
nodo-08	5GB	8	1TB	batty, Tabla 3.1

Tabla 4.6: Características de los nodos clonados

2. **Aceptación de la licencia**.

3. **Selección de la licencia**, Figura 4.5. Seleccionaremos Cloudera Express, con lo que no dispondremos de características avanzadas, como creación de roles de usuario, gestión de permisos, copias de seguridad, etc.

4. **Pantalla de bienvenida a la instalación del clúster**.

5. **Especificar los nodos**, en esta fase se indica los nombres de los nodos en los que se instalará CDH, Figura 4.6. Cloudera Manager los buscará en la red e indica cuales encuentra de los que se han especificado, hay una limitación de 100 nodos, esto es por la licencia que hemos seleccionado, ver Sección 2.3.3.

6. **Seleccionar el Repositorio**. Dado que nuestros nodos tienen acceso a Internet usaremos el repositorio público. En el caso de no tener acceso a Internet tendremos que indicar un repositorio personalizado que nos proporcione los paquetes necesarios. Ver Figura 4.7.

7. **Indicar la contraseña de acceso a los nodos**, en nuestro caso es la misma y utilizaremos el usuario root para el despliegue. Se puede usar otro usuario pero este debe tener capacidad de escalar privilegios como root, lo cual ralentiza la instalación y la complica, por otra parte es un método más seguro.

8. **Instalación del agente**. Figura 4.8, en esta fase el asistente instala en todos los nodos el agente Cloudera, este se encargará de los latidos e informar a Cloudera Manager del estado del nodo, se usa para los despliegues, estadísticas, etc.

9. **Instalación de Parcels**, en esta fase instala los paquetes que hemos seleccionado anteriormente. Al haber seleccionado CDH, este despliega todos los parcels opcionales como son Spark, Kafka o Kudu.

10. **Inspección de nodos**, al terminar realiza una inspección de los nodos y avisa de los fases correctas y incorrectas que encuentre en el clúster.

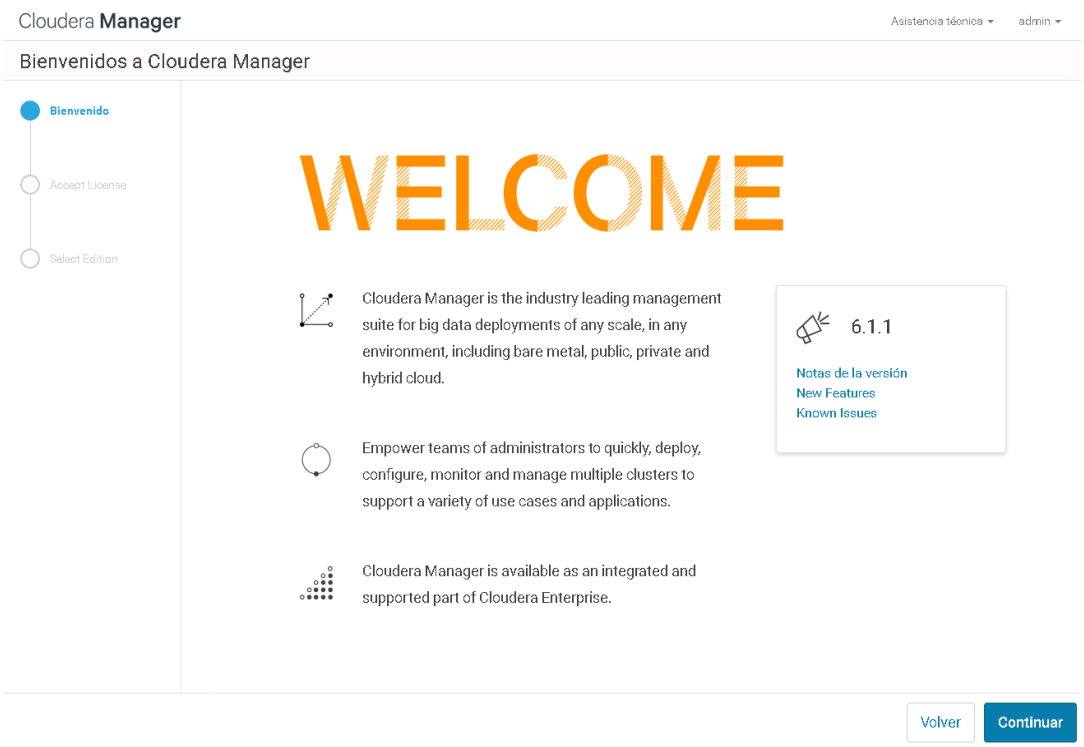

Figura 4.4: Pantalla inicial

4.5 Instalación del Clúster Cloudera

Ahora pasaremos a instalar el clúster en sí, en esta fase decidiremos que roles desempeña cada nodo, para desplegar el software en cada uno con Cloudera Manager. Para saber que roles asignamos a cada nodo nos guiaremos por la Tabla de dependencias de roles 4.7 y por las capacidades de cómputo del rol.

4.5.1 Seleccionar Servicios

En esta fase podemos seleccionar los servicios que desea instalar y configurar. La selección de servicios está vinculada al tipo de licencia y se puede elegir entre:

1. **Essentials**, incluye HDFS, YARN (MapReduce 2 incluido), ZooKeeper, Oozie, Hive y Hue.

2. **Data Engineering**, incluye HDFS, YARN (MapReduce 2 incluido), ZooKeeper, Oozie, Hive, Hue y Spark.

	ADLS Connector or WS S3	KMS[a]	Data Context Connector	Flume	HBase	HDFS	Hive	Hue	Impala	Kafka	Key-Value Store Indexer	Oozie	Sentry	Solr	Spark on YARN	YARN	ZooKeeper	Kudu
ADLS Connector																		
Data Context Connector																		
Flume					✓	✱				✱								
HBase						✓											✓	
HDFS	✱	✱			✓				✓				✓	✓		✓	✱	
Hive					✱			✓	✓					✓		✓	✱	✱
Hue					✱		✓									✓	✱	✱
Impala					✱		✓									✓	✱	✱
Kafka				✱													✓	
Key-Value Store Indexer					✓									✓				
Oozie							✓									✓	✱	✱
Sentry														✱			✱	
Solr						✓							✓				✓	
Spark on YARN					✱											✓		
YARN						✓											✱	
ZooKeeper																		

✓= Obligatorio ✱= Optativo

Tabla 4.7: Tabla de dependencias de servicios en Cloudera 6.2

[a]Key Management System, hace referencia a sistemas de encriptación y manejo de claves, cómo KMS, Thales KMS, Key Trustee, or Luna KMS

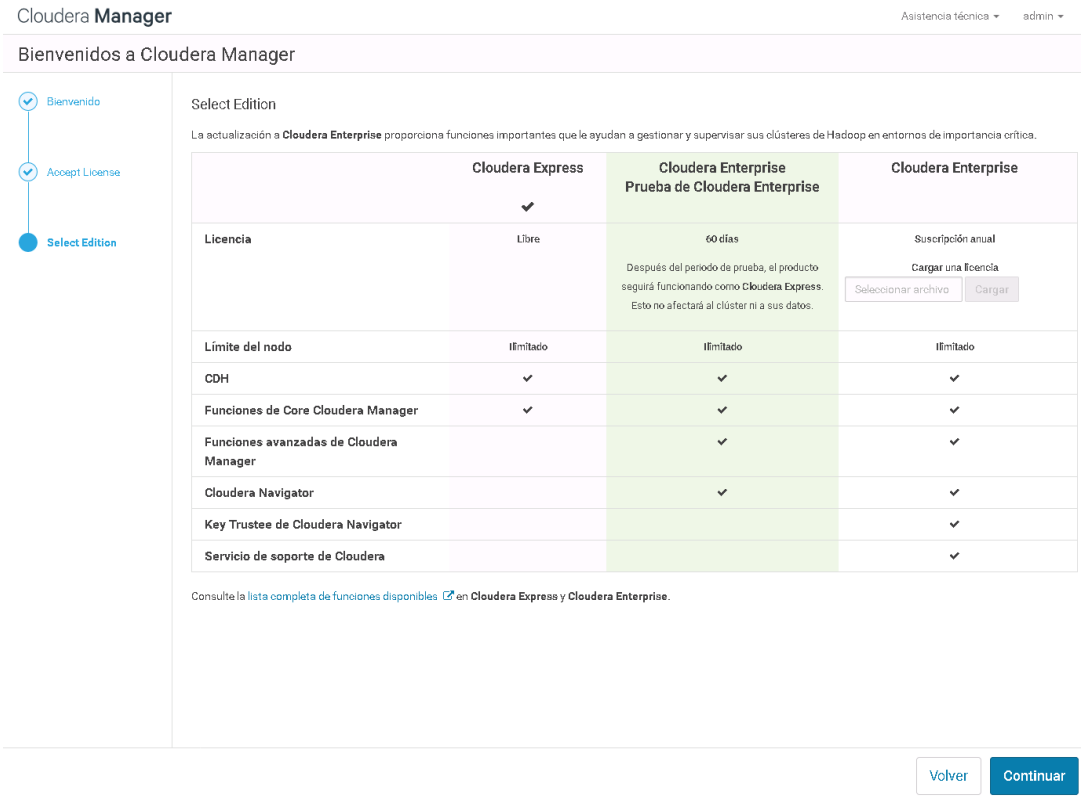

Figura 4.5: Selección de Licencia

3. **Data Warehouse**, incluye HDFS, YARN (MapReduce 2 incluido), ZooKeeper, Oozie, Hive, Hue e Impala.

4. **Operational Database**, incluye HDFS, YARN (MapReduce 2 incluido), ZooKeeper, Oozie, Hive, Hue y HBase.

5. **Todos los servicios incluyen**, HDFS, YARN (MapReduce 2), ZooKeeper, Oozie, Hive, Hue, HBase, Impala, Solr, Spark y Key-Value Store Indexer.

6. **Servicios personalizados**, en este caso podemos elegir los servicios uno por uno.

En nuestro caso seleccionamos todos los servicios ya que es uno de los requisitos del proyecto, disponer de todos en un único clúster.

4.5.2 Distribución de roles

Distribuir los roles en cada uno de los nodos es una tarea que dependerá de las características de cada nodo. Cuantos más roles tenga un nodo más consumo de memoria tendrá. Por otra parte, los servicios tienen dependencias unos con otros como se puede ver en la Tabla 4.7, es conveniente que algunos residan juntos como por ejemplo el rol

	nodo-01	nodo-02	nodo-03	nodo-04	nodo-05	nodo-06	nodo-07	nodo-08
Agente Flume	✓	✓	✓	✓	✓	✓	✓	✓
HBase Master			✓					
HBase Thrift Server		✓						
HBase RegionServer		✓	✓	✓	✓	✓	✓	
HDFS Failover Controller	✓	✓						
HDFS JournalNode	✓	✓		✓				
HDFS NameNode	✓	✓						
HDFS HttpFS				✓				
HDFS DataNode		✓	✓	✓	✓	✓	✓	✓
HDFS Balancer				✓				
Hive GateWay	✓	✓	✓	✓	✓	✓	✓	
Hive Server2		✓	✓					
Hive MetaStore Server	✓							
Hue Balancer Load			✓					
Hue Server			✓					
CM Service Alert Publisher	✓							
CM Service Event Server	✓							
CM Service Host Monitor	✓							
CM Service Navigator Audit Server	✓							
CM Service Navigator Metadata Server	✓							
CM Service Reports Manager	✓							
CM Service Service Monitor	✓							
Oozie Server	✓	✓					✓	
Spark Gateway	✓	✓	✓	✓	✓	✓	✓	
Spark History Server	✓							
Key-Value Store Indexer Lily HBase Indexer				✓				
Kafka Broker					✓	✓	✓	
Solr Server				✓	✓	✓	✓	
YARN (MR2 Included) NodeManager		✓	✓	✓	✓	✓	✓	✓
YARN (MR2 Included) JobHistory Server		✓						
YARN (MR2 Included) ResourceManager	✓							
ZooKeeper Server		✓	✓	✓				

✓= Instalado

Tabla 4.8: Roles por Nodos

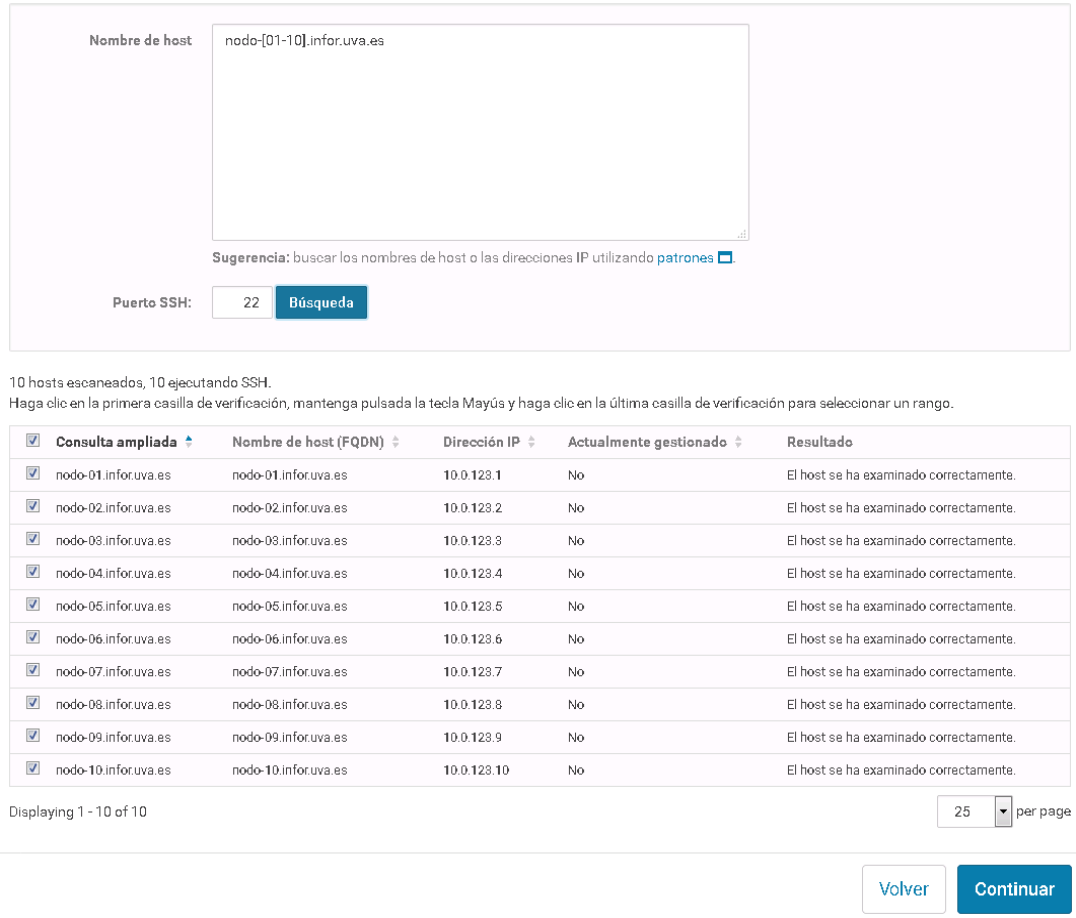

Figura 4.6: Especificar Nodos

de Datanode y Sorl. Los recursos hardware que dispone cada nodo es bien distinto, ver la Tabla 4.6, esto se debe a un estudio previo de los recursos de explotación actuales de cada nodo hardware, ver Tabla 3.1. Dado que no disponemos de memoria suficiente para todas las máquinas virtuales que hay desplegadas, lo que hacemos es sobre explotación de memoria, una técnica usada en sistema de virtualización que consiste en compartir la memoria entre todas las máquinas virtuales, mediante técnicas de deduplicación y asignación en vuelo de bloques de memoria. El tamaño de memoria del nodo es crítico, ya que Cloudera siempre estima el uso para el caso en el que todos los roles demanden la memoria que tienen asignada y establece unos umbrales para general alarmas de falta de memoria. El resultado del despliegue de roles por cada nodo lo podemos ver en Tabla 4.8. La interface de usuario para asignar cada rol a cada nodo se puede ver en la Figura 4.9

En la página de configuración de la base de datos deberemos ingresar los hosts de la base de datos, los nombres, los nombres de usuario y las contraseñas que se crearon al configurar el servidor de bases de datos según la Tabla 4.3. Por último, el asistente nos

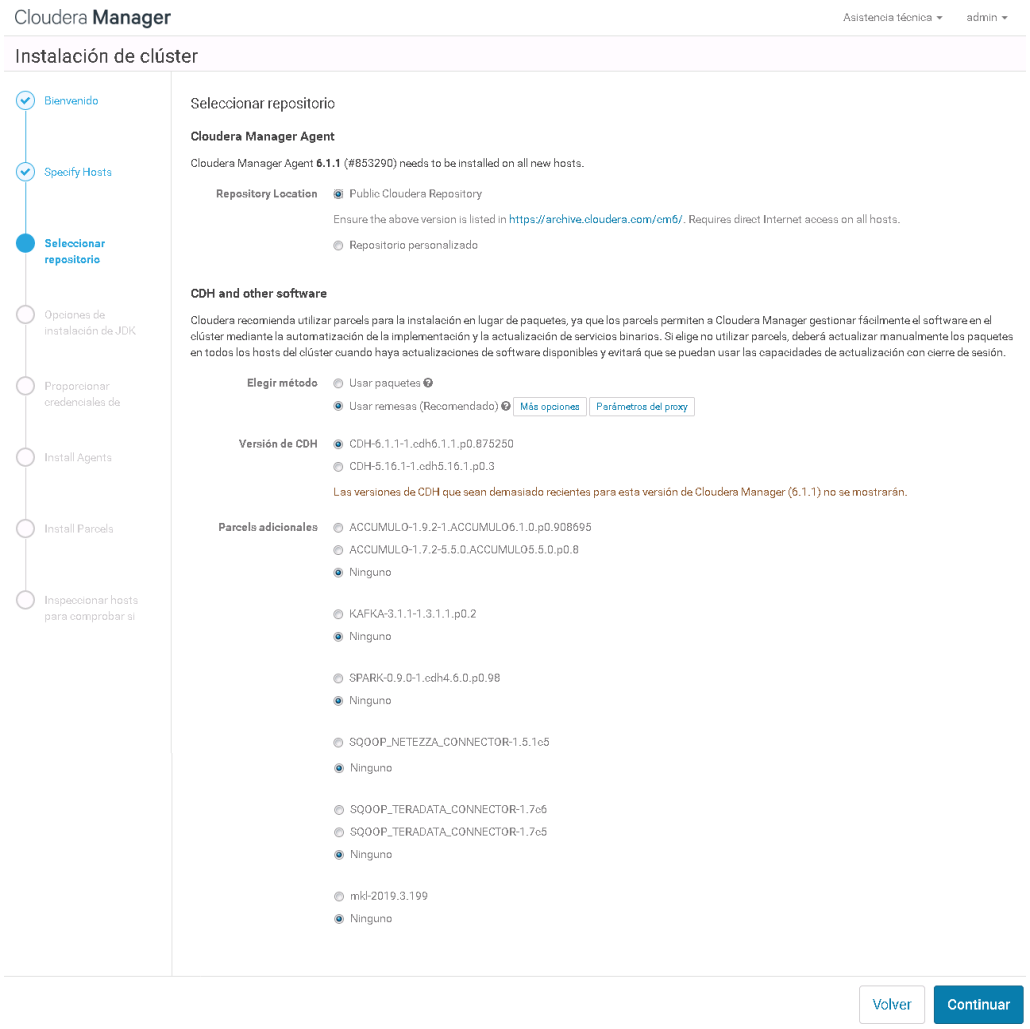

Figura 4.7: Repositorio y Parcels

mostrará la ejecución de los comandos de despliegue y un resumen de la instalación, como se muestra en la Figura 4.10

Tras el despliegue deberemos ajustar una serie de variables de configuración para que el clúster esté operativo. Una de las más importantes es el tamaño de la pila Java tanto para los NameNodes, como para los DataNodes, este lo fijamos a 4GiB. Para que el servicio de monitor funcione adecuadamente este debe tener un tamaño de pila para Java de 2GiB y la cantidad de memoria no Java máxima que necesita es de 14GiB. Otro tamaño de pila que debe ser ajustado es el del servicio de monitor de Host que lo ajustamos a 1GiB y la cantidad de memoria no Java máxima la ajustamos a 2GiB. Estos servicios residen en el nodo-03 que es uno de los que más memoria demanda.

A partir de este momento dispondremos un clúster como el que se muestra en la Figura 4.11, desde él podremos desarrollar proyectos Big Data y todas las tareas de

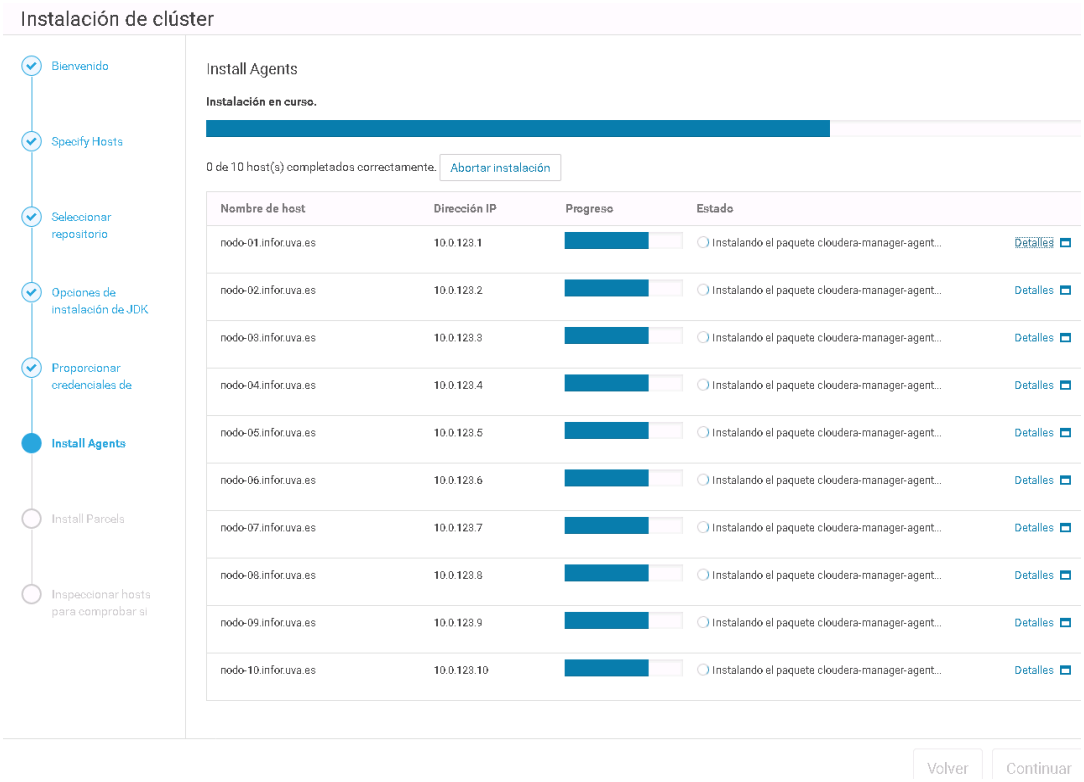

Figura 4.8: Instalación del Agente

administración y gestión del clúster.

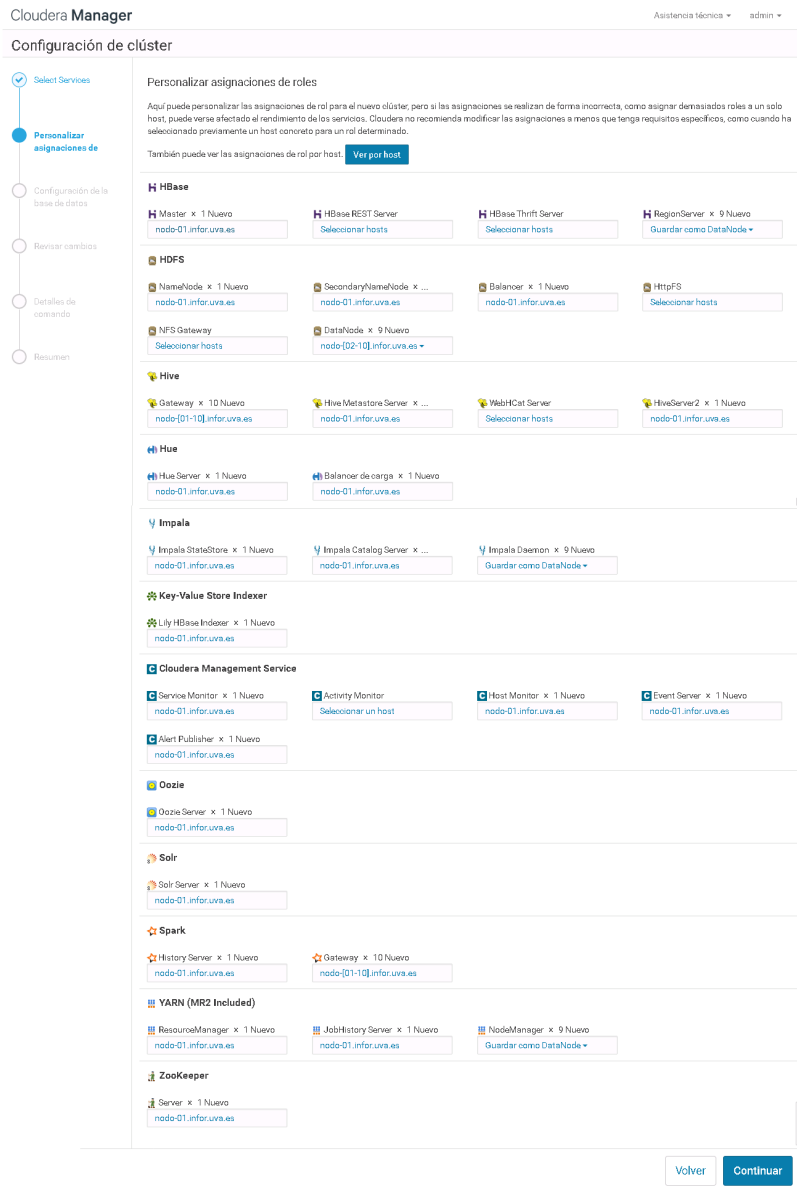

Figura 4.9: Asignación de Roles

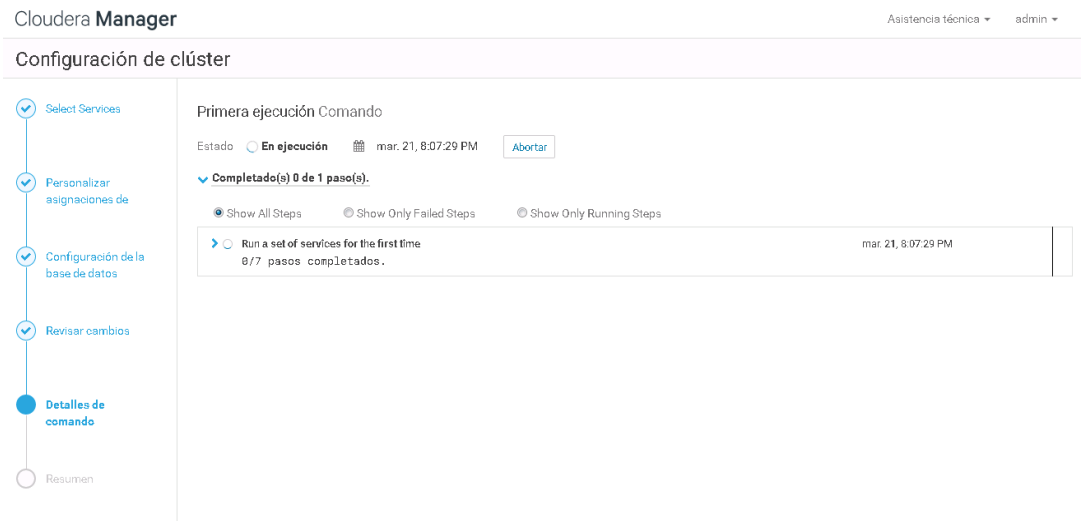

Figura 4.10: Despliegue de Roles

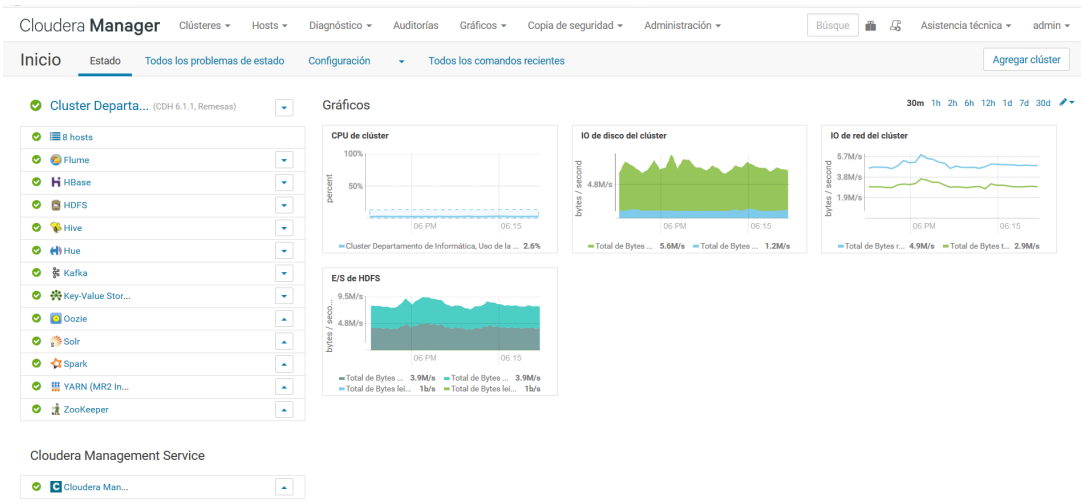

Figura 4.11: Cloudera Manager desplegado

5. Cloudera Manager

Cloudera es una plataforma escalable que permite implementar y administrar Apache Hadoop y los proyectos relacionados, manipular y analizar grandes volúmenes de datos y poderlos mantener seguros y protegidos. Cloudera proporciona varias herramientas y productos, como Cloudera Enterprise Data Hub, Cloudera Data Warehouse, Cloudera DB operacional, Cloudera Data Science & Engineering o Cloudera Essentials. Todas ellas aplicaciones de alto nivel para diferentes sectores, almacenamiento, ingeniería de datos, gestión de bases de datos, etc. Cloudera proporciona los siguientes productos CDH (Cloudera Distribution Hadoop), Apache Impala, Cloudera Search, Cloudera Manager y Cloudera Navigator. Nosotros nos centraremos en tres productos principalmente, CDH, el módulo de búsqueda Cloudera Search y Cloudera Manager.

5.1 CDH

Cloudera Distribution Hadoop, es una distribución de código abierto de Apache Hadoop proporcionada por Cloudera Inc, que es una empresa estadounidense de software empresarial con sede en Palo Alto. Como hemos visto en secciones anteriores, es una solución de Hadoop que ofrece procesamiento por lotes, SQL interactivo, búsqueda interactiva, etc. Cloudera CDH integra los siguientes componentes, en la sección 2.2.2 se puede consultar estos componentes:

- Apache Avro
- Apache Crunch

- Apache Flume
- Apache Hadoop
- Apache HBase
- Apache Hive / Hive on Spark / HCatalog
- Hue
- Apache Impala
- Apache Kafka
- Apache Kudu
- Apache Oozie
- Apache Parquet
- Apache Pig
- Cloudera Search
- Apache Sentry
- Apache Spark
- Apache Sqoop
- Apache Zookeeper

Cada una de estos componentes puede ser desplegado en un nodo del clúster o en varios. Todo ello gestionado por Cloudera Manager.

5.2 Cloudera Search

La búsqueda en Cloudera proporciona acceso casi en tiempo real (NRT) a los datos almacenados o ingeridos en Hadoop y HBase, se entiende como un módulo al igual que el CDH. La búsqueda proporciona indexación casi en tiempo real, indexación por lotes, exploración de texto completo y navegación detallada, así como una interfaz simple de texto completo que no requiere grandes conocimientos de programación o SQL. Es importante comprender cómo Cloudera Manager organiza las búsquedas.

La búsqueda está totalmente integrada en la plataforma de procesamiento de datos y utiliza el sistema de almacenamiento flexible, escalable y robusto incluido en el CDH y basado en HDFS al que le han incorporado HA (High Availability) mediante dos NameNodes. Utilizar HDFS elimina la necesidad de mover grandes conjuntos de datos a través de infraestructuras para realizar tareas. El motor de búsqueda que incorpora es Apache Solr, que incluye Apache Lucene, SolrCloud, Apache Tika y Solr Cell[28].

Este servicio de búsqueda se ejecuta como un servicio distribuido en un conjunto de servidores, y cada servidor es responsable de una parte del conjunto completo de contenido que se debe buscar. Todo el conjunto de contenido se divide en partes más pequeñas, las

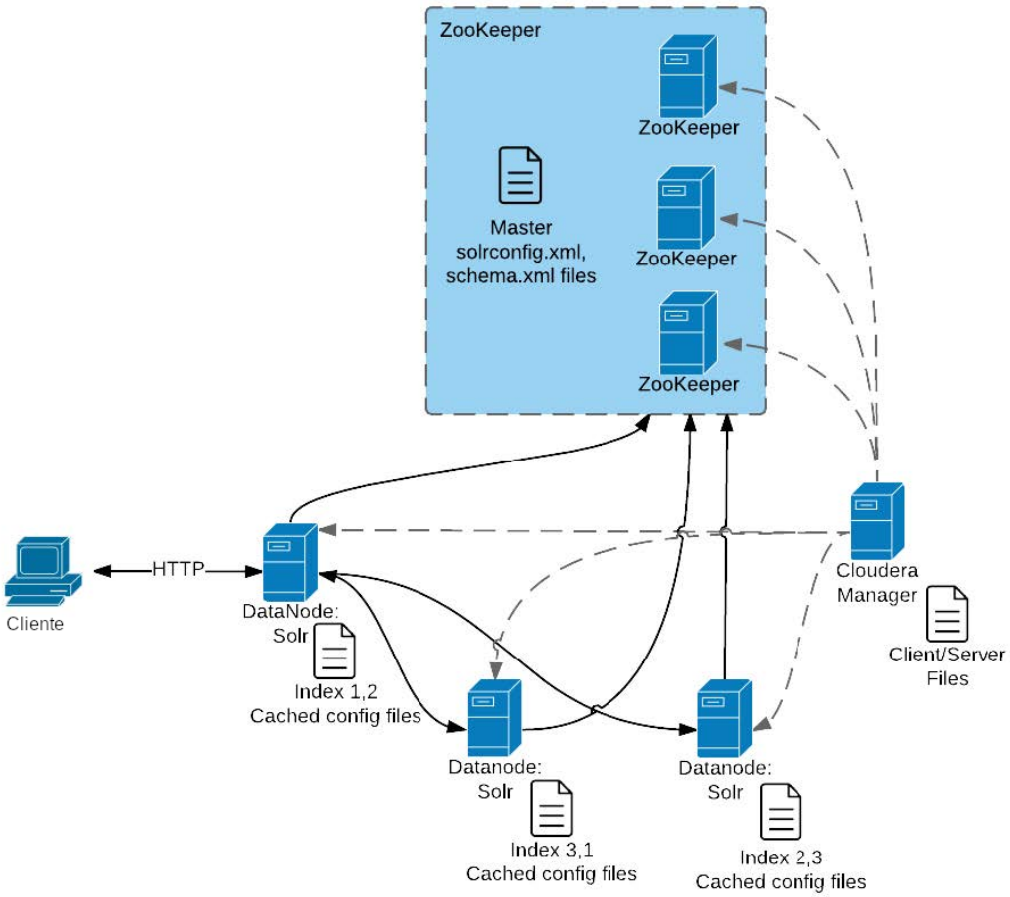

Figura 5.1: Elementos clave de Cloudera en la búsqueda [11]

réplicas se hacen de estas piezas y las piezas se distribuyen entre los servidores. Esto proporciona dos ventajas principales:

- La división del contenido en partes más pequeñas distribuye la tarea de indexar el contenido entre los servidores.
- La duplicación de las piezas del conjunto permite que las consultas se amplíen de manera más efectiva y permita que el sistema proporcione mayores niveles de disponibilidad.

Cada servidor de búsqueda de Cloudera requiere de:

- ZooKeeper en al menos un nodo del clúster. Se puede instalar ZooKeeper, Solr y HDFS en el mismo nodo.
- HDFS en al menos un nodo del clúster, pero lo normal es que esté en todos los nodos.
- Solr en al menos un nodo, pero al igual que HDFS se instala en tantos como DataNodes, esto que en todos los que tengan HDFS.

Instalar Solr y HDFS en todos los nodos y que estos sean lo más numerosos posible, hace que dispongamos de más nodos con capacidades de búsqueda lo cual implica que aumente el grado de localidad de los datos. Al tener los datos locales el rendimiento es más rápido y reducen el tráfico de red resultando un clúster más ágil.

La Figura 5.2 muestra algunos de los elementos clave en un despliegue de Cloudera. Hay que tener en cuenta que, Cloudera Manager proporciona los archivos de configuración de cliente y servidor a los nodos del clúster, y cada nodo dispone una copia caché de esta configuración. El servidor Zookeeper proporciona información sobre el estado del clúster y de todos los nodos que ejecutan Solr. Esta es la forma en que se coordinan las tareas y se recibe información del estado de las mismas. Para realizar una búsqueda el cliente debe de disponer de cierta información que variará dependiendo de lo que requiera esa consulta. Si necesita hacer sólo una consulta necesitará el nombre del servidor Solr y su puerto. Si quiere realizar acciones sobre colecciones, ya sea para agregar o eliminar, necesitará el nombre de la colección. Y si son trabajos de indexación, necesitará las direcciones de un controlador MapReduce. Todas estas configuraciones pueden ser consultadas en el Cloudera Manager que es también el encargado de proporcionarlas. Las fases para una búsqueda serán las siguientes:

1. Un cliente envía una consulta a través de HTTP.
2. La respuesta a esta consulta se la envía el NameNode y luego se pasa a un DataNode.
3. El DataNode distribuye la solicitud entre otros nodos con los fragmentos relevantes.
4. Los resultados de la consulta se recopilan y devuelven al cliente.

Una vez que comprendemos a grandes rasgos cómo se indexa la información y se busca, hay que ver la forma en la que esa información se almacena en un CDH. Para almacenar contenido en un CDH podemos usar, Flume, herramientas de copia para HDFS como puede ser *"distcp"*, Sqoop o *"fusedfs"* que es un conector al sistema de archivos HDFS.

Instalar los sistemas para la búsqueda es un trabajo que realizan los administradores del clúster. Por ejemplo, HDFS se instala para proporcionar almacenamiento; Flume o *distcp* se instalan para la ingesta de contenido. Después de que los administradores instalan estos servicios, los usuarios pueden usar herramientas de ingestión, como las utilidades de copia de archivos o los sinks de Flume.

Cuando tenemos la información almacenada debemos indexarla y es aquí cuando entra el rol del desarrollador, analista de datos. Los índices normalmente se almacenan en un

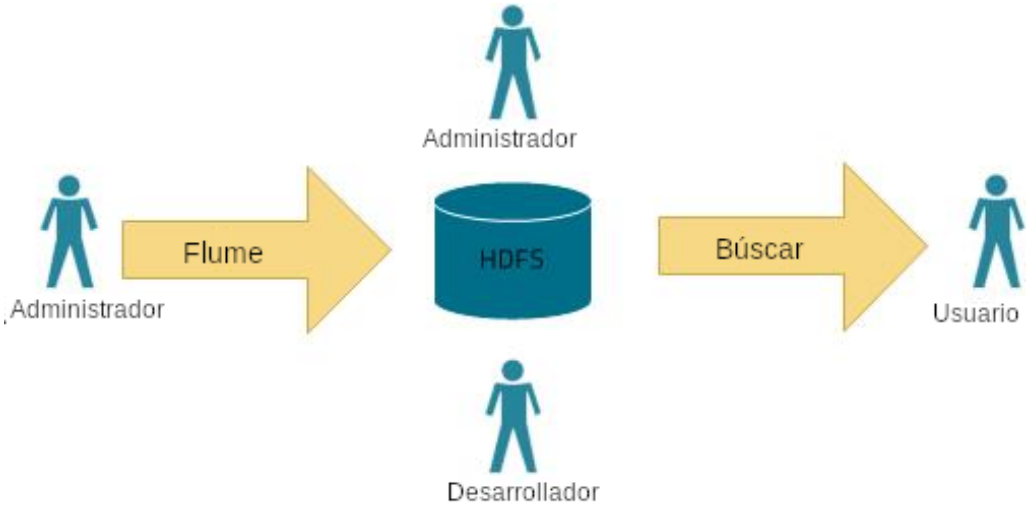

Figura 5.2: Roles en el proceso ETL

sistema de archivos local dentro de HDFS. Para poder indexar los datos se debe pasar por dos procesos:

1. ETL, extracción, transformación y carga para ello podemos usar Apache Tika o Cloudera Morfinas herramientas específicas para la extracción de contenido y metadatos, así como el mapeo de esquemas.

2. Crea índices generalmente utilizando Lucene.

Una vez que los datos están disponibles como un índice, la API de consulta proporcionada por el servicio de búsqueda permite completar consultas directas o facilitarlas mediante una herramienta de línea de comandos o una interfaz gráfica. Cloudera Manager proporciona una aplicación de interfaz de usuario simple que se puede utilizar con Hue, o se puede crear una aplicación personalizada basada en la API de Solr estándar.

5.3 Cloudera Manager

Cloudera Manager es una aplicación que se usa para implementar, administrar, monitorizar y diagnosticar problemas en los despliegues de CDH. Cloudera Manager proporciona la consola de administración, una interfaz de usuario basada en web que hace la administración de los datos más simple. Con la API de Cloudera Manager se puede utilizar para obtener métricas e información sobre el estado del clúster, así como para configurar el administrador de los CDH. Cloudera Manager ofrece características avanzadas para la gestión de clústeres. Permite un despliegue y configuración automatizados, seguimiento

de informes personalizables, y una plataforma robusta para solventar problemas de forma sencilla.

Cloudera Enterprise permite realizar muchas operaciones desde la consola, aunque no podemos disponer de todas ellas debido a la licencia Express que hemos instalado, entre ellas gestión de roles, sistema de copias de seguridad, Cloudera Navigator que es una aplicación web que permite una gestión integrada y completa de sistemas Hadoop, ofrece características para que los administradores, gestores de datos y analistas puedan securizar, gestionar y explorar los datos de Hadoop.

Figura 5.3: Pantalla Principal de Cloudera Manager

La visión general de la aplicación de gestión se muestra en la Figura 4.11. Desde esta aplicación podremos gestionar el propio Cloudera Manager, ver las configuraciones del clúster, manejar los clústeres que tengamos, configurar los nodos, los servicios, los roles, monitorizar y diagnosticar, analizar el rendimiento, manejar los recursos, activar o desactivar alta disponibilidad en servicios, realizar backups y restaturaciones ante desastres, realizar copias de seguridad de la base de datos, administrar Cloudera Navigator, acceso a almacenamiento de Amazon S3, acceso a almacenamiento de Microsoft ADLS, configuración de Google Cloud Storage y crear un centro de datos empresariales multitenant[1].

Las características que podemos explotar de Cloudera Manager son:

- Implementar y operar de forma centralizada los CDH desplegados en los nodos y otros servicios administrados, como se puede ver en la Figura 5.4. La aplicación automatiza el proceso de instalación, reduciendo el tiempo de implementación de semanas a minutos.

[1]Tenencia múltiple o multitenencia en informática corresponde a un principio de arquitectura de software en la cual una sola instancia de la aplicación se ejecuta en el servidor, pero sirviendo a múltiples clientes u organizaciones.[36]

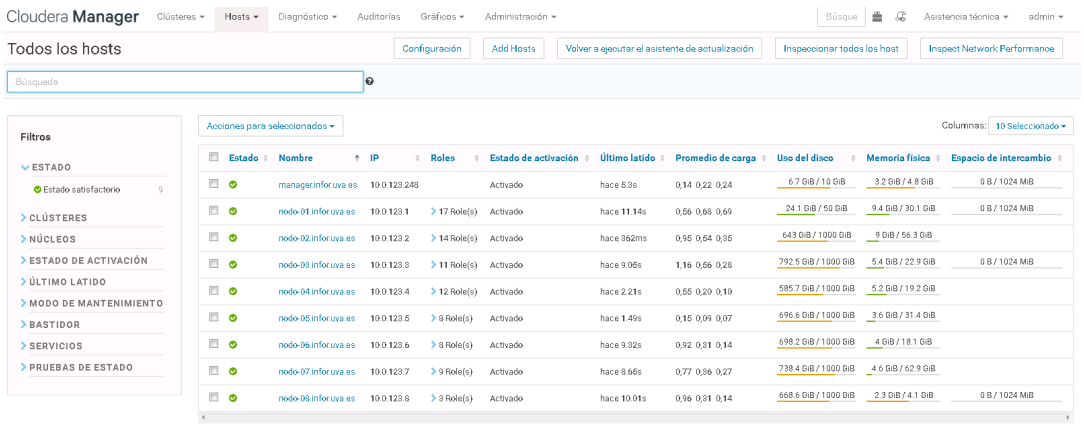

Figura 5.4: Gestión de Nodos

- Ofrece una vista en tiempo real de los nodos y servicios que se ejecutan en todo el clúster. Se pueden diseñar Dashboards personalizados, ver Figura 5.5, con una multitud de parámetros de los nodos y de los servicios que se han instalado.

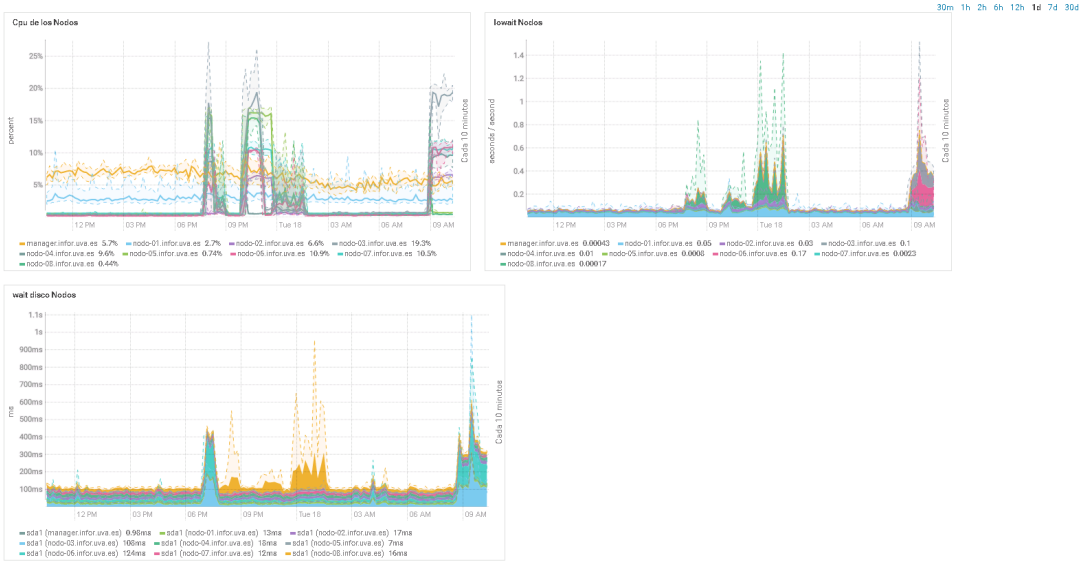

Figura 5.5: Ejemplo de DashBoard

- Proporciona una única consola central para distribuir los cambios de configuración en el clúster. En la Figura 5.6 se puede ver el buscador de configuraciones, cada una de las opciones dispone de una pequeña ayuda y guarda un historio de configuraciones para poder volver atrás en cualquier comento. Una vez realizados los cambios de configuración se pasa al despliegue que se hace desde el panel central, Figura 5.3, de la gestión del clúster.

  En todo momento en las configuraciones ofrece sugerencias como se ve en la Figu-

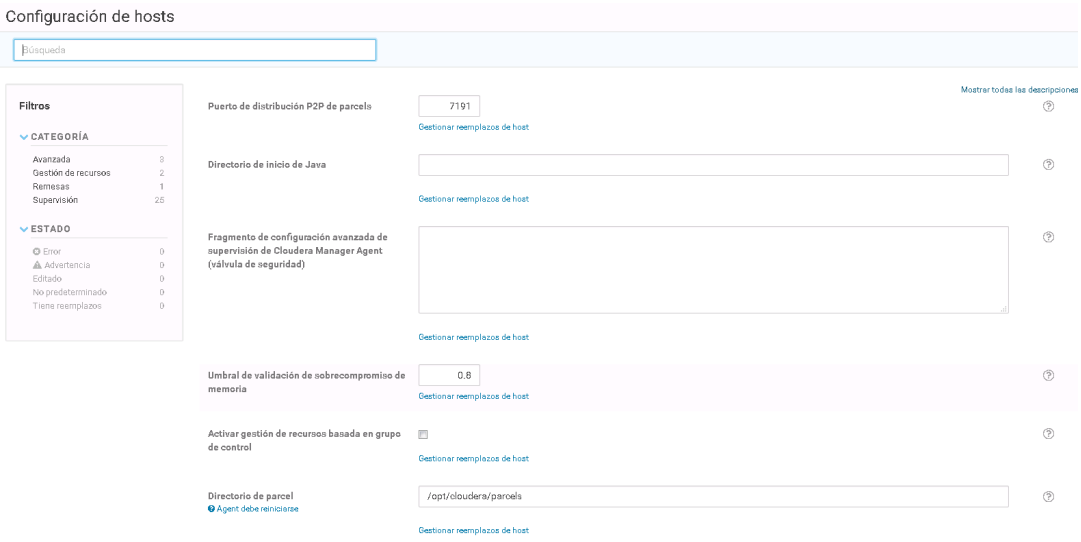

Figura 5.6: Configuraciones de nodos

ra 5.7 o cambia los colores como se ve en la Figura 5.8, indicando una configuración indebida. La interfaz de usuario está pensada para facilitar la gestión del clúster y no perder tiempo en búsqueda de información sobre qué se está haciendo. En todo momento ofrece un acceso a la documentación o la posibilidad de hacer feedbacks al foro de Cloudera sobre errores.

Figura 5.7: Inteface de Configuraciones

- Incorpora una gama completa de herramientas de diagnóstico e informes para ayudar a optimizar el rendimiento y la utilización, como se muestra en la Figura 5.9. Se pueden programar alertas por distintos parámetros, servicios, nodos y definir umbrales de las alertas para emitir avisos o realizar acciones.
- Instalador de software. Una función importante de Cloudera Manager es instalar CDH y el software de administración de servicios. Exiten dos formatos de distribución:

  - **Paquetes**. Un paquete es un formato de distribución binario que contiene código compilado y metainformación, como entendemos normalmente un

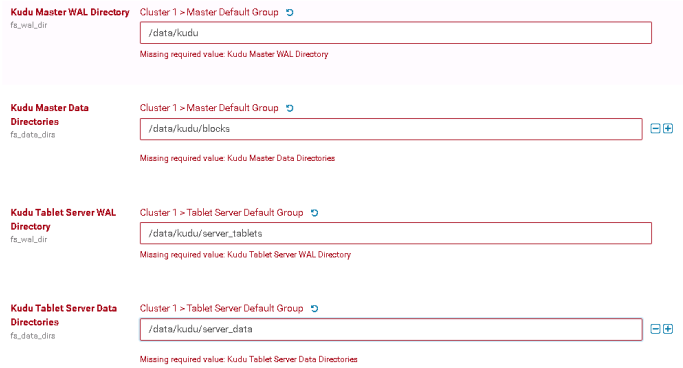

Figura 5.8: Interface de Configuración Errónea

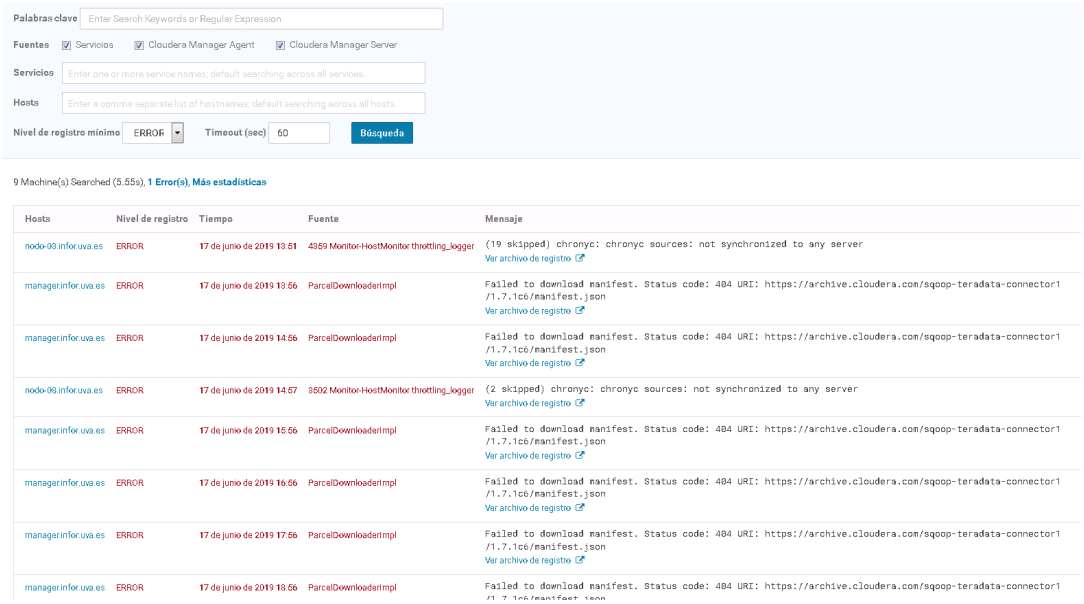

Figura 5.9: Interface de Diagnosticos

paquete, descripción, versión y dependencias.

- **Parcels**. Un Parcel es un formato de distribución binario que contiene los archivos de programa, junto con los metadatos adicionales utilizados por Cloudera Manager. Las diferencias importantes entre paquetes y paquetes son:
  - Los parcels son autónomos e instalados en un directorio versionado, lo que significa que se pueden instalar varias versiones de un parcel en paralelo. Esto nos permite designar una de estas versiones instaladas como la activa. Con los paquetes, solo se puede instalar un paquete a la vez, por lo que no hay distinción entre lo que está instalado y lo que está activo.
  - Se puede instalar parcels en cualquier ubicación en el sistema de archivos.
  - Cuando se instala desde la página de parcels, Cloudera Manager descarga automáticamente, distribuye, y activa el paquete correcto para el sistema

operativo que se ejecuta en cada nodo del clúster. Todos los nodos CDH que conforman un clúster lógico deben ejecutarse en la misma versión principal del sistema operativo que debe ser entre los que soporta Cloudera.

III Tercera Parte: Pruebas

6. Pruebas

En los temas anteriores se han estudiado y comparado cuales son las tecnologías más adecuadas para encontrar una metodología de un clúster de Hadoop, se ha seleccionado una distribución para la gestión del clúster y elegido los servicios que soportará el mismo. También se ha estudiado la distribución de roles de los nodos así como la dimensión de los mismos adecuadas al sistema que disponemos. Adicionalmente, se ha detallado un modelo de arquitectura empleando en el sistema de virtualización, el hardware disponible y para que se emplea, esto nos que sirve para ilustrar el entorno en el que se ha desarrollado el proyecto.

Ahora mostraremos cuál es el resultado final, ejecutando trabajos Big Data, de distintos tipos en el clúster. Para cualquier usuario, es fundamental que los trabajos que se ejecuten en la plataforma sean ágiles, sin perder calidad en los resultados. De esta forma, es un aspecto muy importante para este tipo de proyectos el mantener la calidad y agilidad de los procesos que se ejecuten a pesar de haber incorporado al sistema un modelo de seguridad y protección. El objetivo principal de estas pruebas es entender diferentes aspectos del clúster en términos de velocidad, volumen de trabajo, eficiencia y eficacia.

Durante el proceso de las pruebas hemos ido cambiado configuraciones de las máquinas que nos han llevado a la configuración final que explicamos en el Capítulo 3.

6.1 Conjunto de Datos

Para realizar estas pruebas necesitaremos un conjunto de datos, para ello hemos generado tres tipos de conjuntos de datos:

1. Conjunto de datos aleatorios de 50GB generados en filas de 100 Bytes cada una. El formato de cada fila es: *"clave de 10 bytes | 2 bytes separador | 32 bytes acsii / hex | 4 bytes separador | 48 bytes de relleno | 4 bytes separador | \r \n"*

2. Conjunto de datos aleatorio de tipo texto, un conjunto de datos de 10GB de texto con palabras aleatorias con el que haremos algunas pruebas.

3. Un conjunto de datos de tráfico aéreo obtenido del proveedor OpenSky de 400GB con el hemos probado Flume, Hive y estudiado cómo influyen el tipo de consultas y el tamaño de los archivos en HDFS.

A continuación, explicaremos como se obtiene los datos de OpenSky, fuente del mayor volumen de datos con el que trabajaremos.

6.1.1 Recogida de Datos de Datos OpenSky

OpenSky Network es una red receptora de datos de tráfico aéreo, que soporta la comunidad, recopila continuamente datos de vigilancia del tráfico aéreo, mantiene la información recopilada para siempre y la hace accesible a los investigadores. OpenSky Network dispone del mayor conjunto de datos de vigilancia del tráfico aéreo de este tipo. En la Tabla 6.1 se especifican los campos de datos y las características principales que se obtienen con la API de OpenSky.

Para adquirir los datos en el clúster necesitamos configurar un agente en Flume. Este agente necesita un programa que hará las funciones de canal Flume y debemos desplegarlo en todos los nodos del cluster que tengan este rol. Para ello creamos un playbook en Ansible que lanzaremos desde el nodo que hemos llamado *"orchestatrator"*. Este playbook instala el archivo, *OpenSkyFlumeSource-0.0.1-SNAPSHOT.jar*, que se encarga de recoger los mensajes de Opensky y enviarlos al sumidero Flume, el agente de Flume cuando esté lleno el sumidero lo enviará al almacenamiento HDFS del clúster.

Desde la administración de Cloudera Manager configuramos un agente Flume para recolectar los datos y qué sean públicos a todos los usuarios del sistema y estarán estructurados en un directorio por día. Es importante instalar el agente Flume en todos los DataNodes, de esta forma los datos son distribuidos en todos los DataNode. En la configuración del Flume hay una serie de parámetros a destacar:

Nombre	Descripción	Tipo	Nulo
icao24	Dirección única de 24 bits del transpondedor, ICAO. Se representa como una cadena hexadecimal.	String	✷
callsign	Callsign del avión son 8 caracteres.	String	✓
origin_country	Nombre del país obtenido de la dirección de 24 bit ICAO	String	✷
time_position	Unix timestamp en segundos de la última toma de la posición	String	✓
time_velocity	Unix timestamp en segundos de la última toma de la posición	String	✓
longitude	WGS-84 longitud en grados decimales	String	✓
latitude	WGS-84 latitud grados decimales	String	✓
altitude	Altitud barométrica o geométrica en metros	String	✓
on_ground	Indica si la posición se recuperó de un informe de posición de superficie	Boolean	✷
velocity	Velocidad sobre el suelo en in m/s	String	✓
heading	Encabezado en grados decimales en sentido horario desde el norte	String	✓
vertical_rate	Velocidad vertical en m/s. Un valor positive indica que el avión está ascendiendo; un valor negativo indica que está descendiendo	String	✓
sensors	ID de los receptores que contribuyeron a este vector de estado	String	✓
baro_altitude	Altitud barométrica en metro	String	✓
squawk	El Código del transpondedor aka Squawk.	String	✓
spi	Si el estado del vuelo indica un indicador de propósito especial	String	✓
position_source	Origen de la posición de este estado: 0 = ADS-B, 1 = ASTERIX, 2 = MLAT	String	✓

✓= Cumple la Característica
✷= No Cumple la Característica

Tabla 6.1: Diccionario de datos de la tabla de mensajes OpenSky

Configuración de Flume.

```
tier1.sources = sourceos
tier1.channels = channelos
tier1.sinks = sinkos
tier1.sources.sourceos.type = boeing.brte.opensky.OpenSkyFlumeSource
tier1.sources.sourceos.url = https://opensky-network.org/api/states/all
tier1.sources.sourceos.user = \tacha{usuario de OpenSKY}
tier1.sources.sourceos.password = \tacha{Clave del usuario de OpenSky}
tier1.sources.sourceos.channels = channelos
tier1.channels.channelos.type = file
tier1.channels.channelos.checkpointDir = /var/lib/flume-ng/file-channel/checkpoint
tier1.channels.channelos.dataDirs = /var/lib/flume-ng/file-channel/data
tier1.channels.channelos.checkpointInterval = 30000
tier1.channels.channelos.capacity = 1000000
tier1.channels.channelos.transactionCapacity = 10000
tier1.sinks.sinkos.type = hdfs
tier1.sinks.sinkos.channel = channelos
tier1.sinks.sinkos.hdfs.path = /user/opensky/fecha=%Y-%m-%d
tier1.sinks.sinkos.hdfs.filePrefix = OpenSky
tier1.sinks.sinkos.hdfs.fileSuffix = .json
tier1.sinks.sinkos.hdfs.rollSize = 1073741824
tier1.sinks.sinkos.hdfs.rollInterval = 0
tier1.sinks.sinkos.hdfs.rollCount = 0
tier1.sinks.sinkos.hdfs.idleTimeout = 1800
tier1.sinks.sinkos.hdfs.useLocalTimeStamp = true
tier1.sinks.sinkos.hdfs.fileType = DataStream
tier1.sinks.sinkos.hdfs.batchSize = 10
```

- **user**. Este será el usuario de la página de OpenSky, es necesario para poder adquirir los datos autenticarse en el servicio.
- **password**. Será la clave para el usuario de OpenSky.
- **path**. Hubicación de los archivos en HDFS. Se guardarán en: */user/opensky/fecha=yyyy-mm-dd*.
- **rollSize**. Tamaño de archivo que se graba. Hemos trabajado con dos tamaños 1GB y 256MB.

Con Flume hemos recogido unos 400Gb de datos con los que realizaremos las pruebas. Se han recogido con dos configuraciones distintas, una con archivos de 256MB y otra con archivos de 1GB. Lo que buscamos es observar la carga de trabajo y el rendimiento del clúster cuando trabaja con muchos archivos pequeños frente a archivos grandes.

6.2 Pruebas

Hadoop proporciona una librería con una serie de programas básicos y que están por lo tanto incluidas en el CDH de Cloudera. Usaremos YARN para invocar los comandos

desde cualquiera de los nodos del clúster y realizar varias pruebas con estos programas. Mediante el siguiente comando podemos ver la lista de ejemplos que proporciona esta librería.

```
PATH_JAR="/opt/cloudera/parcels/CDH/lib/hadoop-mapreduce"
yarn jar $PATH_JAR/hadoop-mapreduce-examples.jar
```

El contenido de esta librería incluye los ejemplos siguientes:

- **aggregatewordcount**. Para realizar esta prueba necesitamos un archivo de texto. Este programa de MapReduce cuenta las palabras de los archivos de entrada.
- **aggregatewordhist**. Un programa de MapReduce basado en agregación que cuenta el histograma de las palabras de los archivos de entrada.
- **bbp**. Es un programa de MapReduce que usa una fórmula Bailey-Borwein-Plouffe para calcular los dígitos exactos de π.
- **pi**. Es un programa de MapReduce que calcula π mediante el método Monte Carlo.
- **dbcount**. Es un trabajo de ejemplo que cuenta los registros de vistas de página almacenados en una base de datos.
- **grep**. Este programa de MapReduce cuenta las coincidencias de una expresión regular en los archivos de un directorio.
- **join**. Realiza una unión de conjuntos de datos ordenados con particiones equiparables.
- **multifilewc**. Cuenta las palabras de varios archivos.
- **pentomino**. Programa de MapReduce para la colocación de mosaicos con el fin de encontrar soluciones a problemas de pentomino.
- **randomtextwriter**. Programa de MapReduce que escribe 10 GB de datos de texto aleatorios por nodo. *teragen* que veremos posteriormente.
- **randomwriter**. Programa de MapReduce que escribe datos aleatorios por nodo.
- **sort**. Un programa de MapReduce que ordena los datos escritos por el test *randomwriter*.
- **sudoku**: Programa que resuelve sudokus.
- **teragen**. Genera datos para la ordenación de terabytes (terasort).
- **terasort**. Ejecuta la ordenación de terabytes (terasort).
- **teravalidate**. Comprueba los resultados de la ordenación de terabytes (terasort).
- **wordcount**. Es un programa de MapReduce que cuenta las palabras de los archivos de entrada.

- **wordmean**. Programa de MapReduce que cuenta la longitud media de las palabras de los archivos de entrada.
- **wordmedian**. Programa de MapReduce que cuenta la mediana de longitud de las palabras de los archivos de entrada.
- **wordstandarddeviation**. Programa de MapReduce que cuenta la desviación estándar de la longitud de las palabras de los archivos de entrada.

Carga del Sistema Coudera y de las Maquinas Virtuales

De estos ejemplos usaremos algunos para probar el clúster. Para obtener información del estado de las tareas lanzadas consultaremos el ResorceManager de Yarn, ver Figura 6.1. En esta UI, podemos ver los recursos y estado de todas las tareas lanzadas en el clúster. En la Figura 6.2 podemos ver con más detalle los datos que podemos obtener, los contenedores que usa cada trabajo que son el número nodos del clúster que está usando en ese instante, el número de VCORES reservados para cada trabajo, o el tanto por ciento de uso de la cola y del clúster. El número de VCORES y de la memoria asignada a cada trabajo forma parte de la configuración de YARN y está relacionado con el hardware real disponible. Esta configuración es fundamental para usar todo el hardware de forma óptima. Se pueden crear pools de cálculo por usuario y grupos pero esta funcionalidad está sólo disponible para la licencia Enterprise.

También disponemos del sistema de monitorización de Cloudera Manager que nos informa del estado global, la carga de cada nodo o podemos diseñas un Dashboards personalizados, como el que se mostró en la Figura 5.5.

Por último, podemos ver el estado de las máquinas virtuales con el sistema de monitorización del Proxmox, que nos da información del consumo de CPU, IO de los discos y tráfico de red. Con toda esa información ajustaremos parámetros del clúster si es necesario y veremos si hemos dimensionado adecuadamente las máquinas virtuales de los nodos del clúster Cloudera.

6.2.1 Cálculo de π

Este ejemplo nos vale para evaluar la carga de las CPUs calculando el número π. El método que se utiliza para calcular el número π se basa en un método estadístico, método Monte Carlo [34] que consiste en:

1. Construir el entorno de trabajo:
    - Construiremos un cuadrado de lado 4. Su área lógicamente será 16.
    - Construimos un círculo inscrito en el cuadrado, que tiene de centro, el centro del cuadrado y de radio 2. Por lo tanto su área será 4π.

Figura 6.1: Inspector de Tareas Yarn

Application Priority	StartTime	LaunchTime	FinishTime	State	FinalStatus	Running Containers	Allocated CPU VCores	Allocated Memory MB	Reserved CPU VCores	Reserved Memory MB	% of Queue	% of Cluster	Progress
0	Tue Jun 25 07:23:40 +0200 2019	Tue Jun 25 07:23:40 +0200 2019	N/A	RUNNING	UNDEFINED	1	8	1024	48	6144	9,1	9,1	
0	Tue Jun 25 07:34:55 +0200 2019	Tue Jun 25 07:36:42 +0200 2019	N/A	RUNNING	UNDEFINED	2	16	2048	8	1024	18,2	18,2	
0	Tue Jun 25 07:46:48 +0200 2019	Tue Jun 25 07:50:00 +0200 2019	N/A	RUNNING	UNDEFINED	2	16	2048	0	0	18,2	18,2	
0	Tue Jun 25 07:42:02 +0200 2019	Tue Jun 25 07:43:45 +0200 2019	N/A	RUNNING	UNDEFINED	2	16	2048	0	0	18,2	18,2	

Figura 6.2: Detalle de Tareas Yarn

- Generaremos puntos al azar dentro del cuadrado, con lo cual algunos quedarán dentro del círculo y todos caerán dentro del cuadrado.

2. Aplicar el Método Monte Carlo:

   - Contaremos el total de puntos generados.
   - Contaremos el total de puntos que cayeron dentro del círculo.
   - Realizaremos el siguiente razonamiento:

$$\frac{Area-del-Circulo=2^2\pi}{Area-del-Cuadrado=4*4}=\frac{Numero-de-puntos-dentro-del-circulo}{Numero-de-puntos-totales}$$

   Despejando nos queda:

$$\pi=\frac{4*4*Numero-de-puntos-dentro-del-circulo}{2^2*Numero-de-puntos-totales}$$

Para lanzar este test hemos usado el siguiente comando:

```
PATH_JAR="/opt/cloudera/parcels/CDH/lib/hadoop-mapreduce"
sudo -u hdfs  yarn jar $PATH_JAR/hadoop-mapreduce-examples.jar  pi 1000 300000000
```

El comando lo ejecutamos como el usuario *hdfs* ya que necesitamos escribir archivos de gestión de la tarea que lanzamos. Los parámetros son:

- Número de maps.
- Número de puntos aleatorios por map.

El resultado se ha obtenido en 32 minutos 15 segundos dando como aproximación al número π, 3.141592650466666666667. Conseguir el noveno dígito del número π con este método implica multiplicar por más de 10 el número de puntos por mapper y esto hace que el tiempo se multiplique aproximadamente por 8. Para tener una referencia de la aproximación obtenida, estos son los 20 primeros decimales del número π, 3.14159265358979323846.

En este test, nos hemos centrado en el consumo de las CPUs en los nodos. Con la prueba que hemos lanzado observamos que los nodos están sobredimensionados, el número de cores que tiene asignados los nodos respecto al ancho de banda que dispone el clúster está descompensado. Esta prueba deja patente este problema, ya que a pesar de lanzar 1000 maps y 300 millones de puntos por map no hemos conseguido pasar de un 20 % de uso de CPU en el nodo más sobrecargado. En la Figura 6.3 podemos ver el uso de CPU de uno de los nodos. Otra característica que nos ha mostrado esta prueba son las diferencias entre nodos, dado que disponen de CPUs muy distintas y el reparto de VCORES y memoria de YARN lo hace de forma equitativa (ver Figura 6.4) hay nodos que terminan mucho antes que otros.

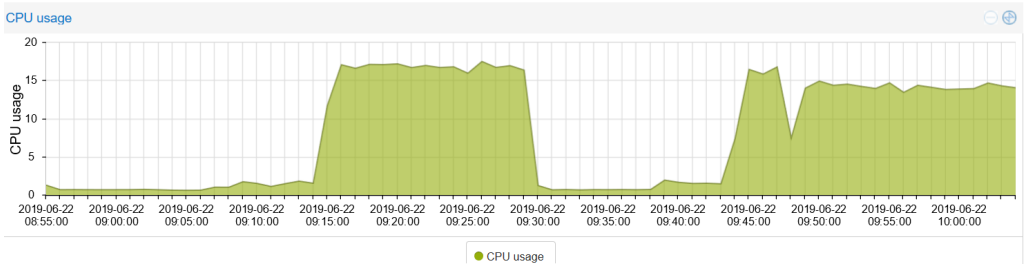

Figura 6.3: Carga de CPU de un Nodo

Total Outstanding Resource Requests: <memory:541696, vCores:4232>

Show 20 entries Search

Priority	ResourceName	Capability	NumContainers	RelaxLocality	NodeLabelExpression
20	nodo-07.infor.uva.es	<memory:1024, vCores:8>	240	true	
20	nodo-08.infor.uva.es	<memory:1024, vCores:8>	218	true	
20	nodo-03.infor.uva.es	<memory:1024, vCores:8>	251	true	
20	/default	<memory:1024, vCores:8>	529	true	
20	nodo-04.infor.uva.es	<memory:1024, vCores:8>	226	true	
20	*	<memory:1024, vCores:8>	529	true	
20	nodo-05.infor.uva.es	<memory:1024, vCores:8>	219	true	
20	nodo-02.infor.uva.es	<memory:1024, vCores:8>	215	true	
20	nodo-06.infor.uva.es	<memory:1024, vCores:8>	218	true	

Showing 1 to 9 of 9 entries First Previous 1 Next Last

Figura 6.4: Reparto de Carga de Yarn (cálculo de π)

6.2.2 Test multifilewc

Este ejemplo cuenta las palabras de todos los ficheros existentes en un directorio. Usaremos parte del repositorio de datos de OpenSky adquirido con Flume. La intención con esta prueba es observar la carga de entrada salida en HDFS y por lo tanto la carga que se produce en la entrada salida de la máquina virtual. Usaremos un formato de fichero JSON, e intentaremos contar las palabras que detecta con este programa. Ejecutamos el siguiente:

```
PATH_JAR="/opt/cloudera/parcels/CDH/lib/hadoop-mapreduce"
sudo -u hdfs  yarn jar $PATH_JAR/hadoop-mapreduce-examples.jar multifilewc /user/
    opensky/fecha=2019-05-02 /user/jiramos/multifilewc
```

El comando lo ejecutamos como el usuario *hdfs* ya que necesitamos escribir el resultado en el sistema de archivos. Los parámetros son:

- Directorio con los archivos de entrada.
- Directorio de salida para escribir el resultado.

En este caso ha consumido 2 horas 10 minutos y 53 segundos debido a que la muestra tiene un gran número de palabras distintas encontradas. El resultado ocupa 13 archivos de 1Gb cada uno donde se muestra la palabra y el número de veces que se han encontrado. Este test no ha sido útil ya que ha lanzado un único map y 12 reduces, sólo ha trabajado en un nodo. Hemos cambiado el repositorio de datos y usado el conjunto de datos de texto aleatorio y ocurre lo mismo Yarn usa un único para planificar la tarea. El problema reside en la implementación del programa. Entendemos que si Yarn no es capaz de repartir la carga entre los nodos podría ser porque los datos están alojados en un único nodo, cuestión que no ocurre, o que no es capaz de paralelizar la tarea por como está programada.

```
HDFS: Number of bytes read=       91.843.517.566 (8,5 TB)
HDFS: Number of bytes written=    13.180.229.469 (146 GB)
HDFS: Number of read operations=           1.085
HDFS: Number of write operations=             24
```

6.2.3 Test wordcount

En este caso usaremos otra implementación para contar palabras. El conjunto de datos que usaremos será de 10GB de palabras aleatorias. Lo que esperamos de esta prueba es observar cómo se reparte la carga entre los nodos del clúster Cloudera y ver cómo las máquinas virtuales gestionan la entrada salida del disco. Se espera que la carga sea mayormente de lectura en el HDFS. Para ello ejecutaremos el siguiente comando:

```
PATH_JAR="/opt/cloudera/parcels/CDH/lib/hadoop-mapreduce"
sudo -u hdfs  yarn jar $PATH_JAR/hadoop-mapreduce-examples.jar  wordcount /user/
    jiramos/random10GText /user/jiramos/wc
```

Con Yarn podemos ver que el planificador ha creado 630 Mappers y 12 Reducers (ver Figura 6.5). Esta prueba ha tardado 1 hora 32 minutos y 41 segundos. En la Figura 6.6, observamos que la carga de entrada Disco IO es de lectura en todos los nodos y cuando esta acaba en las operaciones de Reduce se produce tráfico de red.

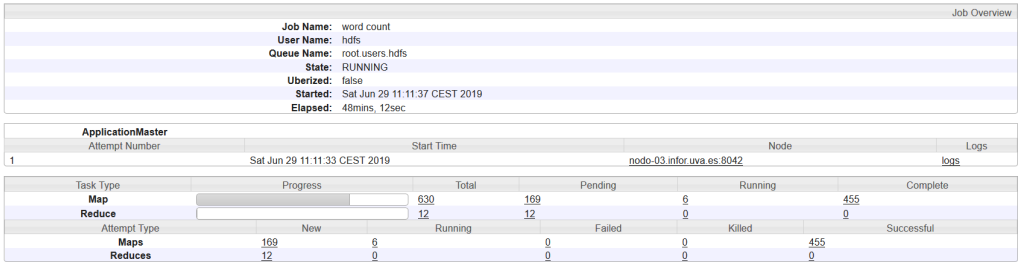

Figura 6.5: Reparto de tareas Yarn

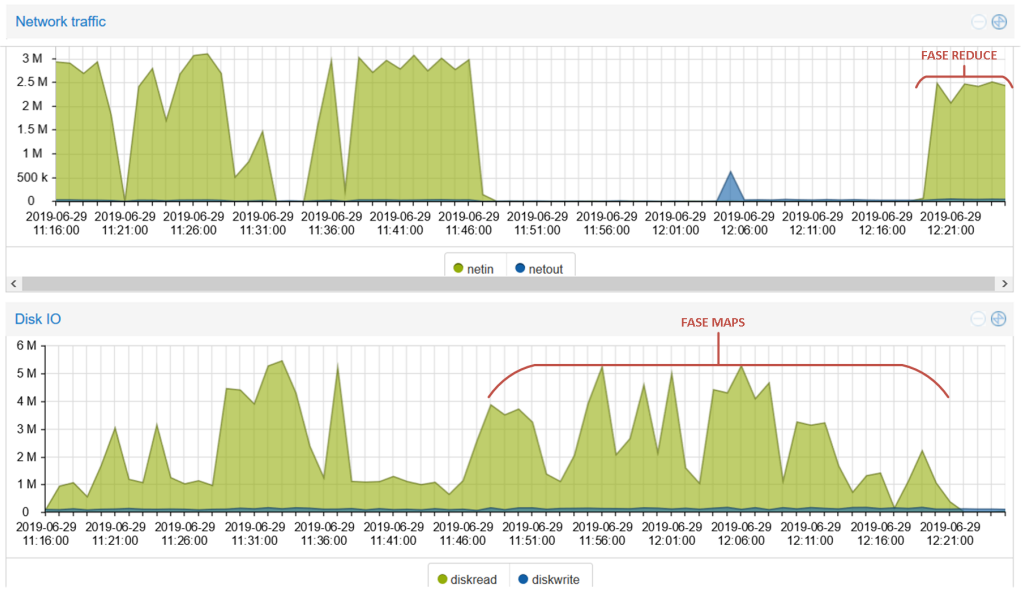

Figura 6.6: Trafico de Red y IO de la Prueba Wordcount

Los datos que nos devuelve son un listado de palabras y el número de veces que aparecen, almacenados en 12 archivos de 50MB, que se corresponde con los aproximadamente 600 MB que nos dan las estadísticas de lecturas y escrituras en el sistema HDFS que se muestran a continuación. Hay que reseñar que esta sencilla tarea de contar palabras ha provocado 70GB de lectura aproximadamente en un conjunto de datos de 10GB, esto nos da una idea de la carga que supone a los discos una instalación de Hadoop.

```
FILE: Number of bits read=      2.641.691.762   (2GB)
FILE: Number of bits written=   4.007.661.262   (4GB)
HDFS: Number of bits read=     76.577.693.824  (70GB)
HDFS: Number of bits written=     628.222.601 (600MB)
HDFS: Number of read operations=        1.950
HDFS: Number of write operations=          24
```

6.2.4 Test grep

Este ejemplo cuenta las coincidencias de una expresión regular en los archivos de un directorio. Disponemos de un conjunto de datos con mensajes de aviones, ver Tabla 6.1. Utilizaremos este programa para saber cuantos mensajes ha enviado un avión en un día. Para ello ejecutamos el siguiente:

```
PATH_JAR="/opt/cloudera/parcels/CDH/lib/hadoop-mapreduce"
sudo -u hdfs  yarn jar $PATH_JAR/hadoop-mapreduce-examples.jar  grep /user/opensky
    /fecha=2019-05-02 /user/jiramos/grep SKW3743
```

El comando lo ejecutamos como el usuario *hdfs* ya que necesitamos escribir el resultado en el sistema de archivos. Los parámetros son:

- Directorio con los archivos de entrada.
- Directorio de salida para escribir el resultado.
- Expresión regular.

La idea de esta prueba es que aprovechando los datos que tenemos en formato JSON con los mensajes emitidos en un días por los aviones, realizar una búsqueda para contar cuantas veces aparece el *icao24* de un avión, que es el identificador único que emiten en los mensajes. De esta forma sabremos cuantos mensajes ha emitido en ese día. Hemos elegido uno al azar, el *SKW3743*, cómo queremos que sea una búsqueda exacta no necesitamos una expresión regular.

Para realizar búsqueda y cuenta usamos un conjunto de datos que 86GB, distribuidos en 344 archivos de 256MB, cada uno ha tardado 1 hora y 11 minutos y 58 segundos. Ha lanzado 683 maps y 12 reducers en 6 nodos. El resultado es de 9854 mensajes emitidos en un día, por ese avión. Al tener que crear tantos maps, el sistema genera muchas operaciones de entrada salida en los discos, con lo cual las CPUs de los nodos no se ven saturadas alcanzando una media de 5 % de carga. Los datos de entrada salida son fundamentalmente de lectura como se ve en la Figura 6.7. Los siguientes son los contadores totales del sistema de archivos HDFS:

```
HDFS: Number of bits read=     91.575.484.075 (90GB)
HDFS: Number of bits written=          1.056
HDFS: Number of read operations=       2.109
HDFS: Number of write operations=         24
```

En este caso se han leído aproximadamente 90GB que se corresponde con el total que ocupan los ficheros, es una diferencia notable con la prueba anterior.

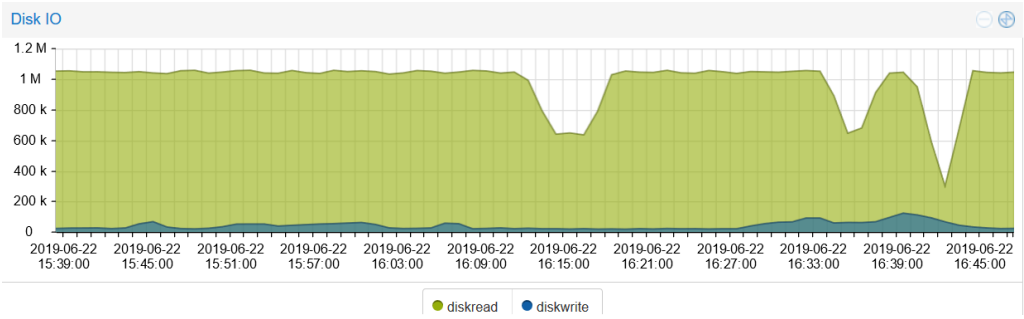

Figura 6.7: Carga de los discos de un Nodo

Este test lo repetimos para ver si el sistema responde de una forma lineal con otro conjunto de datos. Buscaremos una palabra, seleccionada aleatoriamente, del conjunto de datos de 10GB de texto. Para ello ejecutamos la siguiente tarea:

```
PATH_JAR="/opt/cloudera/parcels/CDH/lib/hadoop-mapreduce"
sudo -u hdfs  yarn jar $PATH_JAR/hadoop-mapreduce-examples.jar  grep /user/jiramos
    /random10GText /user/jiramos/greptext JIexcerpt
```

En este caso la tarea ha tardado 1 hora, 1 minuto y 38 segundos, ha encontrado la palabra 2054 veces, ha usasdo 630 mappers y 22 reduces. Las lecturas y escrituras del sistema HDFS son:

```
FILE: Number of bits read=             2.511
FILE: Number of bits written=     11.398.608 (11MB)
HDFS: Number of bits read=     50.000.006.200 (50GB)
HDFS: Number of bits written=             25
HDFS: Number of read operations=         155
HDFS: Number of write operations=          2
```

6.2.5 Test teragen terasort teravalidate

En este test generaremos 50GB de información, los ordenaremos y validaremos el resultado. Para generar los archivos con los que trabajaremos ejecutamos el siguiente comando:

```
PATH_JAR="/opt/cloudera/parcels/CDH/lib/hadoop-mapreduce"
sudo -u hdfs yarn jar $PATH_JAR/hadoop-mapreduce-examples.jar teragen -Dmapred.map
    .tasks=50 500000000 /user/jiramos/50GB
```

El comando lo ejecutamos como el usuario *hdfs* ya que necesitamos escribir archivos de gestión y de salida de la tarea que lanzamos. Los parámetros son:

- -Dmapred.map.tasks indica a Hadoop cuántas tareas de asignación se usarán para el trabajo, hemos fijado 50 para que los nodos tengan al menos 7 tareas por nodo.
- Número de Bytes a escribir, 50 GB.
- Directorio de salida para escribir el resultado.

Esta prueba crea 50GB de información distribuida en 59 archivos de unos 954MB cada uno. El proceso ha tardado 2 horas, 18 minutos y 48 segundos.

Una vez tenemos estos datos aleatorios generados podemos ordenarlos, esto lo hacemos usando el test *terasort*, como se muestra a continuación:

```
PATH_JAR="/opt/cloudera/parcels/CDH/lib/hadoop-mapreduce"
sudo -u hdfs yarn jar $PATH_JAR/hadoop-mapreduce-examples.jar terasort -Dmapred.
    map.tasks=50 -Dmapred.reduce.tasks=50 /user/jiramos/50GB /user/jiramos/50GB.
    ordenado
```

- *mapred.map.tasks* indica a Hadoop cuántas tareas de asignación se usarán para el trabajo, hemos fijado 50 para que los nodos tengan al menos 7 tareas por nodo.
- Directorio de entrada.
- Directorio de salida para escribir el resultado.

La carga de este sistema es equilibrada en lectura/escritura como se muestra en la Figura 6.8. El tiempo que ha tardado en obtener la salida ordenada ha sido de 1 hora 35 minutos y 53 segundos. Menos que en crear los 50GB. La razón se debe a que el número de ficheros creados es de 50 archivos de casi 1GB cada uno, esto hace que las operaciones de escritura sean más eficientes al ser de un menor número de archivos y más grandes. De este test concluimos que HDFS penaliza mucho las escrituras de muchos ficheros pequeños. Para realizar esta tarea YARN ha empleado 471 maps y 71 reducers usando 6 nodos del clúster.

```
FILE: Number of bytes read=     22.369.381.474 (22GB)
FILE: Number of bytes written= 44.323.343.334 (42GB)
HDFS: Number of bytes read=     50.000.046.000 (50GB)
HDFS: Number of bytes written= 50.000.000.000 (50GB)
HDFS: Number of read operations=          1.450
HDFS: Number of write operations=           100
```

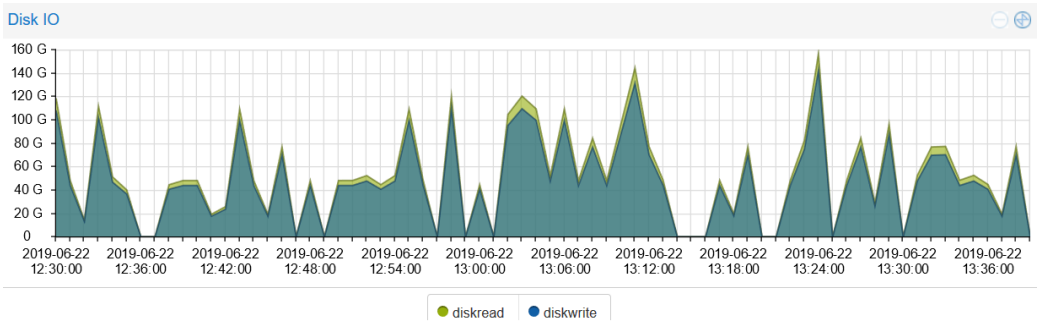

Figura 6.8: Carga de los discos de un Nodo

Ahora pasaremos el test de validación, para ello ejecutamos el siguiente comando:

```
PATH_JAR="/opt/cloudera/parcels/CDH/lib/hadoop-mapreduce"
sudo -u hdfs yarn jar $PATH_JAR/hadoop-mapreduce-examples.jar teravalidate -
    Dmapred.map.tasks=50 -Dmapred.reduce.tasks=50 /user/jiramos/50GB.ordenado /
    user/jiramos/50GB.validado
```

Como era de esperar, el peso de esta operación ha sido de lectura. El tiempo empleado ha sido de 3 minutos y 51 segundos. Para realizar este test ha empleado solo un nodo, y la carga ha sido de lectura en el sistema de archivos, se muestra a continuación:

```
FILE: Number of bytes read=           1.299
FILE: Number of bytes written=    5.720.956 (5,7MB)
HDFS: Number of bytes read= 50.000.003.100 (50GB)
HDFS: Number of bytes written=           25
HDFS: Number of read operations=         80
HDFS: Number of write operations=         2
```

6.3 Prueba Hive

En esta sección vamos a realizar una serie de pruebas con Hive usando los datos de Opensky que adquirimos con Flume. Estas pruebas realizarán una transformación unos 400GB de datos.

Crear Bases de Datos

Para tratar los datos crearemos una base de datos Hive con los datos que disponemos. Las instrucciones son las siguientes:

```
SET PATH_JAR='/opt/cloudera/parcels/CDH/lib/hive-hcatalog/share/hcatalog/'
ADD JAR ${PATH_JAR}/hive-hcatalog-core.jar;

CREATE EXTERNAL TABLE DL0 (time BIGINT, states ARRAY<ARRAY<STRING>>)
  PARTITIONED BY (fecha STRING)
  ROW FORMAT SERDE 'org.apache.hive.hcatalog.data.JsonSerDe'
  STORED AS TEXTFILE LOCATION '/user/opensky';

ALTER TABLE DL0
    ADD PARTITION (fecha = '2019-04-30') LOCATION '/user/opensky/fecha=2019-04-30'
    ;
ALTER TABLE DL0
    ADD PARTITION (fecha = '2019-05-01') LOCATION '/user/opensky/fecha=2019-05-01'
    ;
ALTER TABLE DL0
    ADD PARTITION (fecha = '2019-05-02') LOCATION '/user/opensky/fecha=2019-05-02'
    ;
ALTER TABLE DL0
    ADD PARTITION (fecha = '2019-05-03') LOCATION '/user/opensky/fecha=2019-05-03'
    ;
ALTER TABLE DL0
    ADD PARTITION (fecha = '2019-05-05') LOCATION '/user/opensky/fecha=2019-05-05'
    ;
ALTER TABLE DL0
    ADD PARTITION (fecha = '2019-05-06') LOCATION '/user/opensky/fecha=2019-05-06'
    ;
ALTER TABLE DL0
    ADD PARTITION (fecha = '2019-05-07') LOCATION '/user/opensky/fecha=2019-05-07'
    ;
ALTER TABLE DL0
    ADD PARTITION (fecha = '2019-05-08') LOCATION '/user/opensky/fecha=2019-05-08'
    ;
ALTER TABLE DL0
    ADD PARTITION (fecha = '2019-06-07') LOCATION '/user/opensky/fecha=2019-06-07'
    ;
ALTER TABLE DL0
    ADD PARTITION (fecha = '2019-06-09') LOCATION '/user/opensky/fecha=2019-06-09'
    ;
ALTER TABLE DL0
```

```
    ADD PARTITION (fecha = '2019-06-10') LOCATION '/user/opensky/fecha=2019-06-10'
    ;
```

Con las instrucciones anteriores hemos realizado tres operaciones:

- Cargar la librería para tratar JSON, ya que los datos adquiridos de Opensky están en ese formato, con los campos descritos en la Tabla 6.1.
- Crear la base de datos externa, indicando donde se almacenará y el formato de las columnas.
- Crear las particiones de la tabla, dado que tenemos 11 días adquiridos hacemos una partición por día.

Crear la zona de transición, en la que almacenaremos los datos transformados, consiste en tres procesos:

- Creamos la base de datos DL1, con los campos que se corresponden con los campos del JSON que adquirimos con Flume. El formato de esta base de datos es como un CSV separado por comas.
- Creamos las particiones, una por cada día que tenemos adquirido.
- Creamos índices para que las búsquedas en esta base de datos sean más rápidas.

En el siguiente código podemos ver las instrucciones necesarias:

```
CREATE EXTERNAL TABLE DL1 (icao24 STRING,
                           callsign STRING,
                           ocoun STRING,
                           time_position STRING,
                           time_velocity STRING,
                           lat STRING,
                           lon STRING,
                           altitude STRING,
                           on_ground STRING,
                           speed STRING,
                           track STRING,
                           vrate STRING,
                           sensors STRING,
                           baltitude STRING,
                           quawk STRING,
                           spi STRING,
                           pos_source STRING)
PARTITIONED BY (fecha STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n'
STORED AS TEXTFILE LOCATION '/user/jiramos/DL1';

ALTER TABLE DL1 ADD PARTITION (fecha = '2019-04-30') LOCATION '/user/jiramos/DL1
    /2019-04-30';
```

```
ALTER TABLE DL1 ADD PARTITION (fecha = '2019-05-01') LOCATION '/user/jiramos/DL1
    /2019-05-01';
ALTER TABLE DL1 ADD PARTITION (fecha = '2019-05-02') LOCATION '/user/jiramos/DL1
    /2019-05-02';
ALTER TABLE DL1 ADD PARTITION (fecha = '2019-05-03') LOCATION '/user/jiramos/DL1
    /2019-05-03';
ALTER TABLE DL1 ADD PARTITION (fecha = '2019-05-05') LOCATION '/user/jiramos/DL1
    /2019-05-05';
ALTER TABLE DL1 ADD PARTITION (fecha = '2019-05-06') LOCATION '/user/jiramos/DL1
    /2019-05-06';
ALTER TABLE DL1 ADD PARTITION (fecha = '2019-05-07') LOCATION '/user/jiramos/DL1
    /2019-05-07';
ALTER TABLE DL1 ADD PARTITION (fecha = '2019-05-08') LOCATION '/user/jiramos/DL1
    /2019-05-08';
ALTER TABLE DL1 ADD PARTITION (fecha = '2019-06-07') LOCATION '/user/jiramos/DL1
    /2019-06-07';
ALTER TABLE DL1 ADD PARTITION (fecha = '2019-06-09') LOCATION '/user/jiramos/DL1
    /2019-06-09';
ALTER TABLE DL1 ADD PARTITION (fecha = '2019-06-10') LOCATION '/user/jiramos/DL1
    /2019-06-10';

CREATE INDEX indx_callsign ON TABLE DL1(callsign) AS 'org.apache.hadoop.hive.ql.
    index.compact.CompactIndexHandler' WITH DEFERRED REBUILD ;
CREATE INDEX indx_icao24 ON TABLE DL1(icao24) AS 'org.apache.hadoop.hive.ql.index.
    compact.CompactIndexHandler' WITH DEFERRED REBUILD ;
```

Transformación de datos

Para limpiar los datos insertamos los datos de DL0 a la base de datos DL1 con la siguiente instrucción:

Esta consulta es representativa para un día.

```
INSERT OVERWRITE TABLE DL1 PARTITION (fecha = '2019-05-07')
SELECT upper(trim(split(STATE[0],'[\,]')[0])) AS icao24,
       upper(trim(split(STATE[1],'[\,]')[0])) AS callsign,
       upper(trim(split(STATE[2],'[\,]')[0])) AS ocoun,
       upper(trim(split(STATE[3],'[\,]')[0])) AS time_position,
       upper(trim(split(STATE[4],'[\,]')[0])) AS time_velocity,
       upper(trim(split(STATE[5],'[\,]')[0])) AS lat,
       upper(trim(split(STATE[6],'[\,]')[0])) AS lon,
       upper(trim(split(STATE[7],'[\,]')[0])) AS altitude,
       upper(trim(split(STATE[8],'[\,]')[0])) AS on_ground,
       upper(trim(split(STATE[9],'[\,]')[0])) AS speed,
       upper(trim(split(STATE[10],'[\,]')[0])) AS track,
       upper(trim(split(STATE[11],'[\,]')[0])) AS vrate,
       upper(trim(split(STATE[12],'[\,]')[0])) AS sensors,
       upper(trim(split(STATE[13],'[\,]')[0])) AS baltitude,
       upper(trim(split(STATE[14],'[\,]')[0])) AS squawk,
       upper(trim(split(STATE[15],'[\,]')[0])) AS spi,
       upper(trim(split(STATE[16],'[\,]')[0])) AS pos_source
FROM
  (SELECT explode(DL0.states) AS STATE
```

```
    FROM DL0
    WHERE fecha ='2019-05-07') AS STATE
WHERE upper(trim(split(STATE[1],'[\,]')[0]))!=''
  SORT BY icao24,
          callsign;
```

Cada día tarda unas 5 horas en crearlo. Esto es excesivo y observamos que el problema reside en la configuración de bloque del servicio HDFS que es muy pequeño y lo aumentamos a 1GB. Otro problema que observamos es el número tan grande de archivos que tenemos de entrada. Esto provoca que el número de mapers sea muy grande, 340 y 86 reducers, este problema lo arreglamos fijando la siguiente configuración antes de volver a ejecutar la instrucción:

```
SET mapreduce.input.fileinputformat.split.maxsize=10000000000;
SET mapreduce.input.fileinputformat.split.minsize=10000000000;
```

Con esta instrucción lo que hacemos es forzar a que los mapreducers sean de 1GB. Para evitar este problema cambiamos la configuración del servicio Flume para que creara archivos de 1GB en vez de los de 256MB con los que adquirimos los primeros días.

Con estos cambios y ajustes conseguimos reducir el tiempo a 1 hora y 38 minutos por instrucción obteniendo como resultado una base de datos de unos 86 archivos de unos 900MB de media cada uno. Un ejemplo del resultado obtenido lo podemos ver a continuación:

```
00412D,AZW462,ZIMBABWE,1557230804,1557230804,28.2939,-25.4919,8183.88,FALSE
    ,230.48,10.16,12.35,\N,8176.26,7017,FALSE,0
00412D,AZW462,ZIMBABWE,1557230665,1557230665,28.2388,-25.7571,6012.18,FALSE
    ,200.3,10.36,19.18,\N,6004.56,2614,FALSE,0
00412D,AZW462,ZIMBABWE,1557230799,1557230799,28.2918,-25.5022,8115.3,FALSE
    ,229.65,10.45,13.66,\N,8092.44,2614,FALSE,0
```

Tras estos ajustes nos dimos cuenta que la librería de tratamiento del JSON de entrada no era suficientemente eficiente, y la cambiamos por otra que redujo más el tiempo, unos 12 minutos de media por día. También cambiamos la instrucción para crear la base de datos DL1. A continuación, mostramos las instrucciones para usar la nueva librería, que interpreta los archivos JSON, la creación de la base de datos y un ejemplo de consulta.

```
ADD JAR hdfs:///user/jiramos/json-serde-1.3.8-jar-with-dependencies.jar;
```

```
CREATE EXTERNAL TABLE DL1_B ( time BIGINT, states ARRAY<ARRAY<STRING>> )
PARTITIONED BY (`fecha` STRING)
ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe' WITH SERDEPROPERTIES ("
    ignore.malformed.json" = "true")
STORED AS TEXTFILE LOCATION '/user/opensky';

ALTER TABLE DL1_B ADD PARTITION (fecha = '2019-04-30');
ALTER TABLE DL1_B ADD PARTITION (fecha = '2019-05-01');
ALTER TABLE DL1_B ADD PARTITION (fecha = '2019-05-02');
ALTER TABLE DL1_B ADD PARTITION (fecha = '2019-05-03');
ALTER TABLE DL1_B ADD PARTITION (fecha = '2019-05-05');
ALTER TABLE DL1_B ADD PARTITION (fecha = '2019-05-06');
ALTER TABLE DL1_B ADD PARTITION (fecha = '2019-05-07');
ALTER TABLE DL1_B ADD PARTITION (fecha = '2019-05-08');
ALTER TABLE DL1_B ADD PARTITION (fecha = '2019-06-07');
ALTER TABLE DL1_B ADD PARTITION (fecha = '2019-06-09');
ALTER TABLE DL1_B ADD PARTITION (fecha = '2019-06-10');

INSERT OVERWRITE DIRECTORY '/user/jiramos/DL1_B/2019-06-10'
SELECT to_utc_timestamp(from_unixtime(time),'Europe/Madrid'),
        state[0] AS icao24,
        state[1] AS callsign,
        state[2] AS ocoun,
        state[3] AS time_position,
        state[4] AS time_velocity,
        state[5] AS lat,
        state[6] AS lon,
        state[7] AS altitude,
        CAST (IF(state[8]='false', '', 'true') AS BOOLEAN) AS on_ground,
        state[9] AS speed,
        state[10] AS track,
        state[11] AS vrate,
        state[12] AS sensors,
        state[13] AS baltitude,
        state[14] AS quawk,
        CAST (IF(state[15]='false', '', 'true') AS BOOLEAN) AS spi,
        state[16] AS pos_source,
        to_date(to_utc_timestamp(from_unixtime(time),'Europe/Madrid')) `fecha`
FROM    DL1_B opensky LATERAL VIEW explode(states) s AS state
WHERE   `fecha`='2019-06-10';
```

Con este test hemos aprendido lo importante que son los ajustes del clúster y cómo realizamos las consultas para que estas sean eficientes y reduzcan el tiempo de ejecución. El resultado final ha podido disminuir el tiempo en hasta 6 h para realizar la misma tarea.

6.4 Cálculo de Trayectorias

Como prueba final del clúster vamos a crear un pequeño flujo de tareas, *"DataFlow"*, para poder recrear las trayectorias de un vuelo. El concepto de trayectoria consiste en

integrar la latitud,longitud y altitud, esto es la dimensión espacial junto con una dimensión temporal. Si se consigue tener esta relación de datos se pueden realizar muchos estudios, como por ejemplo retrasos, mejorar las rutas para optimizar el espacio aéreo, etc. En conjunto de datos adquirido con Flume disponemos de una relación temporal de los mensajes emitidos por los aviones, con su posición espacial y su identificador podremos obtener los que se denomina *"legs"* de la trayectoria de un avión, esto es los mensajes emitidos desde el despegue de un avión hasta su siguiente aterrizaje.

Hemos diseñado un modelo conceptual que se muestra en la Figura 6.9. En nuestro caso disponemos de una única fuente de datos, Opensky, aunque lo habitual para este tipo de análisis es que se tengan más fuentes. A esta fuente de datos le añadiremos tablas que crearemos a partir de unos ficheros CSV que hemos puesto en el sistema HDFS.

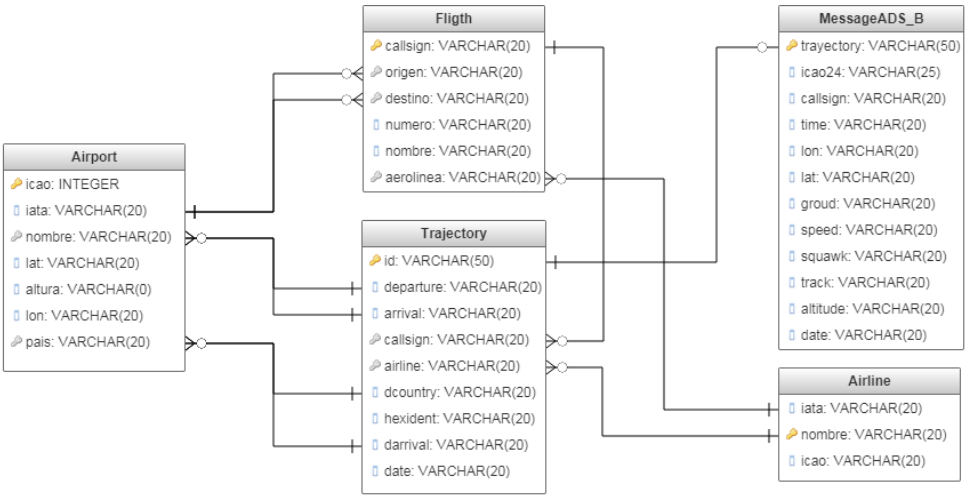

Figura 6.9: Modelo Conceptual

La secuencia de pasos del DataFlow para un día para la implementación de la transformación de los datos en crudo a los datos refinados es:

- **Paso 1.** Crear todas las tablas externas necesarias que son utilizadas durante la transformación: L0, L1, L2, L2_trajectories, L3_vuelo, L3_aerolinea, L3_aeropuerto, L3_trajectories. La tabla de nivel 0 (raw data) es para acceder a los datos de OpenSky depositados directamente por Flume y se crean particiones independientes por cada uno de los días de mensajes ya capturados. La información de aeropuertos, aerolíneas y vuelos se deposita directamente en tablas de nivel 3 (refined data). Las tablas creadas son las siguientes:

Información de los aeropuertos.

```
CREATE TABLE DL3_aeropuerto (nombre STRING,
                             iata STRING,
                             icao STRING,
                             lat STRING,
                             lon STRING,
                             pais STRING,
                             altura STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ';' LINES TERMINATED BY '\n'
    STORED AS TEXTFILE;
LOAD DATA INPATH '/user/jiramos/airports.csv' INTO TABLE DL3_aeropuerto;
```

Información de las aerolíneas.

```
CREATE TABLE DL3_aerolinea (nombre STRING,
                            iata STRING,
                            icao STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ';' LINES TERMINATED BY '\n'
    STORED AS TEXTFILE;
LOAD DATA INPATH '/user/jiramos/airlines.csv' INTO TABLE DL3_aerolinea;
```

Información de vuelos.

```
CREATE TABLE DL3_vuelo (origen STRING,
                        destino STRING,
                        aerolinea STRING,
                        numero STRING,
                        nombre STRING,
                        callsign STRING)
     ROW FORMAT DELIMITED FIELDS TERMINATED BY ';' LINES TERMINATED BY '\n'
    STORED AS TEXTFILE;
LOAD DATA INPATH '/user/jiramos/flights.csv' INTO TABLE DL3_vuelo;
```

Tabla de mensajes refinada.

```
CREATE EXTERNAL TABLE DL2 (trajectory STRING,
                           callsign STRING,
                           icao24 STRING,
                           time BIGINT,
                lat STRING,
                lon STRING,
                altitude STRING,
                speed STRING,
                track STRING,
                squawk STRING,
                ground STRING)
PARTITIONED BY (fecha STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
    LINES TERMINATED BY '\n' STORED AS TEXTFILE;
```

```
ALTER TABLE DL2 ADD PARTITION (fecha = '2019-05-01') LOCATION '/user/jiramos
    /DL2/2019-05-01';

CREATE INDEX indx_callsign ON TABLE DL2(callsign) AS 'org.apache.hadoop.hive
    .ql.index.compact.CompactIndexHandler' WITH DEFERRED REBUILD ;
CREATE INDEX indx_icao24 ON TABLE DL2(icao24) AS 'org.apache.hadoop.hive.ql.
    index.compact.CompactIndexHandler' WITH DEFERRED REBUILD ;
```

Tabla refinamiento trayectorias.

```
CREATE EXTERNAL TABLE DL2_trajectories ( id STRING, callsign STRING,hexident
     STRING,time0 BIGINT, timeN BIGINT)
PARTITIONED BY (fecha STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n' STORED AS TEXTFILE;
ALTER TABLE DL2_trajectories ADD IF NOT EXISTS PARTITION (fecha = '20190501'
    )
LOCATION '/user/jiramos/DL2/trajectories/20190501';

CREATE INDEX indx_callsign ON TABLE DL2_trajectories(callsign)
AS 'org.apache.hadoop.hive.ql.index.compact.CompactIndexHandler' WITH
    DEFERRED REBUILD ;
CREATE INDEX indx_id ON TABLE DL2_trajectories(id)
AS 'org.apache.hadoop.hive.ql.index.compact.CompactIndexHandler' WITH
    DEFERRED REBUILD ;
```

Tabla refinamiento trayectorias.

```
CREATE EXTERNAL TABLE L3_trajectories ( id STRING,
                                        callsign STRING,
                                        hexident STRING,
                                        airline STRING,
                                        departure STRING,
                                        dcountry STRING,
                                        arrival STRING,
                                        darrival STRING)
  PARTITIONED BY (fecha STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',
    ' LINES TERMINATED BY '\n' STORED AS TEXTFILE;
ALTER TABLE L3_trajectories ADD IF NOT EXISTS PARTITION (fecha = '20190501')
     LOCATION '/user/cloudera/L3/trajectories/20190501';

CREATE INDEX indx_callsign ON TABLE L3_trajectories(callsign) AS 'org.apache
    .hadoop.hive.ql.index.compact.CompactIndexHandler' WITH DEFERRED REBUILD
    ;
CREATE INDEX indx_id ON TABLE L3_trajectories(id) AS 'org.apache.hadoop.hive
    .ql.index.compact.CompactIndexHandler' WITH DEFERRED REBUILD ;
```

- **Paso 2.** Insertar los datos en tablas del nivel 1. En este paso se eliminan los mensajes con callsign vacío porque no es un valor descriptivo del vuelo y todos los valores son convertidos en mayúsculas. Esta consulta se es la que se utiliza como prueba de Hive en la Sección 6.3.

on_ground	time_velocity	time_position	Descripción
FALSE	IS NOT NULL	IS NOT NULL	Msg. en vuelo del avion
FALSE	IS NULL	–	Msg. previos al aterrizaje
FALSE	–	–	Msg. previos al aterrizaje + Msg. en vuelo
TRUE	IS NULL	–	Msg. en tierra previos al despegue
TRUE	IS NOT NULL	–	Msg. en tierra
TRUE	–	–	Msg. en tierra + Msg. previos al despegue

Tabla 6.2: Comprobación de la coherencia de mensajes en una trayectoria

- **Paso 3.** Filtrar los datos para hacer la primera inferencia de las trayectorias. Entendemos una trayectoria como el conjunto de los mensajes que tengan el mismo callsign e icao. Para comprender la codificación de los mensajes vea la Tabla 6.2, que muestra la relación entre on_groud, time_position y time_velocity. Para simplificar el problema definimos la trayectoria cómo los mensajes que tenemos de un avión desde que despega hasta que aterriza con el mismo callsign. En esta fase definimos un identificador para cada trayectoria formado por la fecha, el callsign y el icao24, para ello agrupamos los mensajes por callsign y el icao24. De esta forma sabremos la trayectoría de un avion a lo largo del día independientemente del número de aeropuertos en los que haga escala. Para unificar el tiempo de comienzo y de fin tratamos los mensajes según tengan time_position o time_velocity. Todo esto lo realizamos con una sóla consulta creado un mapa de datos almacenado en la tabla L2_trajectories. La consulta es la siguiente:

Conjunto de datos válidos.

```
INSERT OVERWRITE DIRECTORY '/user/jiramos/DL2/trajectories/20190501'
ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n'
    STORED AS TEXTFILE
SELECT concat(concat(concat(concat('20190501','-'), callsign), '-'), icao24)
     AS trajectory,
       callsign,
       icao24,
       IF (MIN(time_position)!='',
            CAST(MIN(time_position) AS STRING),
            CAST(MIN(time_velocity) AS STRING)) AS time0,
          IF (MAX(time_position)!='',
              CAST(MAX(time_position) AS STRING),
              CAST(MAX(time_velocity) AS STRING)) AS timeN
FROM DL1
WHERE fecha ="20190501"
GROUP BY callsign,
         icao24;
```

- **Paso 4.** En este paso se inserta dentro de L2 (segunda tabla de refinamiento) las

trayectorias creadas anteriormente y se hace un join con los mensajes recibidos de OpenSky (tabla L1), solo quedando los mensajes de la fecha en cuestión y que el mensaje se encuentre entre el tiempo de incio y de fin de transmisión de los mensajes seleccionados anteriormente. El siguiente código se encarga de realizar esta tarea:

Segundo refinamiento.

```
INSERT OVERWRITE DIRECTORY '/user/jiramos/L2/20190501'
ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n'
    STORED AS TEXTFILE
SELECT traj.id,
       traj.callsign,
       message.icao24,
       message.time_position,
       message.lat,
       message.lon,
       message.altitude,
       message.speed,
       message.track,
       message.quawk,
       message.on_ground
FROM DL2_trajectories AS traj
JOIN DL1 AS message ON traj.hexident = message.icao24
WHERE traj.fecha = "20190501"
  AND message.fecha = "20190501"
  AND (message.time_position >= traj.time0
       AND message.time_position <= traj.timeN);
```

- **Paso 5.** Por último asociaremos con la trayectoria el aeropuerto de origen y destino, que sería el último paso de refinamiento de la información.

```
INSERT OVERWRITE DIRECTORY '/user/jiramos/DL3/trajectories/20190501'
ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n'
    STORED AS TEXTFILE
SELECT traj.id, traj.callsign, traj.hexident, airl.nombre, airp1.nombre,
    airp1.pais, airp2.nombre, airp2.pais
FROM l2_trajectories traj,
     l3_vuelo fl,
     l3_aerolinea airl,
     l3_aeropuerto airp1,
     l3_aeropuerto airp2
WHERE length(fl.callsign) > 0
  AND traj.callsign = fl.callsign
  AND fl.aerolinea = airl.iata
  AND fl.origen = airp1.iata
  AND fl.destino = airp2.iata
  AND fecha = "20190501";
```

Finalmente una vez que tenemos el DataFlow para un día podremos crear una tarea

con Oozie para automatizar este proceso y que sea ejecutado todos los días obteniendo las trayectorias de los vuelos diarios. Esta es la prueba que hemos realizado para el módulo de Oozie, en la Figura 6.10 se muestra el editor con el que hemos diseñado el flujo, desde este mismo editor se puede lanzar la tarea o programar.

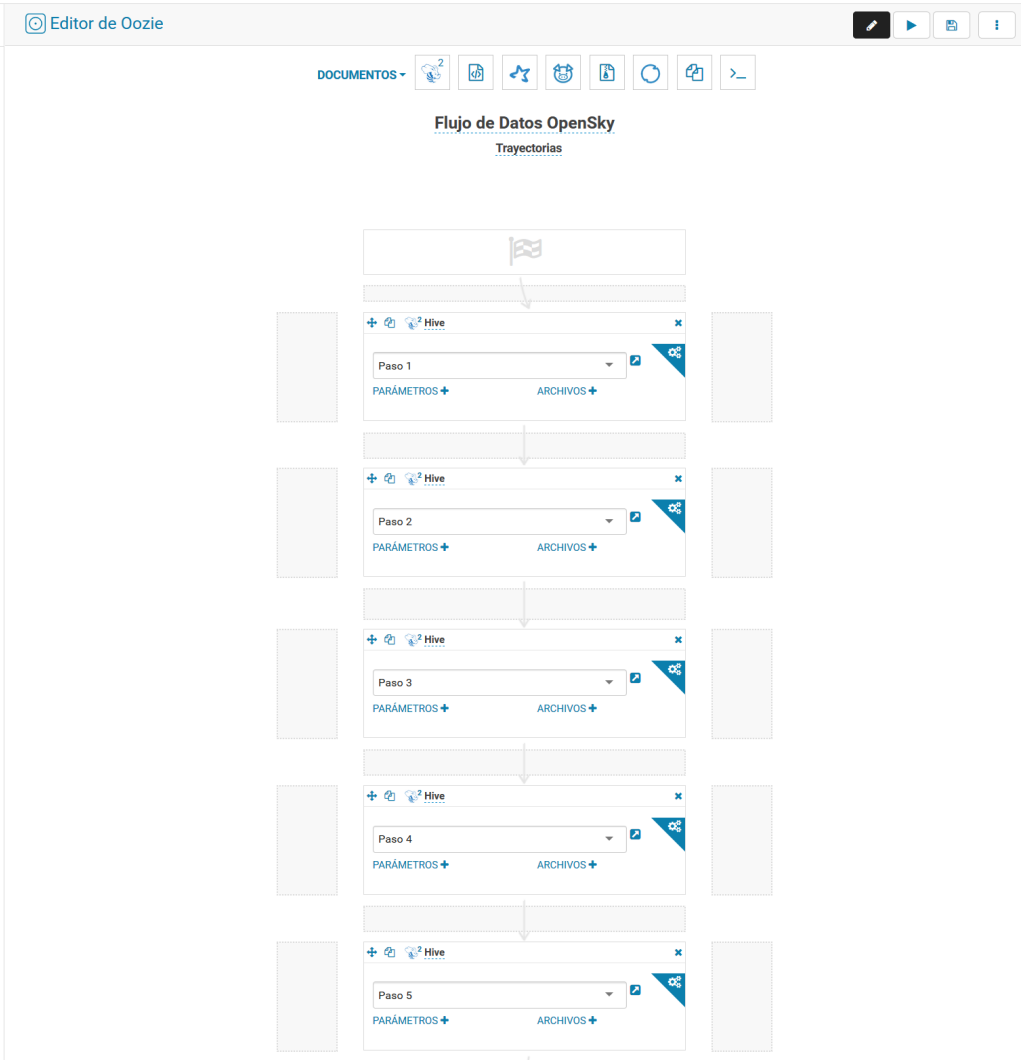

Figura 6.10: Editor Oozie

IV Cuarta Parte: Conclusiones y Trabajo Futuro

7. Conclusiones y Trabajo Futuro

7.1 Conclusiones

A lo largo de este trabajo de fin de máster hemos creado un método para desplegar clústeres de forma semi-automática basados en la distribución CDH de Cloudera y Cloudera Manager en un entorno de virtualización con tecnología propia. Crear un clúster con tecnología propia era el principal objetivo de este trabajo de fin de máster. Hemos descrito cómo crear un modelo de máquina virtual desde cero, basado en CentOS, creado los playbooks necesarios para automatizar su despliegue y especialización, según el rol para el que se destinaría la máquina virtual, y por último hemos creado con ese despliegue de máquinas virtuales un clúster basado en Cloudera Manager completamente funcional. Para llegar a este punto, el proceso ha sido largo, se ha reestructurado todo el almacenamiento del sistema de virtualización existente para optimizar, en la medida de lo posible, el clúster de máquinas virtuales que da servicio al clúster de Cloudera. En las primeras pruebas se concluyó que un clúster Hadoop necesita un sistema de almacenamiento eficiente. Se han estudiado las distintas herramientas existentes para automatizar tareas de mantenimiento y gestión de equipos informáticos. Este trabajo de fin de máster nos ha permitido obtener la experiencia necesaria en Ansible, tecnología seleccionada de las existentes en el mercado, para automatizar la creación de clústeres de una forma sencilla. A lo largo de este trabajo de fin de máster hemos creado 7 clústeres para optimizar este proceso lo que demuestra lo sencilla y eficiente que es esta tecnología. Se han estudiado varias arquitecturas de Big Data comparándolas con Cloudera, esto nos ha permitido familiarizarnos con el ecosistema

Hadoop y ahondar en el funcionamiento interno de varias de sus herramientas que debemos administrar. Este estudio demuestra que no hay muchas diferencias entre unas y otras. Desplegar Cloudera Manager, entender su funcionamiento, la gran cantidad de opciones, medidas y herramientas que ofrece ha sido otro objetivo conseguido. Si bien es cierto que se necesita más tiempo para conocer y comprender todo este sistema, ahora ya se dispone de un clúster en producción para poder dar servicio y seguir aprendiendo a administrar el clúster. Se han realizado varias pruebas con las tecnologías propuestas en los objetivos, como son Yarn, Oozie, Hive, Hue, HDFS, etc. No hemos podido realizar pruebas de todas las tecnologías propuestas, pero si han quedado disponibles y operativas en el clúster. Las pruebas que se han realizado en el clúster nos han permitido evaluar y familiarizarnos con el sistema de monitorización de Cloudera, para lo cual hemos creado un Dashboard personalizado para ver la carga de los nodos. El sistema de monitorización de Cloudera y del sistema de virtualización nos ha permitido dimensionar los nodos y comprender qué tipo de carga generan las pruebas realizadas. Por último, en este apartado, hemos aprendido a crear un sencillo Dataflow basado en datos de Opensky, ilustrando parte de la arquitectura que forma parte de un proyecto Big Data. Big Data ha influido y perfilado el entorno profesional de un informático, que hoy en día debe ser más especializado que hace unos años. Esto es debido al gran número de tecnologías que existen en el ecosistema Big Data. Es evidente que para convertirse en un experto en tantas tecnologías que conviven y dependen unas de otras se necesita más tiempo que el que se emplea en un trabajo fin de máster, este proyecto sería el principio para convertirse en un Data Engineer.

7.2 Trabajo Futuro

Cuando se plantea un trabajo de fin de máster se intenta responder alguna pregunta básica y cumplir una serie de objetivos, si se desarrolla con un mínimo de entusiasmo contribuye a despejar algunas incógnitas sobre el tema tratado, pero de forma simultánea, genera nuevas preguntas, nuevas ideas y/o abre nuevas vías de trabajo. Presentar estas ideas y vías de trabajo es la intención de este apartado.

En relación con tema que hemos tratado en este trabajo sería interesante crear un clúster mucho más grande formado por varios chasis, con nodos físicos y virtuales, un clúster heterogéneo con nodos en la nube de AWS o Azure. Para poder automatizar el despliegue en otros sistemas de Cloud Computing deberíamos ajustar el sistema de despliegue actual, y ajustarlo a otros entornos de Cloud Computing.

Cloudera Manager permite la gestión de varios clústeres, esta característica sería muy interesante para poder disponer de clústeres especializados y optimizados al problema que

se quiere resolver. Mantener el clúster para obtener más experiencia en su administración, brindar los servicios del mismo a educación e investigación sería una de las salidas al trabajo desarrollado y podríamos seguir formándonos en esta compleja tecnología. La autenticación y encriptación es un tema pendiente, lo cual podría hacerse contra un Active Directorio de Microsoft Windows o contra un LDAP como el que dispone la UVa. Activar el sistema de autenticación nos permitiría estudiar la sobrecarga que supone encriptar los datos y profundizar en la seguridad de los Data Lakes.

Bibliografía

Libros

[23] Murthy Arun C & Vavilapalli Vinod Kumar & Eadline Doug & Markham Jeffrey. *Apache Hadoop YARN: moving beyond MapReduce and batch processing with Apache Hadoop 2*. Pearson Education, 2014 (véase página 21).

[26] Alapati Sam R. *Expert Hadoop Administration: Managing, Tuning, and Securing Spark, YARN, and HDFS*. Addison-Wesley Professional, 2016 (véase página 30).

[28] Menon Rohit. *Cloudera administration handbook*. Packt Publishing Ltd, 2014 (véase página 102).

Artículos

[16] Allae Erraissi, Abdessamad Belangour y Abderrahim Tragha. «A Comparative Study of Hadoop-based Big Data Architectures.» En: *IJWA* 9.4 (2017), páginas 129-137 (véase página 52).

[17] Allae Erraissi, Abdessamad Belangour y Abderrahim Tragha. «Digging into hadoop-based big data architectures». En: *International Journal of Computer Science Issues (IJCSI)* 14.6 (2017), páginas 52-59 (véase página 52).

[29] R. D. Schneider. «HADOOP BUYER'S GUIDE». En: (2014) (véase página 52).

[30] R Shireesha y Sunil Bhutada. «A study of tools, techniques, and trends for big data analytics». En: *International Journal of Advance Computing Techniques and Applications* 4.1 (2016), páginas 152-158 (véase página 11).

[33] R. Senior V. Starostenkov y D. Developer. «Hadoop Distributions: Evaluating Cloudera, Hortonworks, and MapR in Micro-benchmarks and Real-world Applications». En: (2013) (véase página 52).

[35] K W. Report Takeaways. «A Comparative Study of Hadoop-based Big Data Architectures.» En: (2016) (véase página 52).

Web

[1] Apache. *Hadoop*. (Accedido 03-05-2019). URL: http://hadoop.apache.org/ (véase página 14).

[2] Varios Autores. *Inteligencia Empresarial*. (Accedido 04-03-2019). 2019. URL: https://es.wikipedia.org/wiki/Inteligencia_empresarial (véase página 17).

[3] Diego Calvo. *Flume*. (Accedido 13-03-2019). URL: http://www.diegocalvo.es/flume/ (véase página 38).

[4] Cloudera. *Cloudera and Hortonworks Announce Merger*. (Accedido 03-10-2018). 2018. URL: https://www.cloudera.com/about/news-and-blogs/press-releases/2018-10-03-cloudera-and-hortonworks-announce-merger-to-create-worlds-leading-next-generation-data-platform-and-deliver-industrys-first-enterprise-data-cloud.html (véase página 55).

[5] Cloudera. *Cloudera Flume*. (Accedido 13-02-2019). URL: https://www.cloudera.com/documentation/enterprise/latest/topics/flume.html (véase página 38).

[6] Cloudera. *Cloudera HDFS*. (Accedido 14-02-2019). URL: https://www.cloudera.com/documentation/enterprise/6/6.2/topics/admin_hdfs_config.html (véase página 26).

[7] Cloudera. *Cloudera Hive*. (Accedido 18-02-2019). URL: https://www.cloudera.com/documentation/enterprise/6/6.2/topics/admin_hive_manage_intro.html (véase página 34).

[8] Cloudera. *Cloudera Hue*. (Accedido 29-02-2019). URL: https://www.cloudera.com/documentation/enterprise/6/6.2/topics/admin_hue_config.html (véase página 39).

[9] Cloudera. *Cloudera Kafka*. (Accedido 18-02-2019). URL: https://www.cloudera.com/documentation/enterprise/6/6.2/topics/kafka_tour.html (véase página 41).

[10] Cloudera. *Cloudera Oozie*. (Accedido 27-02-2019). URL: https://www.cloudera.com/documentation/enterprise/6/6.2/topics/cm_mc_oozie_service.html (véase página 40).

[11] Cloudera. *Cloudera Search*. (Accedido 03-06-2019). URL: https://www.cloudera.com/documentation/enterprise/latest/topics/search_architecture.html (véase página 103).

[12] Cloudera. *Cloudera Solr*. (Accedido 27-02-2019). URL: https://www.cloudera.com/documentation/enterprise/6/6.2/topics/cm_mc_solr_service.html (véase página 41).

[13] Cloudera. *Cloudera Spark*. (Accedido 21-02-2019). URL: https://www.cloudera.com/documentation/enterprise/6/6.2/topics/admin_spark.html (véase página 40).

[14] Cloudera. *Cloudera ZooKeeper*. (Accedido 14-02-2019). URL: https://www.cloudera.com/documentation/enterprise/6/6.2/topics/cm_mc_zookeeper_service.html (véase página 32).

[15] Cloudera. *Lily Hbase*. (Accedido 02-02-2019). URL: https://www.cloudera.com/documentation/enterprise/6/6.2/topics/search_config_hbase_indexer_for_search.html (véase página 36).

[18] Eurecat. *Los 7 perfiles clave de los profesionales del Big Data*. (Accedido 14-12-2018). 2018. URL: https://landings.powerdata.es/guia-sobre-data-lake?hsCtaTracking=4735c4be-4762-4617-ab74-556c00d50bdf%7Cb0bf7206-bbd0-44b4-a17f-f062761a0a73 (véase página 17).

[19] Apache Fundation. *ZooKeeper*. (Accedido 14-03-2018). URL: https://zookeeper.apache.org/doc/r3.5.1-alpha/zookeeperOver.html (véase página 32).

[20] Ganglia. *Ganclia*. (Accedido 29-04-2019). 2019. URL: http://ganglia.sourceforge.net/ (véase página 46).

[21] Guru99. *Data Lake: Superando las limitaciones del Data Warehouse*. (Accedido 02-01-2019). 2019. URL: https://landings.powerdata.es/guia-sobre-data-lake?hsCtaTracking=4735c4be-4762-4617-ab74-556c00d50bdf%7Cb0bf7206-bbd0-44b4-a17f-f062761a0a73 (véase página 25).

[22] Apache Hadoop. *Hue*. (Accedido 1-03-2019). URL: http://gethue.com/ (véase página 39).

[24] PowerData. *What is Data Lake?* (Accedido 02-01-2019). 2019. URL: https://www.guru99.com/data-lake-architecture.html (véase página 24).

[25] Proxmox. *Open-Source Virtualization Platform*. (Accedido 24-05-2019). 2019. URL: https://www.proxmox.com/en/ (véase página 68).

[27] Redhat. *Ansible*. (Accedido 24-05-2019). 2019. URL: https://www.ansible.com/ (véase página 72).

[31] Shubham Sinha. *HBase*. (Accedido 29-04-2019). 2019. URL: https://www.edureka.co/blog/hbase-architecture/ (véase página 37).

[32] Mónica Tilves. *Los "grandes datos" de Facebook*. (Accedido 04-03-2019). 2012. URL: https://www.silicon.es/los-grandes-datos-de-facebook-500-tb-de-informacion-generada-cada-dia-26590 (véase página 13).

[34] El Método Monte Carlo. Estimación del Valor de Pi. *Rafael Pérez Laserna*. (Accedido 10-06-2019). URL: https://www.geogebra.org/m/cF7RwK3H (véase página 118).

[36] Wikipedia. *Multitenant*. (Accedido 03-06-2019). URL: https://es.wikipedia.org/wiki/Tenencia_m%C3%BAltiple (véase página 106).

Índice de figuras