

Big Data

Análisis de herramientas y soluciones



Big Data es el sector emergente dentro del área de las tecnologías de la información y la comunicación que se preocupa de cómo almacenar y tratar grandes cantidades de información o conjuntos de datos. En este trabajo se hace un estudio dentro de un marco teórico de las distintas herramientas y un análisis práctico de algunas de las soluciones existentes.

PROYECTO FINAL DE CARRERA

BIG DATA- ANÁLISIS DE HERRAMIENTAS Y SOLUCIONES

Autor: Robert Serrat Morros

Compañero: Jordi Sabater Picañol

Directores: Manuel Pando Mones

José Roldán Rosón

Ponente: Ruth Raventós Pages

Everis – Facultat d'Informàtica de Barcelona – UPC

Noviembre de 2013



AGRADECIMIENTOS

A mi familia por apoyarme y ayudarme siempre y darme todas las oportunidades de las que he disfrutado,

A Caro por la paciencia, la ayuda y la compañía recibida,

A Jordi Sabater por las horas de trabajo compartidas y por su compañía y ayuda durante la realización del trabajo,

A José Roldán y Manuel Pando por el soporte y guía ofrecidos,

A Everis por la oportunidad y la confianza depositada.

ÍNDICE

1.	Introducción.....	1
1.1.	Everis.....	1
1.2.	Objetivos del proyecto.....	2
1.2.1.	Estudio teórico	2
1.2.2.	Estudio técnico	3
1.2.3.	Implementación de un caso de uso	3
1.3.	Trabajo conjunto.....	4
1.4.	Motivación personal	5
2.	Planificación.....	6
2.1.	Planificación Inicial.....	6
2.2.	Planificación final	8
2.3.	Metodología de trabajo	10
2.4.	Presupuesto	10
3.	Big Data.....	12
3.1.	Las cinco V.....	12
3.2.	Tipos de información	13
3.3.	¿Cuál es el panorama actual de Big Data?	14
3.4.	Casos de uso	15
3.4.1.	Análisis de negocios	15
3.4.2.	Análisis de sentimiento	15
3.4.3.	Seguridad	16
3.4.4.	Usos médicos y científicos.....	16
3.5.	Arquitectura Big Data.....	16
3.5.1.	Recolección de datos.....	17
3.5.2.	Almacenamiento	17
3.5.3.	Procesamiento y análisis	20
3.5.4.	Visualización.....	21
3.6.	Paradigmas Big Data	21
3.6.1.	MapReduce	21
3.6.2.	Massive Parallel Processing (MPP).....	23
3.6.3.	Comparativa	23
3.6.4.	Arquitectura mixta	26
3.7.	Soluciones	27
3.7.1.	Distribuciones.....	27

3.7.2.	Appliance.....	27
3.7.3.	Cloud	28
3.7.4.	Comparativa	29
4.	Hadoop	32
4.1.	Hadoop 1.0.....	33
4.1.1.	Hadoop Distributed File System (HDFS)	33
4.1.2.	MapReduce 1.0	39
4.2.	Hadoop 2.0.....	46
4.2.1.	HDFS 2.0	46
4.2.2.	MapReduce 2.0	51
5.	Herramientas Hadoop	53
5.1.	Utilidades	53
5.1.1.	Avro	53
5.1.2.	ZooKeeper	56
5.1.3.	Solr	58
5.2.	Recolección de datos	59
5.2.1.	Chukwa.....	59
5.2.2.	Flume.....	62
5.3.	Almacenamiento.....	65
5.3.1.	Cassandra	65
5.3.2.	Hive	68
5.4.	Desarrollo de procesos para el tratamiento de datos	70
5.4.1.	Mahout.....	70
5.4.2.	Oozie	71
5.4.3.	Pig.....	73
5.5.	Administración.....	75
5.5.1.	Hue	75
6.	Distribuciones Hadoop	77
6.1.	Cloudera	77
6.1.1.	Cloudera Distribution including Apache Hadoop	78
6.1.2.	Cloudera Manager.....	79
6.1.3.	Cloudera Navigator	80
6.1.4.	Productos	80
6.2.	DataStax	81
6.2.1.	DataStax Enterprise.....	81
6.3.	Pivotal	85
6.3.1.	Pivotal HD Enterprise	85

6.3.2.	HAWQ - Advanced database Services	86
6.3.3.	GemFire XD - Real-time Database Services.....	86
6.3.4.	Pivotal Analytics workbench (AWB)	86
6.3.5.	Pivotal Data Computing Appliances (DCA)	86
6.3.6.	versión de comunidad.....	87
6.4.	Comparativa.....	88
7.	Caso de uso principal	90
7.1.	Caso de uso	90
7.2.	Arquitectura de la solución	90
7.3.	Proceso	91
7.3.1.	Recolección de datos.....	91
7.3.2.	Almacenamiento	92
7.3.3.	Procesamiento	95
7.3.4.	Data Warehousing.....	96
7.4.	Resultados.....	98
7.5.	Conclusiones	99
8.	Caso de uso extra.....	100
8.1.	Diseño de pruebas	100
8.2.	Resultados de las pruebas	101
8.3.	Conclusiones	102
9.	Pruebas	104
9.1.	Administración	104
9.1.1.	A1 - Instalación clúster	104
9.1.2.	A2 - Escalabilidad de nodos.....	104
9.1.3.	A3 - Escalabilidad de servicios.....	105
9.1.4.	A4 - Monitorización.....	105
9.1.5.	A5 - Configuración.....	106
9.2.	HDFS - Almacenamiento	107
9.2.1.	HD1 - Replicación de datos	107
9.2.2.	HD2 - Rendimiento	107
9.2.3.	HD3 - Tolerancia a fallos.....	109
9.3.	Flume - Recolección de datos	110
9.3.1.	F1 - Pérdida de datos.....	110
9.3.2.	F2 - rendimiento	110
9.3.3.	F3 - Tolerancia a fallos 1.....	112
9.3.4.	F4 - Tolerancia a fallos 2.....	112
9.4.	MapReduce - Procesos.....	114

9.4.1.	MR1 - Rendimiento y escalabilidad del Mapper	114
9.4.2.	MR2 - Rendimiento y Escalabilidad del Reducer.....	116
9.4.3.	MR3 - Tolerancia a fallos	117
9.5.	YARN - Procesos	119
9.5.1.	Y1 - Uso de los recursos	119
9.5.2.	Y2 - Escalabilidad.....	119
9.5.3.	Y3 - rendimiento.....	121
9.5.4.	Y4 - Comparación con MRv1	124
10.	conclusiones	126
	Anexos	i
A.	Glosario.....	i
B.	Planificación.....	vi
C.	Arquitectura.....	x
D.	Procesos de instalación	xii
E.	Procesos de configuración	xvi
F.	Códigoxviii
	Bibliografía.....	xix

1. INTRODUCCIÓN

Esta memoria es el resultado del Proyecto de Final de Carrera de los estudios de Ingeniería Superior en Informática, impartidos por la Facultad d'Informàtica de Barcelona de la Universitat Politècnica de Catalunya, del alumno Robert Serrat Morros. La elaboración del trabajo ha sido el resultado de un convenio de cooperación educativa con la empresa Everis y ha durado nueve meses (de febrero a noviembre de 2013).

Everis ofrecía una beca para realizar un estudio del arte de las tecnologías Big Data y ampliar los conocimientos sobre las herramientas existentes. Big Data es un sector emergente dentro del área de las tecnologías de la información y comunicación (TIC) y, como se explica más adelante, todo lo relacionado está en proceso de desarrollo continuo. La realización de este estudio es muy interesante a nivel académico y formativo, pues la mayoría de las tecnologías estudiadas son tan recientes que apenas tienen cabida en los planes de estudios ofrecidos hoy en día. Además, en el plano empresarial las compañías están empezando a adoptar soluciones Big Data dentro de sus infraestructuras, por lo que es un punto clave dentro de la evolución y el crecimiento de éstas.

El proyecto se ha dividido en tres fases:

- Elaboración de un estudio teórico de la situación del campo Big Data.
- Elaboración de un estudio técnico en una solución real Big Data.
- Implementación de un caso de uso mediante herramientas Big Data.

Debido a la envergadura del proyecto Everis pensó que lo más indicado era que el proyecto lo realizaran dos estudiantes. De esta manera, el alumno Jordi Sabater Picañol, del Grado en Ingeniería en Informática impartido también por la Facultad d'Informàtica de Barcelona, ha formado parte del proyecto y ha participado a partes iguales en su realización. En el apartado 1.3. *Trabajo conjunto* se indica la división de trabajo realizada con Jordi. Esta memoria ha sido escrita por Robert Serrat y se describen solamente las tareas que ha realizado su autor.

En esta memoria se empieza contextualizando el trabajo realizado con una breve descripción de Everis y a qué áreas de ésta pertenece, junto con una explicación de los objetivos y la planificación del proyecto, así como la metodología de trabajo utilizada. Seguidamente se describe Big Data con el objetivo de poner al día al lector y para entender los siguientes apartados de la memoria, que pertenecen a las fases definidas anteriormente. Finalmente en el anexo se encuentra un glosario con los términos que puedan suponer alguna dificultad para la comprensión del texto, además de otras secciones para ampliar la información del trabajo.

1.1. EVERIS

Everis es una empresa multinacional con diecisiete años de experiencia en el campo de la consultoría de negocios e IT y que cuenta con más de 10.000 profesionales de distintos sectores. Fundada en Madrid el año 1996, actualmente opera en países de Europa, Estados Unidos y Latinoamérica y tiene como socios inversores a los fondos de inversión 3i y Hutton Collins y al grupo Landon; además de un grupo de pequeños accionistas. Está dedicada a ofrecer soluciones de negocio, estrategia y desarrollo, mantenimiento de aplicaciones tecnológicas y *outsourcing*; cubriendo los sectores de telecomunicaciones, entidades financieras, industria, *utilities & energía*, seguros, administraciones públicas, media y sanidad.



Everis cuenta con cinco líneas o unidades de negocio [1]:

- **Business consulting:** se encarga de los proyectos de estrategia corporativa, consultoría de negocio e ingeniería de procesos. Su actividad se centra en el conocimiento sectorial, en la innovación de servicios y en la especialización.
- **Solutions:** se enfoca en la definición, diseño e implantación de soluciones tecnológicas y a la gestión y operación de aplicaciones, infraestructuras y procesos de *outsourcing*. Se busca el uso de metodologías para aumentar la calidad, traspaso de producción a centros de alto rendimiento y para la especialización funcional y tecnológica.
- **Centers:** se basa en la utilización de alto rendimiento. Gracias a los más de cuatro años de experiencia ya cuenta con la estructura y las capacidades para realizar actividades de forma industrializada. Tiene centros en Sevilla, Murcia, Alicante, Temuco, San Miguel de Tucumán y Uberlandia.
- **Business Process Outsourcing (BPO):** se orienta a ofrecer servicios de externalización de procesos de negocio bajo acuerdos de nivel de servicios, facilitando a sus clientes disponer de mayor capacidad interna para realizar funciones que le aporten más valor a su negocio.
- **Initiatives:** investiga las posibilidades que ofrece el mercado para abrir nuevos negocios en los que invertir con sus clientes.

También cuenta con una división especializada en el asesoramiento financiero, la **FAS (Financial Advisory Services)**.

Dentro de Everis Solutions se encuentra la unidad de Servicio de Tecnología, dedicada a la definición e implantación de arquitecturas tecnológicas, en *outsourcing* BI, arquitecturas tecnológicas y en la gestión documental. Everis Solutions está dividida en cinco unidades:

- Arquitectura
- Business Intelligence (BI)
- Enterprise Content and Service Management
- Servicios de infraestructura
- Gobierno IT

Por la naturaleza del trabajo, el proyecto se ha desarrollado entre las unidades de arquitectura, por el desarrollo de nuevas tecnologías contando con la dirección de Manuel Pando Mones; y de BI, por el estudio de las plataformas de análisis de datos, con la tutoría de José Roldán Rosón.

1.2. OBJETIVOS DEL PROYECTO

Al inicio del proyecto se definieron dos objetivos principales:

1. Realizar un estudio teórico sobre Big Data.
2. Diseñar e implementar dos ejercicios pilotos con el propósito de comparar dos soluciones.

A medida que avanzaba el proyecto y debido a distintas dificultades -comentadas en el apartado de planificación- junto con un mayor conocimiento teórico, se redefinió el proyecto pasando a tener las tres fases comentadas en la introducción. Se cambiaron los dos pilotos por una fase de pruebas a modo de estudio técnico y por la implementación de un caso de uso real sobre una solución Big Data. Cada una de estas fases tiene también pequeños objetivos asignados.

1.2.1. ESTUDIO TEÓRICO

- Puesta al día de Big Data: definición, motivos de su aparición y evolución.

Introducción

- Definición de los distintos casos de uso en los que Big Data tiene influencia actualmente y en los que la tendrá en un futuro.
- Estudio y comparación teórica de los distintos paradigmas Big Data así como sus distintas arquitecturas de *software* y *hardware*.
- Análisis de mercado de las soluciones y las herramientas y realización de una valoración comparativa entre ellas.

El principal objetivo del estudio teórico, a parte de la consolidación del conocimiento sobre Big Data, es especificar y diseñar una infraestructura y solución adecuada -tanto en *software* como en *hardware*- para la siguiente fase del proyecto, el estudio técnico.

1.2.2. ESTUDIO TÉCNICO

- Diseño de las pruebas a realizar sobre la infraestructura especificada en la fase anterior.
- Instalación y configuración de la infraestructura: despliegue de la solución en un clúster.
- Realización de las pruebas referentes a distintos aspectos de la solución escogida:
 - **Escalabilidad:** comprobar la facilidad de instalación de nuevos nodos en un clúster y su influencia en el rendimiento final.
 - **Rendimiento:** comparar las ventajas que ofrece modificando diversos aspectos de su configuración.
 - **Adaptación:** verificar la flexibilidad de la solución a distintos problemas con el tratamiento de la información.
 - **Tolerancia a fallos:** probar la tolerancia de la solución a distintos tipos de errores: caídas de nodos, desconexiones de red, etc.
 - **Facilidad de uso.**
 - **Comparativas:** comparar el resultado final (en tiempo y productividad) de la solución comparada con otras soluciones más tradicionales.
- Análisis de los resultados obtenidos.

Los conocimientos técnicos obtenidos durante esta fase también son uno de los objetivos que se buscan para poder implementar un caso de uso en la siguiente fase. Además, junto con el estudio teórico, se pretende obtener la capacidad de decidir qué una solución es óptima para cada caso.

1.2.3. IMPLEMENTACIÓN DE UN CASO DE USO

Aprovechando la infraestructura de la fase anterior se pretende realizar la implementación de un caso de uso, con los siguientes objetivos:

- Diseño de un caso de uso y elección de las soluciones y herramientas a usar.
- Implementación y configuración de las diferentes etapas:
 - Recolección de datos.
 - Almacenamiento de datos.
 - Análisis de la información.
 - Visualización de resultados.
- Despliegue sobre la infraestructura y ejecución.
- Análisis y conclusiones del desarrollo y del funcionamiento.

1.3. TRABAJO CONJUNTO

Como se ha comentado en la introducción, este trabajo se ha realizado conjuntamente con el alumno Jordi Sabater Picañol, que realizaba su Treball de Final de Grau (TFG). A pesar de la repartición y organización de las distintas tareas a hacer para cumplir los objetivos, el trabajo ha contenido partes conjuntas como el estudio inicial sobre Big Data, sus tecnologías, ventajas y soluciones; el diseño de las pruebas y del caso de uso implementado. Cada uno ha tenido una parte de trabajo adicional, centrándose en las pruebas específicas de una materia a estudiar y en la implementación de distintas capas del caso de uso.

En la *Tabla 1* se puede ver la distribución que se ha seguido a la hora de dividir el trabajo. En esta tabla no se muestran las fases iniciales del proyecto -como la puesta al día de Big Data- ya que son tareas realizadas por los dos por necesidad. Además, algunas de estas tareas están marcadas para los dos ya que al final han sido cosas en común. Por ejemplo, el estudio de la distribución Hadoop de Cloudera, al ser la elegida para el desarrollo, ha sido realizado por los dos. Lo mismo ocurre con algunas de las herramientas.

Tarea	Jordi Sabater	Robert Serrat
Herramientas Hadoop		
Flume	X	X
Hive	X	X
Mahout	X	X
Oozie	X	X
Hue	X	X
Avro		X
ZooKeeper		X
Cassandra		X
Solr		X
Pig		X
Chukwa		X
Sqoop	X	
Ambari	X	
HBase	X	
Whirr	X	
HCatalog	X	
Hama	X	
Distribuciones Hadoop		
Cloudera	X	X
Pivotal HD		X
DataStax Enterprise		X
IBM InfoSphere	X	
MapR	X	
Hortonworks	X	

Introducción

Pruebas		
Flume		X
HDFS		X
Hive/Impala	X	
MapReduce/YARN		X
Oozie	X	
Pentaho	X	
Caso de uso de redes sociales		
Twitter		X
Flume		X
MapReduce para diccionarios	X	
Diccionarios		X
Hive/Impala		X
MapReduce+Mahout	X	
Oozie	X	
Pentaho	X	
Caso de uso de logs		
Implementación MapReduce	X	
Implementación secuencial		X

Tabla 1: Distribución de las tareas.

1.4. MOTIVACIÓN PERSONAL

Las razones por las que he escogido este proyecto como PFC han sido diversas. Una de las principales motivaciones ha sido la experiencia que me aporta el entrar a trabajar en una compañía como Everis e introducirme en la dinámica y empezar a familiarizarme con un entorno de trabajo de una gran empresa. A nivel profesional me ha servido para coger experiencia y empezar a entender cómo funcionan proyectos más grandes que en los que había estado hasta ese momento. El hecho de pertenecer a dos líneas distintas -BI y arquitectura- de la empresa también ha servido para darle dos enfoques diferenciados a los objetivos a cumplir y saber hacia dónde enfocar mi carrera.

Además, el área en el que se enfoca el proyecto era un ámbito en el que había trabajado anteriormente en algunas de las asignaturas de la carrera, pero a la vez era adentrarse en algo desconocido y nuevo para mí, ya que Big Data es un sector joven -aparecido hace pocos años- y que no para de crecer día a día. Prueba de ello es que durante la realización del proyecto no han parado de salir herramientas nuevas para las soluciones ya existentes.

2. PLANIFICACIÓN

En este apartado se empieza describiendo la planificación inicial así como los objetivos originales del proyecto. Como se explica más adelante, tanto la planificación como los objetivos se han tenido que modificar a medida que el proyecto avanzaba, de manera que también se incluye un apartado con la modificación de la planificación y los objetivos finales. Todas las planificaciones de esta sección son del proyecto en global, y no solo del trabajo realizado por el autor. También se incluye un pequeño apartado de la metodología de trabajo seguida y otro con el presupuesto del proyecto.

2.1. PLANIFICACIÓN INICIAL

La planificación inicial del proyecto contemplaba la realización de dos pilotos y se estructuraba en cinco iteraciones:

- Estado del arte:** contextualización del proyecto y estudio teórico de la situación de Big Data, las soluciones existentes y las herramientas disponibles. Se escoge las tecnologías a estudiar en las siguientes iteraciones.
- Especificación de los pilotos y aprendizaje:** investigación sobre los casos de uso actuales para soluciones Big Data y diseño del que se realizará en los pilotos. También se aprovechará para empezar el estudio de la herramienta Flume, para la recolección de datos y documentar lo estudiado hasta el momento.
- Implementación de la arquitectura de los pilotos:** la empresa hace las gestiones para conseguir la infraestructura y las licencias de software necesarias. De mientras, se implementa la capa de recolección de datos y se hace una presentación del proyecto al gerente. Una vez conseguidas las infraestructuras y las licencias, se procederá a su instalación y configuración, así como una toma de contacto inicial a modo de familiarización con el entorno.
- Implementación de los casos de uso:** implementación de los pilotos en paralelo, donde se valorará la facilidad de uso e implementación de las soluciones y herramientas así como el rendimiento y productividad.
- Análisis de los resultados obtenidos y conclusiones:** conclusiones finales sobre todo el proyecto y presentación de los resultados obtenidos.

La *Figura 1* muestra un diagrama de Gantt general con la duración planificada para cada iteración. Para facilitar la lectura, los diagramas detallados con todas las tareas y las dependencias (*Figura 52* y *Figura 53*) se han colocado en el apartado *B. Planificación* del anexo. En este punto se habla de iteraciones debido a que es la palabra usada internamente durante la realización del proyecto, pero en realidad por el tipo de metodología usada se trata de fases.

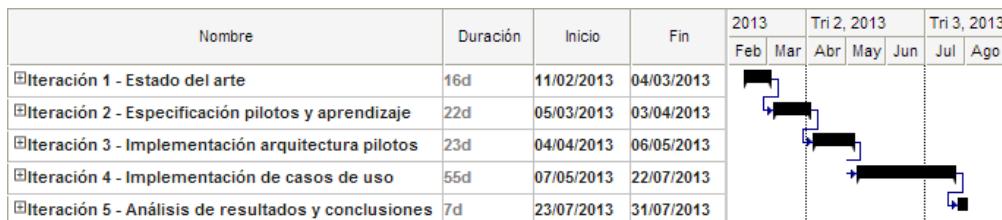


Figura 1: Vista general de la planificación inicial.

La *Tabla 2* muestra un listado de todas las tareas, con la duración planeada y las fechas de inicio y fin para cada una. Al final de cada iteración hay planeada una reunión de seguimiento con los tutores de la empresa de duración de una hora y, para terminar, la revisión de la planificación.

Planificación

En esta planificación, cada piloto tenía su propio responsable, de manera que cada uno de los integrantes realizaba su propio piloto, tanto en la implementación y despliegue como en el apartado de valoraciones. Además, la tarea de adquisición de hardware y de las licencias correspondía a alguien de la empresa.

Tarea	Duración	Inicio	Fin
Iteración 1 – Estado del arte	16 días	11/02/2013	04/03/2013
Contextualización del proyecto	1 día	11/02/2013	11/02/2013
Planificación inicial	1 hora	12/02/2013	12/02/2013
Búsqueda de información sobre Big Data	1 día	12/02/2013	12/02/2013
Comparación de arquitecturas Big Data	1 día	13/02/2013	13/02/2013
Búsqueda y comparativa de soluciones	1 día	14/02/2013	14/02/2013
Búsqueda de herramientas Hadoop	3 días	15/02/2013	19/02/2013
Búsqueda de soluciones Hadoop	6 días	20/02/2013	27/02/2013
Comparativa de soluciones	1 día	28/02/2013	28/02/2013
Escoger propuestas de soluciones para los pilotos	1 día	01/03/2013	01/03/2013
Presentación de propuestas	1 hora	04/03/2013	04/03/2013
Revisión de la planificación	4 horas	04/03/2013	04/03/2013
Iteración 2 – Especificación pilotos y aprendizaje	22 días	05/03/2013	03/04/2013
Estudio de casos de uso	4 días	05/03/2013	08/03/2013
Especificación de los pilotos	10 días	11/03/2013	22/03/2013
Estudio y aprendizaje herramienta Flume	10 días	11/03/2013	22/03/2013
Documentar	7 días	25/03/2013	02/04/2013
Decisión de la infraestructura de los pilotos	1 hora	03/04/2013	03/04/2013
Revisión de la planificación	4 horas	03/04/2013	03/04/2013
Iteración 3 – Implementación arquitectura pilotos	23 días	04/04/2013	06/05/2013
Adquisición de hardware y licencias	10 días	04/04/2013	17/04/2013
Preparación de presentación al gerente	2 días	04/04/2013	05/04/2013
Presentación al gerente	1 hora	08/04/2013	08/04/2013
Implementación de la capa de recolección de datos	8 días	08/04/2013	17/04/2013
Instalación, familiarización y aprendizaje piloto A	12 días	18/04/2013	03/05/2013
Instalación, familiarización y aprendizaje piloto B	12 días	18/04/2013	03/05/2013
Reunión de seguimiento – Estado del proyecto	1 hora	06/05/2013	06/05/2013
Revisión de la planificación	4 horas	06/05/2013	06/05/2013
Iteración 4 – Implementación de los casos de uso	55 días	07/05/2013	22/07/2013
Implementación de caso de uso A	49 días	07/05/2013	12/07/2013
Implementación de caso de uso B	49 días	07/05/2013	12/07/2013
Valoración piloto A	5 días	15/07/2013	19/07/2013
Valoración piloto B	5 días	15/07/2013	19/07/2013
Presentación de las valoraciones	1 hora	22/07/2013	22/07/2013
Revisión de la planificación	4 horas	22/07/2013	22/07/2013
Iteración 5 – Análisis de resultados y conclusiones	7 días	23/07/2013	31/07/2013
Valoración del proyecto	6 días	23/05/2013	30/07/2013
Presentación de la valoración final	1 hora	31/05/2013	31/07/2013

Planificación

Revisión de proyecto	4 horas	31/07/2013	31/07/2013
----------------------	---------	------------	------------

Tabla 2: Lista de tareas de la planificación inicial

2.2. PLANIFICACIÓN FINAL

Al finalizar la iteración 2 y durante la realización de la tercera, se dieron varios factores que obligaron a reestructurar el proyecto. Desde la empresa nos comentaron que no era posible disponer de dos servidores, por lo que la realización de dos pilotos en paralelo no era viable. Además, tampoco se pudieron conseguir las licencias de software de manera que solamente se pudo trabajar con software gratuito. Hay que añadir que prácticamente todas las distribuciones estudiadas y gratuitas proponían soluciones sobre una misma base: Hadoop; y que las herramientas y funcionalidades extra de cada empresa se ofrecen sólo en las versiones de pago. Realizar dos pilotos sobre dos soluciones gratuitas sería como hacer pruebas y comparaciones con dos distribuciones Hadoop prácticamente iguales.

Por estas dos razones, se rechazó la propuesta de implementar dos soluciones distintas. Se cambió a sólo un piloto o caso de uso real y se amplió la beca hasta noviembre para realizar pruebas extensas sobre esa misma solución.

La planificación pasó a tener seis iteraciones, estando las dos primeras ya cerradas, y modificando las cuatro siguientes:

3. **Implementación del caso de uso real en local:** al no disponer de los servidores, en esta iteración se instala una máquina virtual en los ordenadores portátiles personales con una distribución gratuita (la que se utilizará para el caso de uso) para realizar la implementación de las aplicaciones del caso de uso.
4. **Diseño y realización de pruebas:** se diseña un plan de pruebas para que se puedan empezar cuando se disponga del servidor. Se hace la ejecución del conjunto de pruebas antes que el despliegue de la aplicación del caso de uso para familiarizarnos y para testear la aplicación, ya que algunas de las pruebas utilizan el código implementado en la iteración anterior.
5. **Despliegue del caso de uso:** se despliegan las aplicaciones del caso de uso sobre la distribución del servidor y se comprueba el correcto funcionamiento. Durante el periodo de ampliación del proyecto, surgió la posibilidad de implementar una pequeña aplicación a modo de caso de uso extra que nos sirvió para aumentar el estudio de la distribución y sacar conclusiones interesantes.
6. **Análisis de resultados y conclusiones:** conclusiones finales sobre todo el proyecto y presentación de los resultados obtenidos, tanto en el caso de uso como en la fase de pruebas.

En la Figura 2 se muestra de forma general la distribución de las seis iteraciones en la planificación del proyecto y, como en la planificación inicial, en la Tabla 3 se muestran de forma detallada las tareas que componen cada iteración.

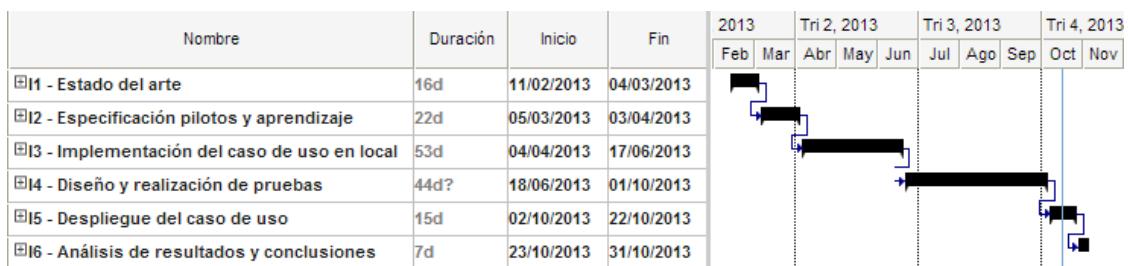


Figura 2: Vista general de la planificación final.

Planificación

Tarea	Duración	Inicio	Fin
Iteración 1 – Estado del arte	16 días	11/02/2013	04/03/2013
Contextualización del proyecto	1 día	11/02/2013	11/02/2013
Planificación inicial	1 hora	12/02/2013	12/02/2013
Búsqueda de información sobre Big Data	1 día	12/02/2013	12/02/2013
Comparación de arquitecturas Big Data	1 día	13/02/2013	13/02/2013
Búsqueda y comparativa de soluciones	1 día	14/02/2013	14/02/2013
Búsqueda de herramientas Hadoop	3 días	15/02/2013	19/02/2013
Búsqueda de soluciones Hadoop	6 días	20/02/2013	27/02/2013
Comparativa de soluciones	1 día	28/02/2013	28/02/2013
Escoger propuestas de soluciones para los pilotos	1 día	01/03/2013	01/03/2013
Presentación de propuestas	1 hora	04/03/2013	04/03/2013
Revisión de la planificación	4 horas	04/03/2013	04/03/2013
Iteración 2 – Especificación pilotos y aprendizaje	22 días	05/03/2013	03/04/2013
Estudio de casos de uso	4 días	05/03/2013	08/03/2013
Especificación de los pilotos	10 días	11/03/2013	22/03/2013
Estudio y aprendizaje herramienta Flume	10 días	11/03/2013	22/03/2013
Documentar	7 días	25/03/2013	02/04/2013
Decisión de la infraestructura de los pilotos	1 hora	03/04/2013	03/04/2013
Revisión de la planificación	4 horas	03/04/2013	03/04/2013
Iteración 3 – Implementación del caso de uso en local	53 días	04/04/2013	17/06/2013
Adquisición de hardware y licencias	10 días	04/04/2013	17/04/2013
Preparación de presentación al gerente	2 días	04/04/2013	05/04/2013
Presentación al gerente	1 hora	08/04/2013	08/04/2013
Implementación de la capa de recolección de datos	8 días	08/04/2013	17/04/2013
Reestructuración de la planificación	3 días	18/04/2013	22/04/2013
Instalación en local de Cloudera	9 días	23/04/2013	03/05/2013
Implementación caso de uso	30 días	06/05/2013	14/06/2013
Reunión de seguimiento – Estado del proyecto	1 hora	17/06/2013	17/06/2013
Revisión de la planificación	4 horas	17/06/2013	17/06/2013
Iteración 4 – Diseño y realización de pruebas	44 días	18/06/2013	30/09/2013
Diseño del plan de pruebas	4 días	18/06/2013	21/06/2013
Implementación de las pruebas	5 días	24/06/2013	28/06/2013
Adquisición del hardware e instalación	5 días	01/07/2013	05/07/2013
Ejecución de las pruebas	29 días	08/07/2013	30/09/2013
Sistema de ficheros HDFS	3 días	08/07/2013	10/07/2013
MapReduce + Mahout	5 días	11/07/2013	17/07/2013
YARN (MapReduce v2.0)	10 días	18/07/2013	31/07/2013
Hive/Impala	9 días	16/09/2013	26/09/2013
Flume	2 días	27/09/2013	30/09/2013
Reunión de seguimiento – Estado del proyecto	1 hora	01/10/2013	01/10/2013
Revisión de la planificación	4 horas	01/10/2013	01/10/2013
Iteración 5 – Despliegue del caso de uso	15 días	02/10/2013	22/10/2013
Implementación y despliegue caso de uso extra	3 días	02/10/2013	04/10/2013

Planificación				
Despliegue caso de uso principal	5 días	07/10/2013	11/10/2013	
Experimentación con el caso de uso	6 días	14/10/2013	21/10/2013	
Valoración del caso de uso	1 día	22/10/2013	22/10/2013	
Iteración 6 – Análisis de resultados y conclusiones	7 días	23/10/2013	31/10/2013	
Valoración del proyecto	6 días	23/10/2013	30/10/2013	
Presentación de la valoración final	1 hora	31/10/2013	31/10/2013	
Revisión de proyecto	4 horas	31/10/2013	31/10/2013	

Tabla 3: Lista de tareas de la planificación final

En la *Figura 54* del apartado *B. Planificación* del anexo se puede observar detalladamente un diagrama de Gantt con todas las tareas de las nuevas iteraciones. No se incluyen las iteraciones 1 y 2 ya que no han sido modificadas.

Clarificar que durante el mes de agosto y la primera quincena de septiembre el proyecto estuvo parado por vacaciones, razón por la que no hay tareas asignadas a ese periodo de tiempo y no se contabilizan los días en la planificación.

2.3. METODOLOGÍA DE TRABAJO

Durante este periodo de siete meses, mi compañero Jordi y yo hemos trabajado de forma paralela, delegándonos el trabajo para no realizar tareas de forma duplicada. En la planificación muchas tareas tienen como responsables a los dos pero la gran mayoría de ellas se han dividido en “*subtareas*” para avanzar más rápidamente. Estos deberes más pequeños no aparecen detallados ya que se ha querido generalizar un poco para facilitar la lectura de la planificación pero un ejemplo podría ser la repartición de las distintas soluciones Big Data a la hora de realizar el estudio teórico. Esta metodología de trabajo ha sido fácil de aplicar ya que teníamos contacto directo ya fuera en persona o por el sistema de comunicación interna de la empresa.

Los directores del proyecto han estado en contacto con nosotros día a día a través del sistema de comunicación Lync o por correo electrónico, además de realizar una reunión corta de seguimiento cada semana. Al finalizar cada iteración se ha realizado una reunión más extensa para planificar los siguientes pasos y para comprobar el trabajo realizado.

Para la realización del proyecto, la empresa proporcionó un ordenador portátil a cada uno en el que se ha realizado todo el trabajo, apoyado por un servidor dentro de la red local de la empresa para el despliegue y la ejecución de la solución Big Data.

2.4. PRESUPUESTO

Se ha separado el capítulo económico del proyecto en dos apartados: el coste de personal y el de material. Para cada uno de ellos se han desglosado los diferentes conceptos que han influido en la realización del proyecto. Hay que destacar que algunos de los conceptos -como el de recursos de la empresa- son aproximaciones ya que son datos confidenciales de la empresa y no se pueden facilitar.

- **Costes de personal (Tabla 4):** se incluyen las horas de dedicación personal dentro del horario laboral, tanto de Jordi como mías, y también las horas dedicadas por nuestros directores - Manuel Pando y José Roldán- a modo de tutoría, dirección, formación y soporte del proyecto. Además se incluye el trabajo realizado por el administrador de sistemas en su tarea de administrar y configurar la infraestructura.

Planificación

- **Costes materiales** (Tabla 5): engloba el alquiler de los portátiles y el coste de obtención y mantenimiento de los servidores, así como otros recursos que la empresa nos haya facilitado (como acceso a internet, licencias de software, material de oficina...).

	Meses	Coste mensual (€)	Subtotal (€)
Jordi Sabater	7	808,9	5.662,5
Robert Serrat	7	808,9	5.662,5
José Roldán	7	520	3.640
Manuel Pando	7	520	3.640
Administrador de sistemas	3	150	450
Total			19.055

Tabla 4: Costes de personal del proyecto.

	Meses	Coste mensual (€)	Subtotal (€)
Ordenadores portátiles (x2)	7	160	1.120
Recursos de la empresa (licencias de software, Internet, material de oficina...)	7	279	1.953
Servidores	4	300	1.200
Total			4.273

Tabla 5: Costes materiales del proyecto

De esta manera el coste total del proyecto es:

	Subtotal (€)
Costes de personal	19.055
Costes materiales	4.273
Total	23.328

Tabla 6: Coste total del proyecto

3. BIG DATA

En los últimos años la manera en la que los usuarios interactúan con la tecnología ha cambiado de manera radical debido a la constante evolución de ésta. Revoluciones tecnológicas como la web 2.0 - blogs, foros de opinión, redes sociales, multimedia...- y dispositivos como los teléfonos inteligentes facilitan la conectividad y la generación de grandes cantidades de información que hasta hace muy poco eran impensables. Y no solo la sociedad de consumo ha avanzado tecnológicamente; campos como la ciencia, medicina o la economía también requieren cada vez más tratar con grandes cantidades de datos.

Big Data es el sector de las tecnologías de la información y la comunicación (TIC) que se preocupa de como almacenar y tratar grandes cantidades de información o conjuntos de datos.

3.1. LAS CINCO V

Es común que cuando se hable de Big Data se haga referencia a grandes cantidades de datos. Pero es más que eso. Para describir mejor lo que representa, frecuentemente se habla de las cinco V -IBM fue la que empezó definiendo tres V y luego se han añadido las otras dos dependiendo de la fuente- que definen perfectamente los objetivos que este tipo de sistemas buscan conseguir: [2]

- **Volumen:** un sistema Big Data es capaz de almacenar una gran cantidad de datos mediante infraestructuras escalables y distribuidas. En los sistemas de almacenamiento actuales empiezan a aparecer problemas de rendimiento al tener cantidades de datos del orden de magnitud de petabytes o superiores. Big Data está pensado para trabajar con estos volúmenes de datos.
- **Velocidad:** una de las características más importantes es el tiempo de procesado y respuesta sobre estos grandes volúmenes de datos, obteniendo resultados en tiempo real y procesándolos en tiempos muy reducidos. Y no sólo se trata de procesar sino también de recibir, hoy en día las fuentes de datos pueden llegar a generar mucha información cada segundo, obligando al sistema receptor a tener la capacidad para almacenar dicha información de manera muy veloz.
- **Variedad:** las nuevas fuentes de datos proporcionan nuevos y distintos tipos y formatos de información a los ya conocidos hasta ahora -como datos no estructurados-, que un sistema Big Data es capaz de almacenar y procesar sin tener que realizar un preproceso para estructurar o indexar la información.
- **Variabilidad:** las tecnologías que componen una arquitectura Big Data deben ser flexibles a la hora de adaptarse a nuevos cambios en el formato de los datos -tanto en la obtención como en el almacenamiento- y su procesado. Se podría decir que la evolución es una constante en la tecnología de manera que los nuevos sistemas deben estar preparados para admitirlos.
- **Valor:** el objetivo final es generar valor de toda la información almacenada a través de distintos procesos de manera eficiente y con el coste más bajo posible.

De esta manera, un sistema Big Data debe extraer valor -en forma de nueva información, por ejemplo- sobre grandes volúmenes de datos, de la manera más rápida y eficiente posible, adaptándose a todos los formatos -estructurados o no- existentes y futuros.

3.2. TIPOS DE INFORMACIÓN

Se puede hablar de una clasificación de los tipos de datos según sea su naturaleza u origen que también ayuda a entender mejor el porqué de la evolución de los sistemas de explotación de la información hacia Big Data:

- **Datos estructurados:** es información ya procesada, filtrada y con un formato estructurado. Es el tipo de datos que más se usan hoy en día (sobre todo con bases de datos relacionales como en la *Tabla 7*).
- **Datos semi-estructurados:** es información procesada y con un formato definido pero no estructurado. De esta manera se puede tener la información definida pero con una estructura variable. Dos ejemplos son las bases de datos basadas en columnas y los ficheros con información en un lenguaje de etiquetas (HTML o XML como el de la Figura 3).
- **Datos no estructurados:** es información sin procesar y que puede tener cualquier estructura. Se puede encontrar cualquier formato: texto, imagen, vídeo, código, etc. Los directorios de *logs* de aplicaciones o la información colgada en las redes sociales son buenos ejemplos de datos no estructurados.

La manera de trabajar hoy en día implica almacenar solamente datos de tipo estructurado o semi-estructurado, obligando a pasar por un proceso de filtrado y transformación los datos no estructurados. Este proceso radica en un coste y en una pérdida inevitable de datos que cada vez es más difícil ignorar, ya que va totalmente en contra de las cinco V comentadas anteriormente y que un sistema de explotación de la información busca obtener -especialmente de la variedad, variabilidad y velocidad de recolección de información-.

Al implementar un proceso de transformación de información pierdes variabilidad ya que un cambio en el origen de datos también obligaría a cambiar dicho proceso. Además de un coste en el rendimiento de la recolección y en la pérdida de datos que si bien a priori puede parecer descartable, en un futuro puede ser necesaria.

Por estas razones una de las características principales de un sistema Big Data es el de trabajar también con los datos no estructurados, permitiendo aumentar la variedad y la variabilidad. De esta forma también se induce que el sistema debe poder almacenar y trabajar con un gran volumen de información.

Nombre	PrimerApellido	SegundoApellido	Teléfono	Mail
Alberto	Pérez	Ramiro	924875963	alberto@hotmail.com
Carola	Zapatero	Reyes	941236598	carola@gmail.com
María	Gallego	Serra	657832465	maria@gmail.com
Víctor	Abel	Cantalapiedra	917485264	victor@gmail.com
Cristian	Tome	Losada	952487139	cistrian@hotmail.com

Tabla 7: Tabla “Personas” de una base de datos relacional. Un ejemplo de información estructurada.

Big Data

```

<personas>
  <persona>
    <nombre orden="primero">María</nombre>
    <nombre orden="segundo">Mercedes</nombre>
    <apellido orden="primero">Gallego</apellido>
    <apellido orden="segundo">Serra</apellido>
    <nacionalidad>española</nacionalidad>
  </persona>
  <persona>
    <nombre>Victor</nombre>
    <apellido orden="primero">Abel</apellido>
    <apellido orden="segundo">Cantalapiedra</apellido>
    <nacionalidad>española</nacionalidad>
    <nacionalidad>italiana</nacionalidad>
  </persona>
</personas>

```

Figura 3: Ejemplo de un fichero XML con información de personas. Un ejemplo de información semi-estructurada. Los campos con la información de una persona están definidos pero pueden variar (la primera persona tiene dos nombres y la segunda dos nacionalidades).

3.3. ¿CUÁL ES EL PANORAMA ACTUAL DE BIG DATA?

Una de las primeras comunidades en trabajar con grandes cantidades de datos fue la científica. Aunque no trabajaran los aspectos Big Data como la variedad, variabilidad o velocidad, sí que lo hacían con grandes volúmenes de datos o buscando extraer valor a la información -en campos como por ejemplo los de la genética o la farmacia-. Hoy en día pero, ya hay multitud de terrenos que enfocan sus problemas al sector de Big Data como las redes sociales o los sistemas basados en sensores y datos generados por máquinas, como sistemas de *logs* o sistemas automatizados.

Algunos de los siguientes datos -así como los mostrados en la Figura 4- pueden servir para ilustrar mejor el concepto de grandes volúmenes de datos, la necesidad de procesarlos a gran velocidad y ser compatible con una gran variedad de formas y formatos: [3]

- Durante los experimentos del *Large Hadron Collider* (LHC) los científicos del CERN pueden llegar a generar 40 terabytes de datos por segundo.
- El motor de reacción de un *Boing* produce 10 terabytes de información cada treinta minutos. Esto quiere decir que durante un vuelo que cruce el atlántico genera hasta 640 terabytes de datos. Teniendo en cuenta la cantidad de vuelos que se hacen al día, en este caso se puede llegar a la conclusión de que volumen y velocidad van de la mano.



Figura 4: Volúmenes de datos generados por las distintas redes sociales cada minuto.

Big Data

- Twitter tiene aproximadamente unos 200 millones de usuarios que producen 90 millones de *tweets* al día -unos 1000 por segundo- de unos 200 bytes cada uno. Al día son 12 gigabytes pero el ecosistema de Twitter -tweets, fotos y otros contenidos- llega generar hasta 8 terabytes.
- Facebook, la red social por excelencia, ya ha superado los 750 millones de usuarios activos. Sus usuarios pasan más de 700 billones de minutos al mes en el servicio y crean, de media, 90 tipos de contenido cada treinta días. Estos tipos son cada vez más variados y entre los más usados están enlaces a otras webs, noticias, historias, videos o fotos. Es decir que la variedad en el tipo de contenido es muy alta, así como su volumen.

Con datos como estos es normal que las compañías empiecen a mirar Big Data con otros ojos y pongan su atención cada vez más en desarrollar sus propios sistemas para mejorar la productividad, puesto que los métodos más tradicionales de almacenamiento y análisis de datos están quedando obsoletos debido a los avances y cambios.

3.4. CASOS DE USO

La aparición de Big Data permite almacenar y tratar datos de una manera más eficiente, derivando en diversos casos de uso en los que su arquitectura puede ofrecer una solución eficiente. Muchos de estos casos de uso tienen como objetivo mejorar los procesos internos de las empresas, reduciendo su coste y tiempo a la vez que mejoran el resultado obtenido al reducir la cantidad de información descartada o tratada.

3.4.1. ANÁLISIS DE NEGOCIOS

Una de las principales razones de las compañías para desarrollar o usar tecnologías Big Data es la de mejorar sus procesos de negocios. Disciplinas como la relación con el cliente, la administración de capital o la toma de decisiones -política de precios, gestión de las TI, etc.- son las más beneficiadas. Gracias a esto, las compañías pueden incrementar sus beneficios, disminuir los costes o hacer un análisis de riesgos sobre las diferentes posiciones que pueden adoptar. [4]

Los casos de uso en los que las empresas se están centrándo actualmente para generar nuevo conocimiento a través de recopilar, consolidar y analizar datos son: [5]

- **Análisis personalizado de perfil según cliente o consumidor** mediante un historial de transacciones o actividades, ayudando a las empresas a clasificar de distintas maneras al cliente usando diferentes factores. Por ejemplo, decidir si es apto para recibir un crédito bancario o si le puede interesar un artículo específico o una línea de producto en general.
- **Análisis de transacciones** recibiendo constantemente datos en *streaming*.
- **Análisis social masivo**, demográfico o de mercado.
- **Descubrir nuevas oportunidades** de mercado mediante análisis de tendencias y comportamiento.
- **Reducción de riesgos:** analizando las tendencias de mercados potenciales y entendiendo las posibilidades que ofrecen se puede reducir el riesgo de posicionarse de una manera determinada, observando la cartera de inversiones y el retorno de capital.

3.4.2. ANÁLISIS DE SENTIMIENTO

Hasta hace pocos años, cuando una compañía quería conocer cuál era la opinión que los usuarios tenían sobre ella, sus productos o sus servicios tenían que recurrir a encuestas y/o formularios, realizar grandes estudios de investigación o delegar el trabajo sobre terceros -como una agencia-. Este proceso podía derivar en un estudio alterado debido a una selección de la población y una posible influencia sobre ella,

Big Data

aparte de tener un alto coste en presupuesto, esfuerzo y tiempo. Hoy en día, con la aparición de las redes sociales -como Twitter o Facebook- es más fácil obtener la opinión de los consumidores, obteniendo una valoración más natural y espontánea.

Este análisis no solo se basa en la opinión general de la población sobre un elemento, sino que se puede inducir mucha más información, respondiendo todo tipo de preguntas. ¿Cuál es el perfil de la población contenta/descontenta con un producto en concreto? ¿A qué suceso es debida la variación de la opinión en un espacio temporal concreto? Estas son algunas de las preguntas que se pueden responder de manera sencilla y rápida con Big Data.

3.4.3. SEGURIDAD

Uno de los casos de uso más comunes y que las nuevas tecnologías Big Data han facilitado y mejorado su práctica es el de seguridad. Cada vez hay más empresas que utilizan estas tecnologías para implementar soluciones para este campo. Un análisis completo de seguridad puede llegar a involucrar muchas fuentes de información como *logs* (de servidores o web, por ejemplo), transacciones con los clientes o distintos sensores. Las tecnologías Big Data permiten almacenar todo el contenido –no estructurado- ofrecido por estas fuentes sin tener que hacer una preselección o cualquier tipo de proceso. Luego pueden distribuir el análisis de la información en distintos nodos agilizando el proceso y obteniendo un resultado de manera más rápida que con los procedimientos habituales hasta ahora.

Algunos de los ejemplos más específicos son la detección de ataques ciberneticos -como malware en el sistema o *phishing*- en, por ejemplo, entidades bancarias para detectar intentos de fraude [6] o localizar a una persona o su teléfono mediante la información generada por las torres de comunicación inalámbrica -a través de sus llamadas o conexiones a internet-. [7]

3.4.4. USOS MÉDICOS Y CIENTÍFICOS

Big data también ofrece nuevas oportunidades para la comunidad científica y, en especial, a la médica.

El análisis de grandes cantidades de datos generados en *streaming* de eventos originados desde sensores, RFID o dispositivos médicos se puede implementar de manera sencilla, permitiendo el procesado de eventos complejos y la monitorización de varios pacientes en tiempo real.

La facilidad en la escalabilidad y su bajo coste permite tener grandes cantidades de datos, eliminando restricciones de espacio, permitiendo el mantenimiento de datos históricos para poder hacer seguimientos más personalizados para cada paciente o llevar a termo estudios más amplios. En especial datos relacionados con la genética, el metabolismo o las proteínas generan grandes cantidades de datos no estructurados que serían difíciles de almacenar en bases de datos relacionales. [8]

Un último caso de uso para arquitecturas Big Data sería el estudio de la población desde un punto de vista científico, pudiendo identificar posibles causas de infecciones -buscando patrones con altas tasas de infección-, redistribuir los recursos entre la población -por ejemplo, de 100.000 pacientes solo un 1% tenía el 30% de los recursos [9]- o hacer predicciones de futuras pandemias. [10]

3.5. ARQUITECTURA BIG DATA

La arquitectura Big Data está compuesta generalmente por cinco capas: recolección de datos, almacenamiento, procesamiento de datos, visualización y administración. Esta arquitectura no es nueva, sino que ya es algo generalizado en las soluciones de Business Intelligence que existen hoy en día. Sin embargo, debido a las nuevas necesidades cada uno de estos pasos ha ido adaptándose y aportando nuevas tecnologías a la vez que abriendo nuevas oportunidades.

Big Data

En la *Figura 5* se puede observar el flujo que la información tendría en una arquitectura Big Data, con orígenes de datos diversos -bases de datos, documentos o datos recibidos en *streaming*- que se reciben y almacenan a través de la capa de recolección de datos, con herramientas específicamente desarrolladas para tal función. Los datos recibidos pueden procesarse, analizarse y/o visualizarse tantas veces como haga falta y lo requiera el caso de uso específico.

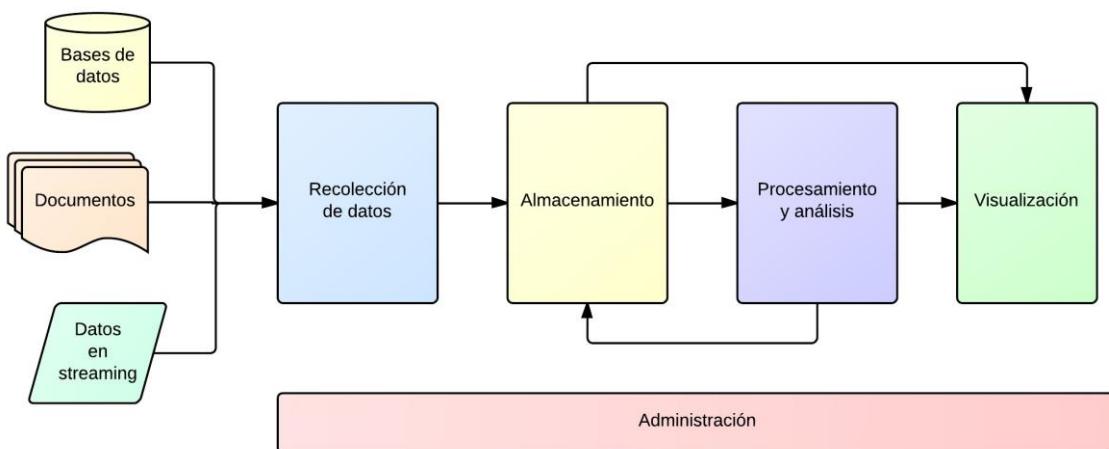


Figura 5: Arquitectura Big Data por capas.

3.5.1. RECOLECCIÓN DE DATOS

En esta etapa el sistema debe conectarse a sus fuentes de información y extraer la información. Las herramientas de recolección de datos pueden dividirse en dos grupos, dependiendo de cómo se conecten al origen de los datos:

1. **Batch o por lotes:** se conectan de manera periódica a la fuente de datos buscando nueva información. Generalmente se usan para conectarse a sistemas de ficheros o bases de datos, buscando cambios desde la última vez que se conectaron. Una herramienta para migrar datos periódicamente -una vez al día, por ejemplo- de una base de datos a otra es un ejemplo de recolección de datos por lotes.
2. **Streaming o por transmisión en tiempo real:** están conectados de manera continua a la fuente de datos, descargando información cada vez que ésta transmite. Se acostumbra a usar para monitorización de sistemas -para aumentar la seguridad y la detección de fallos-, de conjuntos de sensores o para conectarse a redes sociales y descargar información en tiempo real.

Actualmente las herramientas han evolucionado de manera que muchas de ellas ya pueden usarse de las dos maneras, tanto como para descargarse información en *streaming* como con procesos *batch*.

En esta etapa, los datos pueden sufrir algún tipo de proceso o cambio si la aplicación así lo requiere, por ejemplo el filtrado de información no deseada o el formateo con el que se guardará finalmente en el sistema de almacenamiento.

3.5.2. ALMACENAMIENTO

La capa de almacenamiento tiene, a grandes rasgos, dos elementos básicos: el sistema de ficheros y la base de datos. Hasta hace poco los sistemas de tratamiento de la información se centraban principalmente en las bases de datos pero, debido a que en los sistemas Big Data se busca la mayor variedad posible -las bases de datos acostumbran a ser poco flexibles-, los sistemas de ficheros han cobrado mayor importancia.

3.5.2.1. SISTEMAS DE FICHEROS Y SISTEMAS DE FICHEROS DISTRIBUIDOS

Los sistemas de ficheros son una parte fundamental de la arquitectura Big Data ya que es por encima de ellos que el resto de herramientas están construidas. Además, el hecho de querer trabajar con datos no estructurados los hace aún más importante ya que son el medio principal para trabajar con este tipo de información.

Adicionalmente, un objetivo que buscan los sistemas Big Data es la **escalabilidad**, es decir, un sistema que pueda variar su tamaño (ya sea aumentándolo o disminuyéndolo) según las necesidades y que esto no afecte al rendimiento general de todo el sistema. Esta necesidad fue la que motivó la aparición de los sistemas de ficheros distribuidos, que consisten en una red o clúster de ordenadores -o nodos- interconectados que están configurados para tener un sólo sistema de ficheros lógico.

En la *Figura 6* se puede observar un ejemplo simplificado del funcionamiento de un sistema de ficheros distribuido. Se tiene un directorio con cuatro ficheros -FicheroA, FicheroB, FicheroC y FicheroD- que el usuario, al conectarse al sistema y entrar en el directorio, ve como si estuvieran todos almacenados en un mismo ordenador (sistema lógico). La realidad es que cada fichero está físicamente almacenado en un nodo u ordenador distinto a los demás (sistema físico).

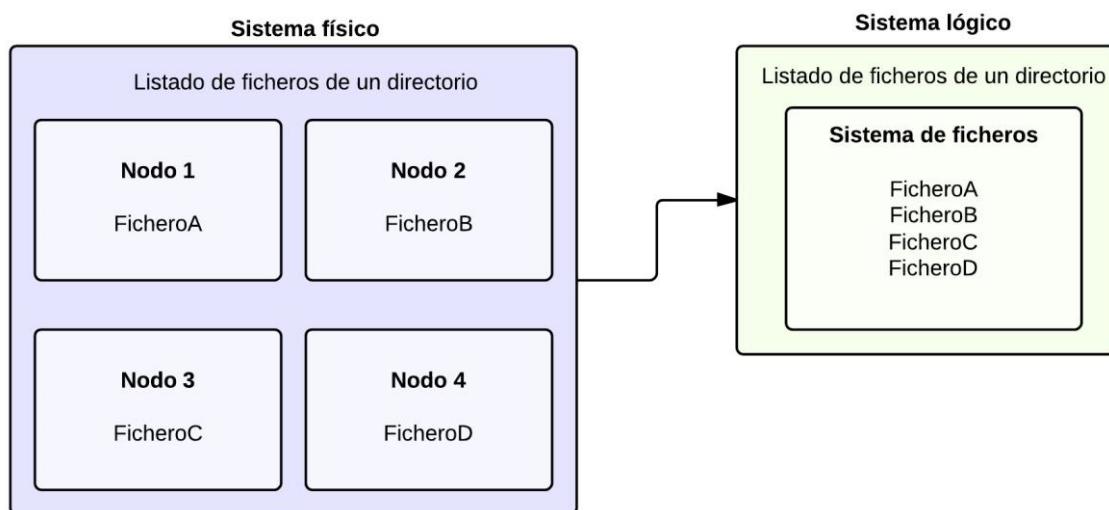


Figura 6: Vistas física y lógica de un sistema de ficheros distribuido con cuatro ficheros: FicheroA, FicheroB, FicheroC y FicheroD.

3.5.2.2. BASES DE DATOS

Las bases de datos siempre han tenido mucha presencia en los sistemas de explotación de la información, especialmente las bases de datos relacionales, que se establecieron como un nuevo paradigma en los años 70 y que se consolidaron gracias a la facilidad de conceptualizar los problemas. Estas bases de datos siguen el modelo relacional que permite interconectar los datos –almacenados en tablas, como la de la *Tabla 7*- mediante relaciones que permiten relacionar los datos de distintas tablas. Un ejemplo de estas relaciones lo podemos observar en la *Tabla 8*, que usa el campo Teléfono para relacionar sus datos con los de la *Tabla 7*.

Big Data

Teléfono	Compañía	Tarifa	Coste mensual (€)	Minutos gratis al mes
924875963	Movistar	Tarifa plana interprovincial	35	500
941236598	Orange	León	25	250
657832465	Vodafone	Prepago	-	0
917485264	Amena	En casa	15	100
952487139	Digitel	Radical	10	0

Tabla 8: Tabla “Contratos” de una base de datos relacional con información de tarifas contratadas que usa el campo teléfono para relacionar las tarifas con los datos de personas de la Tabla 7.

Con la aparición de las bases de datos relacionales también surgió SQL (*Structured Query Language*) una especificación de lenguajes para trabajar con estas bases de datos –en la actualidad hay muchos lenguajes basados en SQL como PostgreSQL, MySQL, MariaDB o SQLite; que a la vez también son Sistemas Gestores de Bases de Datos Relacionales o SGBDR-. Los lenguajes SQL se benefician de consultas muy sencillas, parecidas al lenguaje humano, que las hacen muy accesibles a los usuarios no expertos. Además, se aprovecha de las características del álgebra y el cálculo relacional para efectuar con el fin de recuperar de forma sencilla información de interés.

```
SELECT p.Nombre, p.PrimerApellido, p.Teléfono, c.Compañía, c.Tarifa
FROM Personas p, Contratos c
WHERE p.Teléfono == c.Teléfono
```

Figura 7: Consulta SQL que lista para cada contrato de la Tabla 8 el nombre y el primer apellido de la persona que lo ha contratado.

p.Nombre	p.PrimerApellido	p.Teléfono	c.Compañía	c.Tarifa
Alberto	Pérez	924875963	Movistar	Tarifa plana interprovincial
Carola	Zapatero	941236598	Orange	León
María	Gallego	657832465	Vodafone	Prepago
Víctor	Abel	917485264	Amena	En casa
Cristian	Tome	952487139	Digitel	Radical

Tabla 9: Resultado de la consulta SQL de la Figura 7.

A pesar de ser sistemas muy fáciles de usar y rápidos en la ejecución de las consultas –gracias a la creación de índices- los SGBD relacionales tradicionales tienen ciertos hándicaps que hacen que estén empezando a perder rendimiento con los problemas Big Data. Son sistemas rápidos y ágiles pero cuando la información almacenada supera unos límites -normalmente alrededor de terabytes- mantener la información estructurada tiene un coste en la creación y mantenimiento de los índices y en el rendimiento de las consultas. Además son bases de datos poco flexibles ya que cuando se crea su estructura es bastante conflictivo realizar cambios en esta (como añadir nuevas columnas a una tabla o cambiar el tipo de una columna).

Debido a estos problemas la popularidad de las llamadas bases de datos NoSQL o Not-only SQL aumentó al surgir los primeros brotes de los problemas Big Data: las empresas que basan su actividad en Internet y las redes sociales. No es de extrañar pues que las principales compañías que promueven estas bases de datos sean Amazon, Google, Facebook o Twitter. Son modelos de bases de datos que no siguen el modelo relacional -y por lo tanto no usan lenguajes SQL- y aportan más flexibilidad al no requerir estructuras fijas como las tablas. Otras ventajas de estos sistemas es que responden a las necesidades de escalabilidad, ya que al no tener que mantener los índices para los datos el volumen de información

Big Data

que almacenan siempre crece de forma horizontal (en las bases de datos SQL el mantenimiento de índices hace que crezcan de manera parecida a exponencial al añadir nuevos datos). Algunos ejemplos de sistemas NoSQL son MongoDB (basado en ficheros JSON), Riak (basado en el modelo clave-valor), eXist (ficheros XML) o BigTable de Google (basado en columnas).

Independientemente del paradigma de base de datos que sea, cada vez es más frecuente que los sistemas se adapten para funcionar con los sistemas distribuidos, obteniendo una mayor escalabilidad. Los sistemas NoSQL acostumbran a ser en este sentido más adaptables a los sistemas distribuidos, permitiendo una mayor flexibilidad en la configuración de las máquinas (con hardware más modesto, por ejemplo); mientras que los sistemas SQL requieren infraestructuras más especializadas (y a la vez más costosas) en trabajar con los SGBD relacionales.

En la *Tabla 10* se puede observar los principales puntos fuertes y débiles de los sistemas SQL y los sistemas NoSQL explicados hasta ahora (siempre desde el punto de vista Big Data) y de donde se puede entender que cada modelo se puede usar en distintos casos. El modelo relacional con SQL es más óptimo en casos con menos volumen de información, fáciles de conceptualizar y con la necesidad de obtener un tiempo de respuesta reducido. Por su parte, los sistemas NoSQL son más adecuados para ocasiones donde se necesita tener un volumen de datos mayor, una escalabilidad horizontal y más flexibilidad y variedad en los tipos de datos.

SQL		NoSQL	
Ventajas	Desventajas	Ventajas	Desventajas
Velocidad	Escalabilidad no horizontal	Mayor volumen	Más lento
Facilidad de uso	Requiere de hardware especializado	Más variabilidad	Más complejo
		Más variedad	
		Flexible en la configuración del hardware	

Tabla 10: Comparativa entre SQL y NoSQL.

En las arquitecturas Big Data más recientes se está intentando aprovechar lo mejor de los dos paradigmas. Se crea un sistema de almacenamiento (ya sea un sistema de ficheros distribuido o una base de datos NoSQL) para almacenar la información no estructurada en grandes volúmenes de datos y, a posteriori, se almacenan los resultados de los procesos y análisis realizados sobre estos datos en un sistema SQL, obteniendo una mayor velocidad de respuesta al consultar los resultados.

3.5.3. PROCESAMIENTO Y ANÁLISIS

Una vez se tienen los datos almacenados, el siguiente paso en un sistema Big Data es explotar la información para llegar a los resultados deseados. Las herramientas de análisis y procesamiento de información han evolucionado considerablemente, especialmente aquellas que trabajan sobre datos no estructurados. La necesidad de crear nuevas aplicaciones y que éstas ya estén adaptadas a los sistemas de almacenamiento más recientes (como los comentados en el punto anterior, los sistemas distribuidos y las bases de datos NoSQL) ha promovido la aparición de nuevos paradigmas.

En el apartado *3.6 Paradigmas Big Data* se habla con más detalle de estos paradigmas y se hace una comparación teórica, ya que en las fases de pruebas técnicas y de implementación de un caso de uso se trabajará con una solución e infraestructuras que implementen el paradigma escogido.

3.5.4. VISUALIZACIÓN

El apartado de visualización es el que menos ha cambiado respecto a las arquitecturas más tradicionales. Como se ha comentado en el apartado de *Almacenamiento*, los resultados a visualizar del procesamiento se acostumbran a consultar sobre bases de datos relacionales o SQL, ya que son las que ofrecen un menor tiempo de respuesta.

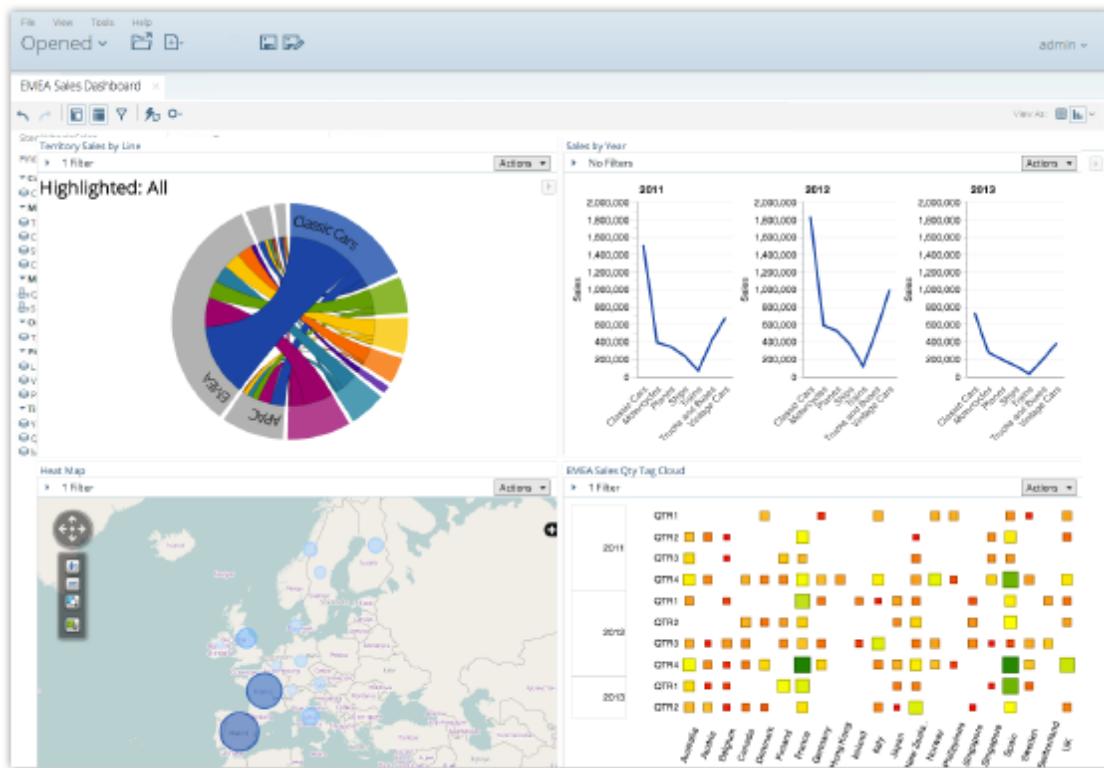


Ilustración 1: Pantalla de Pentaho, herramienta de visualización. [11]

3.6. PARADIGMAS BIG DATA

La aparición del concepto Big Data ha propiciado la creación de varios paradigmas de programación para el proceso de datos que intentan ofrecer un acercamiento a una solución para Big Data. Estos paradigmas han terminado caracterizando las arquitecturas Big Data, adaptando el resto de capas para funcionar de forma óptima. Los dos paradigmas que centran el desarrollo de aplicaciones son MapReduce y las llamadas Massive Parallel Processing (o MPP), ambas con aspectos en común pero bien diferenciadas.

El objetivo de este apartado es el de descubrir cuáles son las características principales de cada una así como los puntos fuertes y los principales hándicaps, para finalmente poder hacer una comparativa y escoger sobre qué paradigma se centrarán los estudios teórico y técnico.

3.6.1. MAPREDUCE

MapReduce es un modelo de programación introducido por Google y que en su evolución han participado decenas de colaboradores, apareciendo multitudes de implementaciones. De entre todas esas implementaciones destaca especialmente Hadoop, un proyecto de Apache para proporcionar una base sólida a las arquitecturas y herramientas Big Data. [12]

Big Data

El objetivo de MapReduce es el de mejorar el procesamiento de grandes volúmenes de datos en sistemas distribuidos y está especialmente pensado para tratar ficheros de gran tamaño -del orden de gigabytes y terabytes-. También mejora el tratamiento de los datos no estructurados, ya que trabaja a nivel de sistema de ficheros.

El nombre viene dado por la arquitectura del modelo, dividida principalmente en dos fases que se ejecutan en una infraestructura formada por varios nodos, formando un sistema distribuido, y que procede de la siguiente manera:

- **Map:** uno de los nodos, con el rango de “*master*”, se encarga de dividir los datos de entrada (uno o varios ficheros de gran tamaño) en varios bloques a ser tratados en paralelo por los nodos de tipo “*worker map*”. Cada bloque es procesado independientemente del resto por un proceso que ejecuta una función map. Esta función tiene el objetivo de realizar el procesamiento de los datos y dejar los resultados en una lista de pares *clave-valor* (es decir, se encarga de “mapear” los datos).

$$\text{Map}(k_1, v_1) \rightarrow \text{list}(k_2, v_2)$$

- **Reduce:** los nodos worker de tipo reduce ejecutan una función reduce que recibe como entrada una de las claves generadas en la etapa de map junto con una lista de los valores correspondientes a esa clave. Como salida genera una lista resultante de una función con los valores recibidos. La unión de los resultados puede corresponder a cualquier tipo de función (agregación, suma, máximo, etc.).

$$\text{Reduce}(k_2, \text{list}(v_2)) \rightarrow \text{list}(v_3)$$

Entre las fases de map y reduce hay una etapa intermedia llamada *agregación*, cuyo objetivo es el de agregar y ordenar las salidas de los distintos maps para que en la etapa de reduce se reciba solo la estructura del tipo “*clave-valores*” comentada. Un ejemplo sencillo de un proceso MapReduce es el de la *Figura 8*, que cuenta el número de palabras en un fragmento de *Don Quijote de la Mancha*.

Este modelo de programación parece poco flexible a la hora de intentar acercarse a diferentes tipos de problemas ya que está limitado por la propia arquitectura del modelo de programación MapReduce. Por el contrario, ofrece multitud de ventajas como el poder trabajar sobre cualquier formato de datos (estructurados, semi-estructurados o no estructurados), no tener requisitos de hardware elevados, ser escalable a nivel de tamaño de clúster y de rendimiento (añadir o quitar un nodo es sencillo y afecta directamente a la productividad del sistema) y funciona bien con grandes cantidades de datos ya que trabaja en bloques independientes del tamaño. Por el contrario la velocidad de procesado no es tan rápida como la de una base de datos ya que tiene que ejecutar varios procesos que encarecen los tiempos de ejecución.

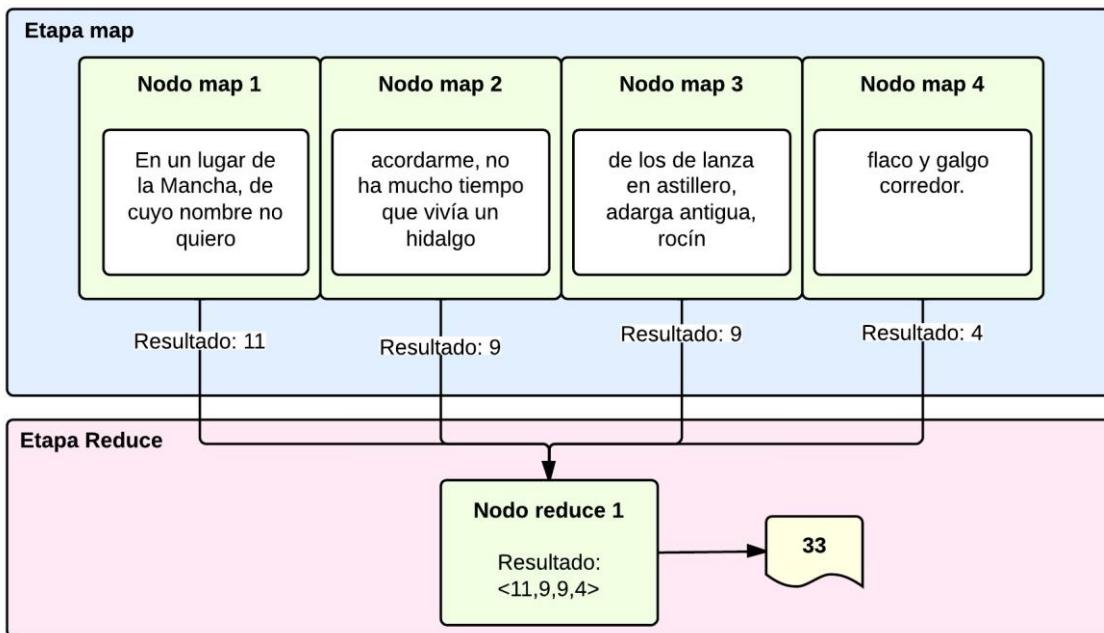


Figura 8: Ejemplo de una ejecución map-reduce para un proceso de contar palabras en una arquitectura con cuatro nodos. En este caso la salida clave-valor de cada nodo tendría como clave “Resultado” y como valor el número de palabras de su bloque. Tras pasar por la agregación, la etapa de reduce recibiría como entrada “Resultado: <11, 9, 9, 4>” y tendría por resultado “33”.

En general, la idea de MapReduce se podría resumir en una frase: es más fácil mover la lógica del proceso a los datos, que los datos al procesado.

3.6.2. MASSIVE PARALLEL PROCESSING (MPP)

El paradigma MPP intenta ofrecer una solución más tradicional a los problemas Big Data. Cuando se habla de MPP en sistemas de explotación de la información se hace referencia a la adaptación de las bases de datos relacionales a un sistema distribuido. El concepto es muy parecido al de MapReduce, pues trata sobre paralelizar las consultas en diferentes nodos, pero cuenta con algunas diferencias notables. Las MPP generalmente trabajan sobre bases de datos relacionales, lo que quiere decir que están construidas sobre modelos relacionales y usan lenguajes de consultas SQL en lugar de los lenguajes de programación como Java o Python de MapReduce.

Al contrario que pasa con MapReduce, el objetivo de las MPP es mantener la velocidad de procesado en las consultas y, a la vez, aumentar la cantidad de datos con la que pueden trabajar. De esta manera, el paradigma MPP sigue sin ser una buena solución para Big Data desde los puntos de vista de la variedad y la variabilidad, pues requiere de datos estructurados. En cambio solventa eficientemente el aspecto del volumen de datos al ser sistemas distribuidos y mantienen la velocidad característica de los SGBDR.

Un punto privativo de las MPP es que este tipo de soluciones acostumbran a venderse en *appliances* –es decir, se vende la licencia del software junto con los servidores–, siendo mucho más caras de lo habitual si se adquiriera el software y se montara el servidor por cuenta propia.

3.6.3. COMPARATIVA

Para la comparativa entre los dos paradigmas se ha creado una tabla (*Tabla 11*) con las valoraciones de distintos aspectos que se consideran importantes en un sistema Big Data, ya que se tratan de los puntos en los que una solución debería destacar –básicamente se trata de lo visto en el apartado 3.1. *Las cinco V-*. Las valoraciones van del 1 (peor) a 3 (mejor). Al ser una comparación teórica no se pueden hacer valoraciones más allá de lo conocido. Los aspectos valorados han sido los siguientes:

Big Data

- **Velocidad:** los tiempos de respuesta en las operaciones, procesos y consultas a realizar. Los baremos de valoración para este aspecto son:
 - **1:** los procesos tardan del orden de horas.
 - **2:** los procesos tardan del orden de minutos.
 - **3:** los procesos tardan del orden de segundos.
- **Volumen:** cuál es el volumen de datos con el que cada sistema puede llegar a trabajar. Los baremos de valoración para este aspecto son:
 - **1:** no permite trabajar con volúmenes de datos del orden de los gigabytes.
 - **2:** no permite trabajar con volúmenes de datos del orden de los terabytes.
 - **3:** no tiene problemas en trabajar con volúmenes del orden de zettabytes.
- **Escalabilidad:** la facilidad y la influencia de escalar la infraestructura. Los baremos de valoración para este aspecto son:
 - **1:** no son escalables en marcha, hay que parar el sistema y reconfigurar todo.
 - **2:** son escalables sin tener que reconfigurar ni parar ningún nodo pero requiere de una especificación de hardware mínima.
 - **3:** son escalables independientemente del hardware que se esté añadiendo.
- **Variedad:** con qué y con cuántos tipos de datos puede trabajar el sistema. Los baremos de valoración para este aspecto son:
 - **1:** solo trabaja con datos estructurados.
 - **2:** admite datos semi estructurados.
 - **3:** puede trabajar con datos no estructurados.
- **Variabilidad:** cómo reacciona un sistema a un cambio en los orígenes de datos o en el significado de su información. Los baremos de valoración para este aspecto son:
 - **1:** no permite modificar el esquema creado desde el inicio.
 - **2:** permite variar o ampliar el esquema.
 - **3:** es independiente de esquema de datos.
- **Productividad:** qué nivel de productividad puede llegar a tener un sistema teniendo en cuenta las tecnologías que usa, su implantación actual, etc. Los baremos de valoración para este aspecto son:
 - **1:** son tecnologías nuevas y en constante evolución, que implican un proceso de aprendizaje y una manera de trabajar distinta.
 - **2:** tecnologías nuevas pero con una curva de aprendizaje correcta y una manera de trabajar parecida a la ya existente.
 - **3:** a pesar de ser tecnologías nuevas son totalmente compatibles con la forma de trabajar actuales -lenguajes de programación, consulta, herramientas de terceros, etc.-.
- **Coste:** el precio tanto en mantenimiento como en la adquisición de la infraestructura y las licencias de software. Los baremos de valoración para este aspecto son:
 - **1:** las soluciones basadas en este paradigma suelen ser costosas y con un mantenimiento alto.
 - **2:** sus soluciones son caras pero tienen un mantenimiento fácil.
 - **3:** tiene soluciones flexibles, que se ajustan a los presupuestos, y no requieren demasiado mantenimiento.

	MapReduce	MPP
Velocidad	2	3
Volumen	3	2
Escalabilidad	3	2
Variedad	3	1
Variabilidad	3	1
Productividad	1	3
Coste	3	1
Total	18	13

Tabla 11: Comparativa entre los paradigmas MapReduce y MPP.

- **Velocidad:** los sistemas MPP son indiscutiblemente más rápidos -del orden de varios segundos en consultas sencillas- al tener los datos ya estructurados y preparados para la consulta mediante índices. De todas formas, las arquitecturas MapReduce ofrecen un rendimiento escalable linealmente, de manera que aumentando el número de nodos aumentaría también la velocidad, por lo que al tratar grandes volúmenes de datos podría llegar a igualar a los sistemas MPP -de todas formas, no baja del orden de unos pocos minutos en consultas sencillas-.
- **Volumen:** las MPP están penalizadas por la creación de índices, que ocupan espacio y además limitan el crecimiento, un problema heredado de los SGBD relacionales. De todas formas al ser sistemas escalables, puede que no aumenten el rendimiento como MapReduce (que no cuenta con esta limitación) pero sí que permiten una gran cantidad de datos. Las soluciones MPP empiezan a ser sistemas limitados a partir de volúmenes de datos superiores al orden de los gigabytes y terabytes, mientras que MapReduce puede llegar a tratar volúmenes de zettabytes.
- **Escalabilidad:** los sistemas MapReduce son escalables a todos los efectos. Se puede incrementar el número de nodos incluso añadiendo nuevos con distintas especificaciones (la ejecución de cada nodo es independiente de las demás). En el caso de las MPP, se requiere unas especificaciones más exigentes y uniformes, además de que añadir un nodo implica la reorganización de los metadatos o índices.
- **Variedad:** al aceptar datos no estructurados, las arquitecturas MapReduce admiten una mayor variedad de formatos para los datos. Las MPP en cambio requieren de datos estructurados.
- **Variabilidad:** tal y como pasa con la variedad, MapReduce está preparado para aceptar cambios de cualquier tipo, mientras que las MPP están ligadas al modelo de datos que se diseña al crear la base de datos, dificultando la modificación a posteriori.
- **Productividad:** las MPP son bases de datos relacionales que llevan usándose durante décadas, por lo que la productividad es mayor al no tener el usuario que aprender nuevos lenguajes (además de ser un lenguaje muy sencillo). MapReduce en cambio cuenta con un tiempo de aprendizaje y adaptación más amplio, ya que el usuario debe aprender un nuevo modelo de programación y acostumbrar a enfocar y atacar los problemas con esta arquitectura.
- **Coste:** MapReduce cuenta con un coste bastante menor que una solución MPP estándar. Hay muchas soluciones *open source* que se pueden instalar y ejecutar en infraestructuras de bajo presupuesto mientras que la totalidad de las soluciones MPP son de pago y normalmente se requiere un hardware más costoso (normalmente las soluciones MPP ya vienen con su propia infraestructura, ofrecida por la propia empresa y que acostumbra a tener un precio más elevado).

La valoración teórica final se decanta favorablemente hacia el lado de MapReduce, ya que su diseño está pensado exclusivamente para tratar grandes volúmenes de datos no estructurados en arquitecturas

Big Data

sencillas. Además tampoco acumula ciertas limitaciones de los sistemas tradicionales. También hay que añadir que el estudio de una arquitectura MapReduce es mucho más enriquecedora ya que es mayoritariamente desconocida, mientras que las MPP están basadas en tecnologías ya conocidas y que a nivel de usuario no aportan tantas novedades.

Por todas estas razones se decidió estudiar las distribuciones Hadoop, ya que es la implementación MapReduce con más soporte y que más éxito ha tenido en el mercado. En el apartado 4. *Hadoop* se estudia el funcionamiento y la estructura de la base que aporta el proyecto de Apache, en 5. *Herramientas Hadoop* se habla de las herramientas que complementan y agrandan las funcionalidades de Hadoop y, finalmente, en 6. *Distribuciones Hadoop* se estudian y comparan las distintas distribuciones Hadoop que hay en el mercado; con el fin de decidir cuál será la que se use para realizar la fase de pruebas y la implementación del caso de uso.

3.6.4. ARQUITECTURA MIXTA

A pesar de que de la comparativa se puede sacar la conclusión de que un sistema MapReduce es más adecuado para un problema Big Data que una arquitectura MPP, todo depende del caso de uso al que se haga frente. En realidad ambos paradigmas ofrecen debilidades: MapReduce no tiene la velocidad de un sistema de bases de datos relacional y las MPP no se adaptan tan bien a los datos no estructurados, además de no tener una escalabilidad demasiado fuerte.

En realidad, los puntos fuertes de ambos paradigmas se complementan muy bien y las soluciones que han aparecido los últimos meses han sabido aprovechar esta complementación, creando sistemas mixtos con ambas tecnologías implementadas.

La idea de una arquitectura mixta es la de usar un sistema de almacenamiento distribuido junto a MapReduce para procesar y analizar grandes cantidades de datos no estructurados, guardando los resultados de este análisis en arquitecturas MPP, que ofrecen una alta velocidad en la consulta de datos ya estructurados y de menor volumen, como sería la información previamente procesada por MapReduce.

De esta forma se consigue un sistema que acepta un gran volumen de datos, ya sean estructurados o no, con una buena capacidad para su procesamiento y una alta velocidad en la visualización.

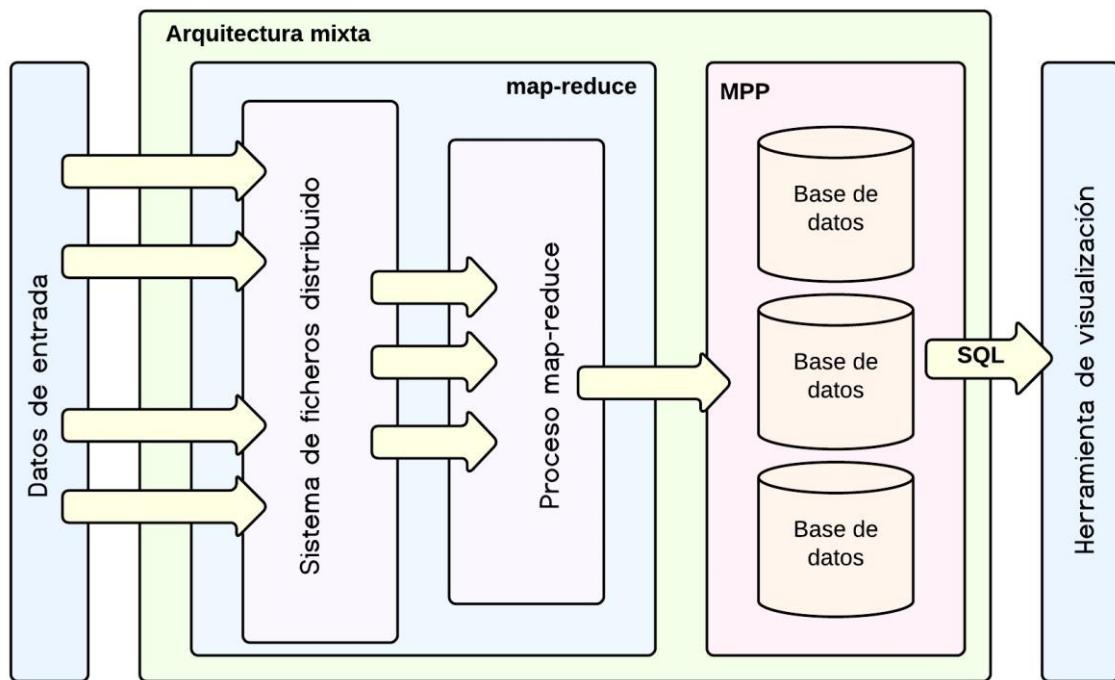


Figura 9: Arquitectura mixta formada por una solución MapReduce para procesar y filtrar datos no estructurados y una solución MPP para visualizar los resultar mediante conectores SQL en una herramienta de visualización.

3.7. SOLUCIONES

Una vez decidida la solución que se estudiará en el proyecto, toca escoger el tipo de solución que se usará para realizar las pruebas y la implementación del caso de uso. Para hacer la toma de decisiones se tuvo en cuenta tres tipos de soluciones que se pueden conseguir para Hadoop: una basada en distribuciones, las appliances que ofrecen las compañías y las infraestructuras en cloud como servicio.

3.7.1. DISTRIBUCIONES

Las soluciones basadas en distribuciones no son más que clústeres configurados según las necesidades del usuario para instalarle una distribución Hadoop. El usuario es el encargado de encontrar las especificaciones que deben cumplir los servidores y también de la instalación y configuración de la solución. La principal ventaja de este tipo de soluciones es la flexibilidad a la hora de elegir las especificaciones para que estén dentro de unos límites presupuestarios y que a la vez cumplan con los requisitos del proyecto. También permiten mucha flexibilidad a la hora de realizar modificaciones en la infraestructura y un mayor abanico de opciones a la hora de montar una solución (a nivel de programario).

Por el contrario, requieren de un conocimiento mínimo en administración de sistemas tanto para encontrar las especificaciones adecuadas como para el mantenimiento de las máquinas. También se requiere un conocimiento previo en distribuciones Hadoop para poder realizar la instalación y configuración del software.

3.7.2. APPLIANCE

Las appliances son infraestructuras vendidas por compañías distribuidoras de soluciones Hadoop que también se encargan normalmente de su instalación y servicio técnico. Tienen un coste bastante más elevado ya que también se añade en el paquete el servicio técnico y de soporte, además de las licencias y una infraestructura generalmente con unos requisitos muy elevados. Esto descarga mayoritariamente

Big Data

al usuario de las tareas de mantenimiento pero le resta flexibilidad a la hora de encontrar las especificaciones y también cuando se trata de instalar herramientas o características fuera del appliance, pues acostumbran a ser sistemas muy cerrados.

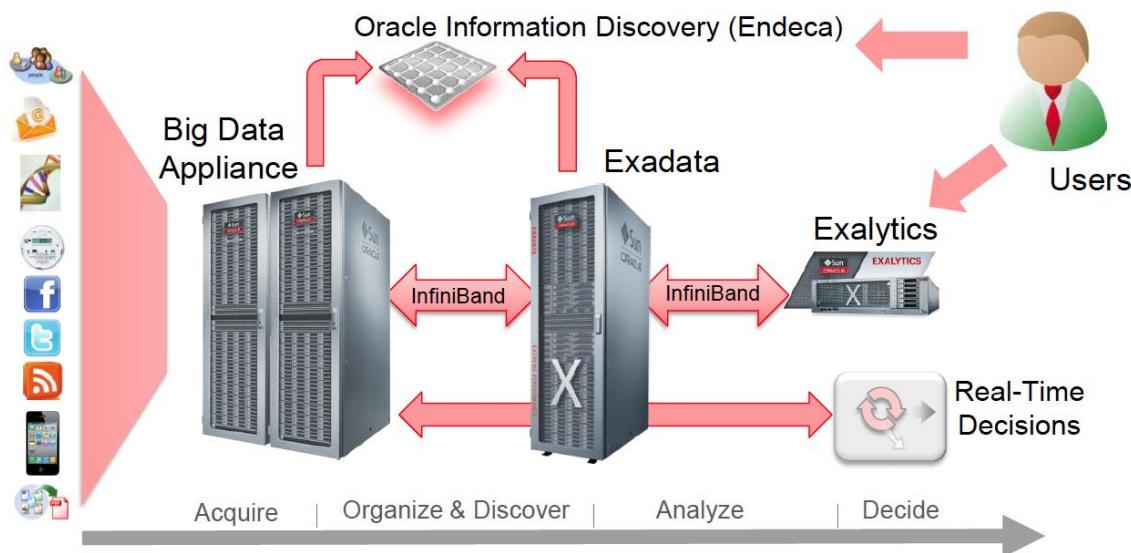


Ilustración 2: Esquema de la appliance de Oracle, una de las compañías líderes del sector. [13] Además, su appliance tiene una estructura mixta, como la comentada en el apartado 3.6.4 Arquitectura mixta.

3.7.3. CLOUD

El auge de las tecnologías de computación en la nube también llega hasta Big Data. Empresas como Amazon, Microsoft o IBM ofrecen sus servidores en la nube para configurar y desplegar soluciones Hadoop. La principal característica de este tipo de solución es que la empresa no se encarga del mantenimiento ni de la localización de las máquinas, ya que solo paga el coste del alquiler de los servidores en lugar de pagar el coste de adquisición (es decir, que solo se está alquilando la infraestructura). Esto hace que el precio sea más flexible ya que se paga únicamente lo que se usa pero para aplicaciones que deban usarse durante un periodo de tiempo bastante prolongado puede llegar a salir costoso. Además, se pierden los datos almacenados en cuanto se dejan de alquilar los servidores.

Otro punto a favor es la escalabilidad que ofrecen estos servicios, variar el número de nodos de un clúster suele ser bastante sencillo además de poder escoger que tipo de máquina se desea según las necesidades. En contraposición, el abanico de posibilidades a la hora de instalar aplicaciones suele estar acotado a un número limitado de opciones.

Big Data

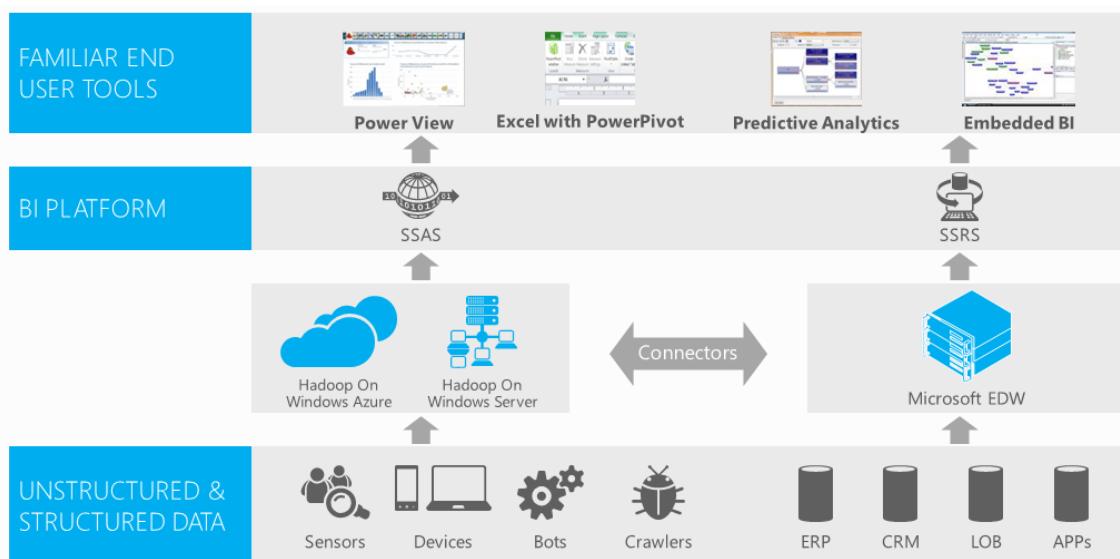


Ilustración 3: Microsoft tiene un ecosistema bastante completo para sistemas de explotación de la información. [14] En este caso tenemos otra arquitectura mixta que además ofrece servicios en cloud y también en local.

3.7.4. COMPARATIVA

Los puntos valorados en la comparativa entre los tipos de soluciones son los que se han considerado importantes para la realización del proyecto. De esta manera en este punto ya no se entra a valorar solamente qué es lo mejor para una solución Big Data sino cuál es el tipo de infraestructura que se necesita en este proyecto. Los parámetros escogidos en esta comparativa han sido seleccionados teniendo en mente las capacidades de la empresa -coste- y los objetivos del proyecto -flexibilidad, escalabilidad y, sobretodo, experimentación-. Las valoraciones van de 1 (peor) a 3 (mejor). A continuación se explica el significado de cada punto:

- **Coste:** el precio de la adquisición y mantenimiento de la solución. No se entra a detallar un precio exacto sino el tipo de gasto que implica la adquisición y mantenimiento de la infraestructura. Los baremos de valoración para este aspecto son:
 - **1:** el precio es elevado y muy poco flexible.
 - **2:** tiene un precio elevado pero es flexible a las necesidades del proyecto.
 - **3:** el precio es totalmente flexible a las necesidades del proyecto.
- **Flexibilidad:** las opciones que ofrecen a nivel de software. Significado de las valoraciones:
 - **1:** solo permite trabajar con el software que incluye de serie.
 - **2:** permite trabajar con otro software pero incluye algún tipo de restricción (permisos, lista de software compatible, etc.).
 - **3:** es totalmente abierto a la configuración de software deseada por el usuario.
- **Escalabilidad:** las facilidades o dificultades de cada infraestructura a la hora de cambiar el tamaño de los clústeres. Los baremos de valoración para este aspecto son:
 - **1:** permite la escalabilidad pero está restringida a un cierto tipo de hardware.
 - **2:** permite añadir y escalar sin importar el hardware pero requiere un cierto nivel de conocimiento.
 - **3:** añadir o quitar nodos es sencillo, no implica configuraciones nuevas ni trabajar con el hardware.
- **Experimentación:** la libertad que se ofrece para experimentar y hacer pruebas en la solución (tanto en hardware como en software), antes y después de la adquisición. Los baremos de valoración para este aspecto son:

Big Data

- **1:** no permite experimentar en exceso y además solo se puede en el entorno definitivo.
- **2:** permite experimentación pero en un entorno de producción.
- **3:** se puede realizar cualquier tipo de experimentación e incluso es posible de probar en un entorno de prueba.
- **Configuración:** los conocimientos y la facilidad de instalación y configuración. Los baremos de valoración para este aspecto son:
 - **1:** requiere un alto nivel de conocimientos, tanto de hardware como de software.
 - **2:** requiere solamente un nivel de conocimiento medio en la instalación de software.
 - **3:** es totalmente transparente al usuario por lo que no requiere conocimientos.

	Appliance	Cloud	Distribuciones
Coste	1	3	3
Flexibilidad	2	1	3
Escalabilidad	1	3	2
Experimentación	1	1	3
Configuración	3	3	1
Total	8	11	12

Tabla 12: Comparativa entre los tipos de infraestructuras appliance, en cloud y basadas en distribuciones.

- **Coste:** las valoraciones en este punto se deben estimar según las necesidades de cada uno. Las appliances son más caras pero ofrecen simplicidad y rendimiento sin que el usuario tenga que tener grandes conocimientos. En el caso de las infraestructuras en cloud se paga por el uso que se les da, haciéndolas más económicas en usos puntuales, sin perder la simplicidad en la configuración y aportando la ventaja de no tener que adquirir hardware. Las distribuciones ofrecen una solución intermedia, permite hacer la configuración de hardware y de software más adecuada a las necesidades del usuario -y ajustar los presupuestos a cambio de un cierto grado de conocimiento en Hadoop-, de manera que en usos continuos se puede llegar a amortizar la inversión.
- **Flexibilidad:** en las infraestructuras basadas en distribuciones se permite más margen a la hora de escoger el software que se usa, al tener un entorno de trabajo menos limitado, ya sea en hardware y configuración -como las appliances- o solamente en configuración -como las soluciones cloud-.
- **Escalabilidad:** en las appliances y distribuciones añadir nuevos servidores para aumentar la capacidad de almacenamiento y de procesado implica adquirir nuevas máquinas. En el caso de las appliances es más restrictivo aún porque se debe añadir el hardware específico de cada appliance (ya sea un módulo u otro rack). Para cloud simplemente se piden más servidores -alquilándolos- sin tener que realizar una gran inversión en tiempo ni en esfuerzo.
- **Experimentación:** tanto las appliances como las infraestructuras cloud tienen la limitación de tener que realizar las pruebas de concepto sobre las mismas soluciones, una vez ya pagadas e instaladas. Las distribuciones en cambio ofrecen un mayor nivel de permisividad a la hora de instalarlas en distintas máquinas (virtuales, por ejemplo) y hacer todo tipo de pruebas.
- **Configuración:** el gran hándicap de las soluciones basadas en distribuciones es que se necesita tener un conocimiento en Hadoop a nivel de administrador, para decidir cuál es la configuración adecuada. Las appliances vienen configuradas para ofrecer un rendimiento óptimo y las cloud también ofrecen configuraciones a elegir, simplificando este proceso.

[Big Data](#)

Después de realizar las valoraciones oportunas, la infraestructura más adecuada para este proyecto es la basada en distribuciones, puesto que deja ajustar el presupuesto a la vez que permite realizar todo tipo de pruebas sobre las distintas herramientas.

4. HADOOP

Hadoop es un proyecto de Apache de código libre en el que colaboran un gran número de empresas -entre ellas Yahoo!, Facebook, Microsoft o Cloudera [15]- y que su finalidad es englobar un conjunto de herramientas, aplicaciones y *frameworks* Java para el desarrollo de sistemas y utilidades de computación escalable y distribuida. La base de Hadoop está basada en los documentos originales de Google de MapReduce y de Google File System, el sistema de ficheros distribuido de Google. Actualmente hay cientos de empresas usando Hadoop para sus procesos -tanto internos como de servicios al cliente- pero entre las más destacadas encontramos a Microsoft, Facebook, Adobe, eBay, Google, IBM, Spotify o Twitter [16].



Hadoop permite la creación de aplicaciones para procesar grandes volúmenes de información distribuida a través de un modelo de programación sencillo. Está diseñado para ser escalable puesto que trabaja con almacenamiento y procesamiento local (pero distribuido), de manera que funciona tanto para clústeres de un solo nodo como para los que estén formados por miles. Otra característica importante de Hadoop es la detección de errores a nivel de aplicación, pudiendo gestionar los fallos en los distintos nodos y ofreciendo un buen nivel de tolerancia a errores.

El éxito de Hadoop ha sido tal que la mayoría de implementaciones de MapReduce con sistemas distribuidos usan Hadoop como base. En la actualidad hay un buen número de estas distribuciones comercializadas por las compañías más importantes del sector. En el apartado 6. *Distribuciones Hadoop* se estudian las distribuciones que se han considerado que tienen más importancia e influencia en el desarrollo de Hadoop.

El proyecto Hadoop está construido básicamente sobre dos módulos:

- **Hadoop Distributed File System (HDFS):** el sistema de ficheros sobre el que se ejecutan la mayoría de las herramientas que conforman el ecosistema Hadoop.
- **Hadoop MapReduce:** el principal framework de programación para el desarrollo de aplicaciones y algoritmos.

A parte de estos bloques también hay otros proyectos que completan el ecosistema Hadoop para desarrollar soluciones Big Data. Algunas de estas herramientas son estudiadas en el apartado 5. *Herramientas Hadoop*.

Actualmente hay dos versiones de Hadoop -1.0 y 2.0- que están siendo usadas por las distintas distribuciones. Ambas versiones tienen diferencias notables en su arquitectura y el hecho de coexistir de momento -hasta que finalmente desaparezca la primera versión- hace necesario su estudio. Además, de esta manera se facilita el entendimiento de la arquitectura Hadoop y las carencias de la primera versión que han llevado a los desarrolladores a hacer cambios significativos en la última.

4.1. HADOOP 1.0

A pesar de existir ya la versión 2.0 de Hadoop, la primera versión aún es bastante utilizada por muchos desarrolladores al ser la más sólida y estable. El proyecto original de Hadoop se construyó sobre tres bloques fundamentales:

- **HDFS**
- **MapReduce**
- **Hadoop Common**: es un conjunto de las principales librerías y utilidades que la mayoría de proyectos Hadoop utilizan.

En este apartado se estudia la versión 1.2.1 [17], la última estable en el momento de escribir esta memoria.

4.1.1. HADOOP DISTRIBUTED FILE SYSTEM (HDFS)

HDFS es el sistema de ficheros distribuido sobre el que se ejecutan las aplicaciones Hadoop y que proporciona un buen número de características al sistema. Es un sistema de ficheros distribuido -como los explicados en 3.5.2. *Almacenamiento*- diseñado especialmente para ejecutarse en hardware asequible o de bajo coste y para ser tolerante a fallos. Las principales características de HDFS son:



- **Sistema de ficheros amigable.** El esquema de HDFS está diseñado para que se parezca lo máximo posible a los sistemas de ficheros conocidos, especialmente al de Unix. Desde el espacio de nombres a los permisos de los ficheros y la seguridad.
- **Tolerancia a fallos de hardware.** Las instancias de HDFS pueden llegar a tener hasta miles de máquinas trabajando como servidores, almacenando cada una de ellas una parte del sistema de ficheros. Con tal cantidad de componentes formando un sistema, un fallo de hardware que implique la caída o desconexión de uno o más de estos componentes es la regla, no la excepción.
- **Acceso en streaming.** En las aplicaciones para las que está pensado HDFS, el usuario necesita acceder a los datos con un rendimiento constante y elevado.
- **Grandes cantidades de datos.** No solo en cuanto a la capacidad total del sistema de ficheros sino también al volumen individual de los ficheros que lo componen. Un tamaño normal y aconsejable para un fichero de HDFS puede ir de los gigabytes a los terabytes.
- **Modelo simple y coherente.** HDFS está pensado para aplicaciones que necesiten escribir el fichero una sola vez y que, una vez cerrado, no necesite cambios. De esta manera se puede conservar la coherencia de los datos y habilita su acceso rápido.
- **Portabilidad.** El sistema ha de ser portable a una gran cantidad de plataformas, tanto de hardware como de software.
- **Escalabilidad simple.** HDFS también permite la fácil expansión del sistema en caliente, pudiendo añadir nuevos nodos sin tener que pausar o parar los procesos que hay en ejecución en el clúster y sin tener que configurarlo; es decir, que el propio sistema se encarga de determinar qué bloques de ficheros almacenará y qué trabajos realizará. [18]

Todas estas características son invisibles para el usuario, ya que HDFS las realiza sin que éste tenga que hacer nada.

4.1.1.1. ARQUITECTURA HDFS

HDFS tiene una arquitectura de tipo maestro-esclavo. La arquitectura se compone de un servicio único llamado NameNode -que hace la función de maestro- y que se encarga de mantener la coherencia del sistema de ficheros y de permitir el acceso a los ficheros a los diferentes clientes. El otro servicio importante que compone la arquitectura es el DataNode, que acostumbra a estar activo en la mayoría - o todos- los nodos de un clúster. La función del DataNode es la de administrar el almacenamiento de los datos en el nodo donde se ejecutan.

La principal particularidad del sistema de ficheros HDFS es que cada fichero que se almacena en éste se divide en bloques y, a su vez, cada bloque se almacena en un nodo distinto. De esta manera se facilita el uso de modelos de programación como MapReduce, ya que se puede acceder a varios bloques de un mismo fichero de forma paralela. Para asegurar la disponibilidad de los datos y evitar la pérdida de estos debido a un error en alguno de los nodos, cada bloque está, además, replicado en distintos nodos, de manera que la caída de un nodo no implica la pérdida de los datos que contiene.

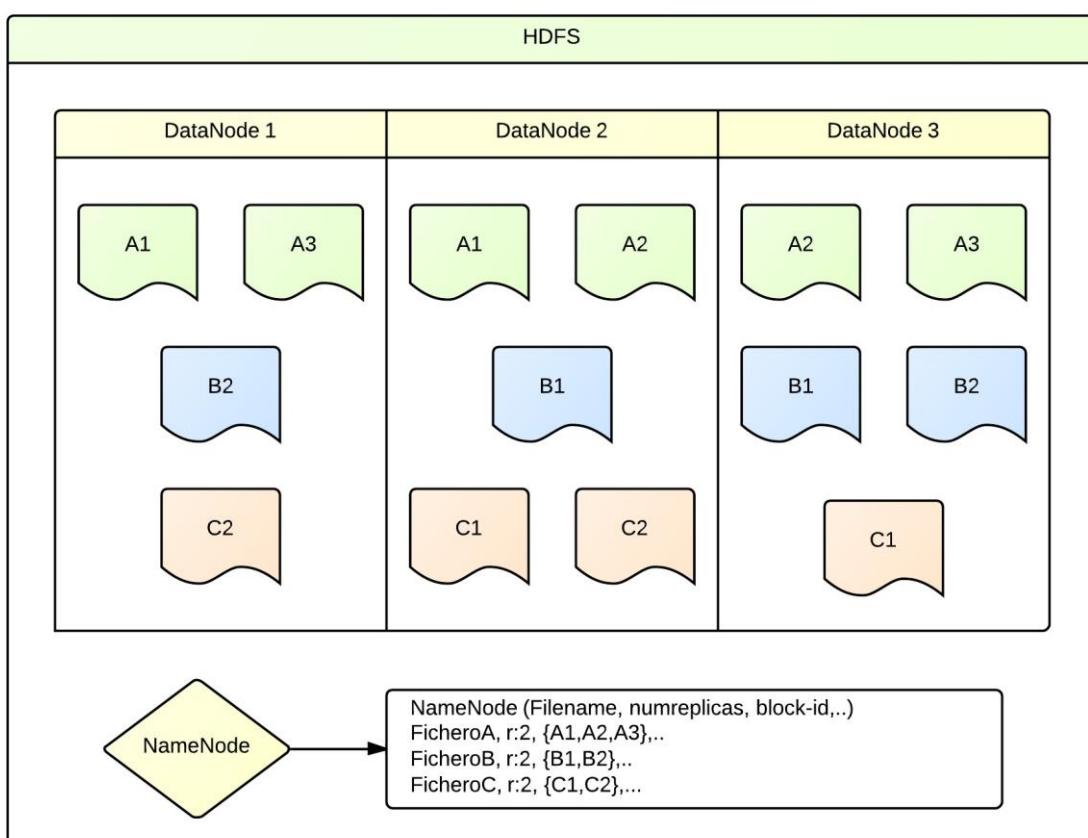


Figura 10: Ejemplo de replicación y división de los ficheros en bloques de tres ficheros (FicheroA, FicheroB y FicheroC) con factor de replicación dos. El NameNode guarda los metadatos del sistema de ficheros.

Hay dos aspectos de la configuración de HDFS importantes en este aspecto:

- **Tamaño de bloque:** indica el tamaño en bytes o megabytes que debe tener como máximo el bloque de un fichero. Por defecto está configurado a 64 MB o a 128 MB, que son los valores aconsejados para un buen rendimiento del sistema.
- **Factor de replicación:** el número de veces que se debe replicar cada fichero. Por defecto está configurado a tres y no se recomienda tener un factor de replicación menor. Hay que tener en cuenta que un factor de replicación elevado tampoco ayuda al rendimiento del equipo ya que cada fichero terminaría ocupando más espacio en disco.

Hadoop

En resumen, los DataNode son los encargados de servir los datos que contiene el nodo donde están activos mientras que el NameNode es el que gestiona la coherencia del sistema de ficheros a través de los metadatos.

En la *Figura 10* se puede observar un ejemplo de la división y replicación de los ficheros en bloques. Se tiene un sistema HDFS con un NameNode y tres DataNodes que contiene tres ficheros: FicheroA, FicheroB y FicheroC; con un factor de replicación dos. Los ficheros se dividen en bloques -cada uno con su propio identificador- y cada bloque está almacenado en dos DataNode distintos. La lista de ficheros y de bloques, con sus identificadores y en qué DataNode está almacenado cada uno, la administra el NameNode. Como se puede observar, cada fichero puede tener su propio factor de replicación (el segundo argumento de los metadatos indica el factor de replicación del fichero).

4.1.1.2. NAMENODE E INTEGRIDAD DEL SISTEMA

Debido a su arquitectura, el servicio NameNode es la parte más importante de un sistema HDFS. Aparte de guardar los metadatos del sistema, el NameNode tiene otras funciones muy básicas e importantes:

- **Balanceo de peticiones:** cuando un usuario o proceso pide acceso a un fichero el NameNode es el encargado de redirigirlo a los DataNodes correspondientes, intentando mantener siempre una uniformidad en la carga del sistema.
- **Detección de errores:** si una petición de acceso a un fichero falla, también es el encargado de redirigir las peticiones a un nuevo nodo que puedan satisfacerlas.
- **Redistribución de datos:** cuando se elimina o pierde un nodo, los datos que éste contenía no se pierden debido a la replicación pero sí que se rompe el factor de replicación. Para solventar este problema, el NameNode realiza copias de los bloques perdidos y los redistribuye entre los nodos para volver a tener integridad en los datos.
- **Balanceo de carga de datos:** por la naturaleza de los sistemas Hadoop, lo más probable es que con el tiempo la carga del sistema no sea siempre uniforme. La adición o supresión de nodos, por ejemplo, hace que queden algunos DataNodes vacíos o demasiado llenos. El NameNode tiene varias políticas para asegurarse la integridad de los datos y también el correcto balanceo del clúster cada vez que se añaden nuevos datos al clúster. Algunas de estas políticas son la de mantener una de las réplicas de los bloques en un rack distinto al que se está escribiendo, otra en el mismo (pero en un nodo distinto) o intentar mantener la uniformidad de espacio en el disco entre los nodos.
- **Comprobar la integridad del sistema:** cada vez que el sistema arranca, el NameNode se pone en modo seguro o safemode -el equivalente a modo read-only de un sistema de ficheros convencional- que no permite la modificación de ninguno de los bloques. Este estado dura mientras comprueba con todos los DataNode que la mayoría de los bloques están disponibles.
- **Mantenimiento de la integridad del sistema:** el NameNode guarda el estado del sistema en un fichero de log llamado *fsimage* y, cada vez que se realiza una operación de modificación dentro de HDFS, escribe en el fichero de log *edits* los cambios realizados. Cada vez que HDFS se inicializa, el NameNode carga el estado del sistema desde el fichero *fsimage* y realiza todos los cambios marcados en *edits*, volviéndolo a dejar vacío. También se pueden crear checkpoints o puntos de guardados para que el NameNode recupere un estado anterior mediante la importación de los ficheros *fsimage* y *edits*.

Cuando un sistema HDFS lleva mucho tiempo encendido, el fichero *edits* puede llegar a ser demasiado grande debido a que solamente se vacía durante el proceso de arranque del sistema, haciendo que la carga sea más lenta. Existe un segundo servicio, llamado Secondary NameNode, que tiene como objetivo principal mantener la integridad del sistema y evitar que esto último ocurra. Este servicio -que debería ejecutarse en un nodo distinto- puede tener varias instancias y realiza periódicamente las

Hadoop

modificaciones de *fsimage* marcadas en *edits* en lugar del NameNode, manteniendo así los logs dentro de unos límites de tamaño.

El NameNode es sumamente importante dentro de un sistema HDFS y su pérdida o desconexión debido a un error puede hacer que el sistema quede totalmente bloqueado y deje de funcionar hasta su recuperación. Por lo que el NameNode es un punto único de fallo en el sistema.

4.1.1.3. MONITORIZACIÓN, ADMINISTRACIÓN Y USABILIDAD

Aunque como se ha dicho HDFS funcione de manera transparente al usuario, hay varias maneras de trabajar con HDFS y administrar su funcionamiento y parámetros de configuración:

- **Interfaz web:** el NameNode cuenta con una interfaz web (por defecto está configurado para acceder desde la url: *namenode.hostname:50070*) en la que permite consultar información general del sistema (*Ilustración 4*), navegar por los directorios (*Ilustración 5*), consultar los logs o ver los ficheros y su lista de bloques.
- **API Java:** también se cuenta con una API de Java para crear aplicaciones o procesos que trabajen directamente sobre HDFS.
- **Línea de comandos:** Hadoop permite trabajar sobre HDFS mediante una consola de comandos de tipo UNIX mediante el comando “*hdfs dfs*” o “*hdfs dfsadmin*”. El primero es para realizar las operaciones clásicas de un sistema de ficheros (mover, copiar, eliminar ficheros) mientras que la segunda es para configurar el sistema de ficheros. También hay comandos para trabajar directamente con el NameNode (“*hdfs namenode*”), los DataNode (“*hdfs datanode*”) y el Secondary NameNode (“*hdfs secondarynamenode*”). En la *Tabla 13* hay algunos ejemplos de los comandos más utilizados.

Acción	Comando
Listar un directorio	<code>hdfs dfs -ls "directorio"</code>
Crear un directorio	<code>hdfs dfs -mkdir "directorio"</code>
Eliminar un fichero o directorio	<code>hdfs dfs -rm [-r] "path/fichero/o/directorio"</code>
Ver el contenido de un fichero de texto	<code>hdfs dfs -cat "path/fichero"</code>
Añadir un fichero a HDFS	<code>hdfs dfs -put "fichero/en/local" "directorio/HDFS"</code>
Sacar un fichero de HDFS	<code>hdfs dfs -get "fichero/en/HDFS" "directorio/en/local"</code>
Sacar varios ficheros de HDFS en uno solo	<code>hdfs dfs -getmerge "directorio/en/HDFS" "directorio/en/local"</code>
Muestra información sobre el sistema	<code>hdfs dfsadmin -report</code>
Entrar o salir del modo seguro	<code>hdfs dfsadmin -safemode (enter leave)</code>
Refresca la información de los DataNodes	<code>hdfs dfsadmin -refreshNodes</code>
Actualiza el NameNode después de una actualización de Hadoop	<code>hdfs namenode -upgrade</code>
Carga un checkpoint del sistema	<code>hdfs namenode -importCheckpoint "directorio/checkpoint"</code>

Tabla 13: ejemplos de algunos comandos para trabajar y administrar HDFS.

Cluster Summary

Security is OFF

1625 files and directories, 1576 blocks = 3201 total.

Heap Memory used 55.37 MB is 68% of Committed Heap Memory 81.06 MB. Max Heap Memory is 1019.88 MB.

Non Heap Memory used 49.47 MB is 62% of Committed Non Heap Memory 78.91 MB. Max Non Heap Memory is 130 MB.

Configured Capacity	:	330.17 GB
DFS Used	:	180.70 GB
Non DFS Used	:	32.67 GB
DFS Remaining	:	116.80 GB
DFS Used%	:	54.73%
DFS Remaining%	:	35.38%
Block Pool Used	:	180.70 GB
Block Pool Used%	:	54.73%
DataNodes usages	:	Min % Median % Max % stdev %
		54.72% 54.73% 54.73% 0.00%
Live Nodes	:	3 (Decommissioned: 0)
Dead Nodes	:	0 (Decommissioned: 0)
Decommissioning Nodes	:	0
Number of Under-Replicated Blocks	:	8

Ilustración 4: Información general del clúster mostrada por la interfaz web del NameNode.

Contents of directory /user

Goto :

[Go to parent directory](#)

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
cloudera	dir				2013-10-14 01:25	rwxrwxrwx	cloudera	cloudera
flume	dir				2013-10-03 03:12	rwxrwxrwx	flume	supergroup
hdfs	dir				2013-07-12 02:55	rwxrwxrwx	hdfs	supergroup
history	dir				2013-07-30 06:45	rwxrwxrwx	mapred	hadoop
hive	dir				2013-07-15 01:11	rwxrwxr-t	hive	hive
hue	dir				2013-07-29 03:14	rwxr-xr-x	hue	hue
oozie	dir				2013-07-15 01:15	rwxrwxr-x	oozie	oozie
root	dir				2013-07-12 03:01	rwx-----	root	supergroup

[Go back to DFS home](#)

Ilustración 5: Navegación por los directorios de HDFS a través de la interfaz web del NameNode.

Hadoop

File: [/user/hive/warehouse/comments25g/f25600](#)

Goto : [/user/hive/warehouse/com](#)

[Go back to dir listing](#)

[Advanced view/download options](#)

[View Next chunk](#)

```
351490327393206273      From 0-10,00,00 #Google+ circlers in 3 months.~ @tygrscott. http://t.co/nzMlOhETiv
01:59:59.0      HootSuite      #google
351490068873101314      Plz see http://t.co/Gpl5Dxu7f6 our 1st #Google #1 Ranked #offbeat #cartoon shop 4 #o
TweetAdder v4      #google
351490023029358592      #Google domina publicidad en móviles http://t.co/TbdNYegO3E #Mobile #Publicidad 4190
351489851297759234      #Google #Hot #News Brooke Hogan is Engaged to Dallas Cowboys Player Phil Costa! http
dlvr.it      #google
351489704543264768      Quick UDP Internet Conn. #Google again demonst. that a data structure expert is not
01:57:30.0      web          #google
351489523911372800      #yahoo #Google #wp http://t.co/7Yha3wKS8w asks What is #keyword density? Can it be i
01:56:47.0      TweetAdder v4      #google
351489505297055747      RT @ByOneSpirit: 'The Sign of the Covenant' http://t.co/YQuxeUGyyJ on #Google+ 2615
351489346584588290      Plz Visit http://t.co/OXo6ol9BGt our largest #Google #1 Ranked #offbeat #cartoon sho
TweetAdder v4      #google
351489303441981441      'The Sign of the Covenant' http://t.co/YQuxeUGyyJ on #Google+ 260757663 By C
351489029969158144      RT @DavidJamesJnr: New #PRISM slides suggest that #Google, #Facebook, #Microsoft, et
Twitter for iPhone
351488355311169537      #Google #Hot #News Fast & Furious 6 http://t.co/LAAlanI4Gh5 #TeamFollowBack YANews
351488312235663362      Check out http://t.co/L2eV8xQ1W6 new shop of #Google #1 #weirdest shop 4 #iPhone c
351488223614222336      RT @ellalg: Happy Pride Day! #google represent! #nycpride #nyc #loveislove #firstins
01:51:37.0      web          #google
351488115950624768      PRISM, la machine américaine pour espionner le monde http://t.co/BFmhJAAFTf #espion
Post      #google
351488076251545600 BRUCE LEE The only person in the world to DEFEAT Chuck ...lol #google tho 354520794 _n*.c
351487891316285441      The Weekly Roundup for 06.24.2013 http://t.co/AtMoaQCTfR #active #android #apple #go
```

[Download this file](#)

[Tail this file](#)

Chunk size to view (in bytes, up to file's DFS block size):

Total number of blocks: 200

9138498662761311927:	7.110.8.23:50010 View Block Info	7.110.8.22:50010 View Block Info	7.110.8.21:50010 View Block Info
-2761169775562361233:	7.110.8.23:50010 View Block Info	7.110.8.22:50010 View Block Info	7.110.8.21:50010 View Block Info
5137899440932880069:	7.110.8.23:50010 View Block Info	7.110.8.22:50010 View Block Info	7.110.8.21:50010 View Block Info
-6208634936870400560:	7.110.8.23:50010 View Block Info	7.110.8.22:50010 View Block Info	7.110.8.21:50010 View Block Info
-378310171874420866:	7.110.8.23:50010 View Block Info	7.110.8.22:50010 View Block Info	7.110.8.21:50010 View Block Info
-9033515255928099211:	7.110.8.23:50010 View Block Info	7.110.8.22:50010 View Block Info	7.110.8.21:50010 View Block Info
2167797110776843719:	7.110.8.23:50010 View Block Info	7.110.8.22:50010 View Block Info	7.110.8.21:50010 View Block Info
5915687913166392131:	7.110.8.23:50010 View Block Info	7.110.8.22:50010 View Block Info	7.110.8.21:50010 View Block Info
-6755968614403662830:	7.110.8.23:50010 View Block Info	7.110.8.22:50010 View Block Info	7.110.8.21:50010 View Block Info
-5498385858195514154:	7.110.8.23:50010 View Block Info	7.110.8.22:50010 View Block Info	7.110.8.21:50010 View Block Info

Ilustración 6: Vista de un fichero de HDFS desde la interfaz web del NameNode. Permite ver su contenido y descargarlo.

También lista los bloques que conforman el fichero y en que nodos están replicados cada uno.

4.1.2. MAPREDUCE 1.0

La versión de MapReduce de Hadoop está basada en los documentos que Google escribió inicialmente - explicados en el apartado 3.6.1. *MapReduce*- y está preparada para trabajar con HDFS y, por lo tanto, para que se ejecute sobre el mismo clúster. De esta manera obtiene algunas de las características que ofrece el sistema de ficheros de Hadoop como la alta disponibilidad de los datos, su distribución y la integridad del sistema.

El framework, escrito en Java aunque tenga una versión para C++, está pensado para escribir procesos de la forma más sencilla posible y para que el programador solo tenga que pensar en su algoritmo. El sistema se encarga de gestionar las tareas o trabajos MapReduce, monitorizarlos y ejecutarlos de nuevo si han fallado.

4.1.2.1. ARQUITECTURA MAPREDUCE

Al igual que con HDFS, MapReduce cuenta con una arquitectura maestro-esclavo y con dos tipos de servicios que conforman su arquitectura:

- **Servicio JobTracker:** el encargado de gestionar, monitorizar y distribuir la carga de los trabajos MapReduce.
- **Servicio TaskTracker:** se ejecuta en un nodo con un servicio DataNode de HDFS. Recibe las órdenes del JobTracker y se encarga de realizar el proceso sobre el bloque de datos que contenga.

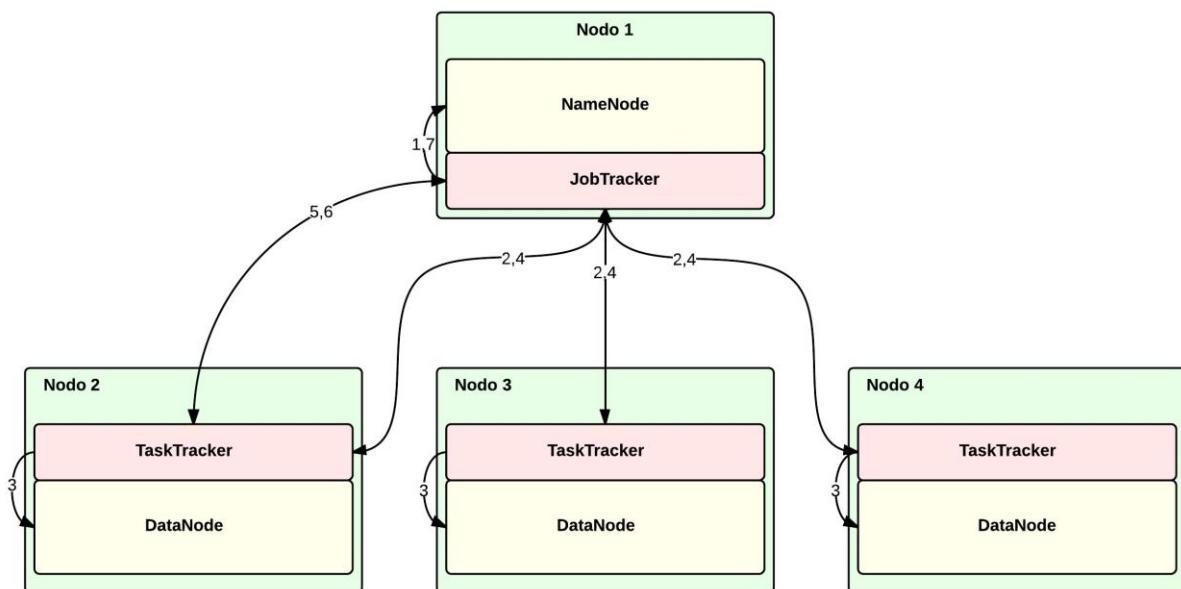


Figura 11: Workflow de la ejecución de un trabajo MapReduce en una arquitectura Hadoop con cuatro nodos: tres de trabajo (DataNode+TaskTracker) y uno de gestión (NameNode+JobTracker).

En la *Figura 11* podemos observar cual es el *workflow* habitual (sin errores, entre otras eventualidades) de un trabajo MapReduce en una arquitectura Hadoop con HDFS como sistema de ficheros. La arquitectura de la figura se compone de cuatro nodos: tres nodos de trabajo -DataNode+TaskTracker- y uno de gestión o administración -NameNode+JobTracker-. Los pasos que se ejecutan en el clúster explicados con detalle son los siguientes:

1. Se ejecuta el proceso MapReduce. El JobTracker recibe los datos de ejecución (qué proceso se lanza y sobre qué datos -almacenados en HDFS- se realiza). Se comunica con el NameNode para conocer la distribución de los datos.

Hadoop

2. Una vez sabe qué DataNodes contienen los bloques con los que hay que trabajar, distribuye los procesos map a los TaskTrackers candidatos, los que se ejecutan en los nodos que contengan los bloques.
3. Cada TaskTracker realiza el trabajo sobre el bloque asignado comunicándose con el DataNode.
4. Una vez obtiene los resultados, los TaskTrackers los envían al JobTracker.
5. Una vez todos los procesos map han terminado, el JobTracker envía los resultado a un nodo de trabajo para que realice el reduce del proceso.
6. El TaskTracker realiza el reduce y envía los resultados al JobTracker.
7. El JobTracker envía los resultados a HDFS para que puedan ser consultados.

Puede que durante la ejecución de un proceso MapReduce alguno de los nodos de trabajo tenga un error o se desconecte del clúster, en ese caso el JobTracker lo detecta y, gracias a que los datos están replicados en HDFS y en distintos nodos, delega el trabajo fallido sobre otro nodo que contenga el bloque. De esta manera cuando hay un error no se tiene que volver a rehacer todo el proceso sino que solamente tiene que volver a ejecutarse la parte perdida.

4.1.2.2. ETAPAS DE UN PROCESO MAPREDUCE

En el ejemplo anterior se explicaba la función de los servicios JobTracker y TaskTracker y su relación con los de HDFS. Pero el esquema de trabajo de un proceso MapReduce es más complejo y completo. Durante una ejecución MapReduce los datos pasan por muchas fases o etapas:

- **Mapping** (mapeo): en esta etapa los nodos escogidos por el JobTracker -llamados Mappers- realizan el trabajo de forma paralela y distribuida sobre los datos de entrada -uno o varios ficheros en HDFS- y obtienen una salida con la forma:

$$\text{list}(\text{Key}_{mo}, \text{Value}_{mo})$$

- **Shuffle** (mezcla): los resultados obtenidos en la etapa anterior se mezclan, obteniendo una lista con todas las parejas clave-valor.
- **Sort** (ordenación): se agrupan las parejas resultado del mapeo basándose en las claves. En caso de buscar otro orden de ordenación, el usuario puede implementar una segunda ordenación usando un objeto Comparator y configurando el trabajo para usarlo. Se obtiene un resultado con el siguiente formato:

$$\text{list}(\text{Key}_{so}, \text{list}(\text{Values}_{so}))$$

- **Partitioning** (partición): parte la lista resultante de la etapa de ordenación delegando cada parte a un Reducer. El objeto Partitioner puede ser implementado si el usuario busca una partición distinta a la que viene por defecto (un HashPartitioner).
- **Combining** (combinación): esta es la etapa que menos suele aparecer ya que se acostumbra a configurar los trabajos para que el Combiner sea el mismo que el Partitioner y, en caso contrario, acostumbra a usarse para liberar de trabajo a este último.
- **Reducing** (reducción): se realiza la reducción final del trabajo. La entrada del Reducer -el nodo que realiza esta función- recibe las claves y las listas de valores para estas claves. Puede haber más de un Reducer o ninguna, si lo que se busca es la salida de la fase de mapeo. El número de Reducers aconsejables acostumbra a ser:

$$(0.95|1.75) * (<\text{número de nodos}> * <\text{número de Reducers}>)$$

Cuando se usa la constante 0.95 todos los Reducers pueden lanzarse a la vez, mientras que usando 1.75, se lanzaran por olas a medida que vayan terminando (puede darse el caso que los nodos más rápidos realicen más reduces que los más lentos).

Hadoop

La salida de un proceso MapReduce se almacena en un directorio dentro de HDFS y contiene los siguientes ficheros:

- `_SUCCESS`: es un fichero vacío que indica que el proceso ha terminado de manera satisfactoria.
- `_FAIL`: es un fichero vacío que indica que el proceso ha terminado de manera incorrecta.
- `Part-r-XXXXX`: es el resultado de un Reducer, donde `XXXXX` corresponde a un número identificativo del Reducer. Por lo que hay un fichero por Reducer.
- `Part-m-XXXXX`: para los casos que se realiza la fase de reducción. Es el resultado de un Mapper, donde `XXXXX` corresponde a un número identificativo del Mapper, por lo que hay un fichero para cada uno.

4.1.2.3. PROGRAMACIÓN DE UN TRABAJO MAPREDUCE

Como ya se ha comentado anteriormente, existe una API Java para programar los trabajos MapReduce. La API requiere básicamente de la implementación de dos interfaces sencillas -aunque como se verá más adelante se puede configurar el trabajo añadiendo más objetos personalizados-:

INTERFAZ MAPPER

Las clases que implementen la interfaz Mapper deben extender también la clase `MapReduceBase` y se les debe indicar el formato de entrada y salida de los datos:

```
public class MyMapper extends MapReduceBase implements Mapper < Ki, Vi, Ko, Vo >
```

En esta interfaz Mapper, la pareja `Ki-Vi` hace referencia a los tipos de los datos de entrada (parejas clave-valor) mientras que `Ko-Vo` hace referencia al formato de salida (también parejas clave-valor). En los cuatro casos los objetos deben ser de tipo `Writable` -es decir, heredar de la clase `Writable`- y no necesitan ser del mismo tipo.

La única función a implementar en esta interfaz es la siguiente:

```
void map(Ki key, Vi value, OutputCollector < Ko, Vo > output, Reporter reporter)
```

Esta función es la que se ejecutará en cada nodo de trabajo durante la fase de map. Los Mapper están pensados para trabajar con ficheros de texto por lo que leerá la entrada línea a línea y llamará la función `map`. Generalmente los parámetros de la función `map` se usan de la siguiente manera:

- Key: número de línea del fichero.
- Value: línea de texto.
- Output: la salida en formato de parejas del tipo `<Ko,Vo>`. Las parejas se añaden a la salida con la llamada a la siguiente función:

```
OutputCollector.collect(Object, Object)
```

- Reporter: se usa para informar del estado de proceso.

En la *Figura 12* se puede observar la implementación de un Mapper para un trabajo MapReduce wordcount (parecido al de la *Figura 8*), que cuenta cuantas apariciones tiene cada palabra. Como entrada recibe el número de línea y el texto de la línea y como salida genera una lista de parejas palabra-uno, que contiene una entrada para cada aparición de cada palabra.

```

public static class Map extends MapReduceBase
    implements Mapper<LongWritable, Text, Text, IntWritable> {
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(LongWritable key, Text value,
                    OutputCollector<Text, IntWritable> output,
                    Reporter reporter)
        throws IOException {
        String line = value.toString();
        StringTokenizer tokenizer = new StringTokenizer(line);
        while (tokenizer.hasMoreTokens()) {
            word.set(tokenizer.nextToken());
            output.collect(word, one);
        }
    }
}

```

Figura 12: Ejemplo de implementación de un map para un problema wordcount (21). Cuenta cuantas veces aparece cada palabra en un fichero de texto. La función map trata el fichero de línea en línea.

INTERFAZ REDUCER

Como en las clases que implementan los Mappers, los Reducers deben extender MapReduceBase y también deben indicar los tipos de entrada y salida (en este caso, además, los tipos de entrada del Reducer deben coincidir con los tipos de salida del Mapper).

```
public static class MyReduce extends MapReduceBase implements Reducer < Ki, Vi, Ko, Vo >
```

Como en el caso de estos últimos se tiene que implementar una función:

```

public void reduce(Ki key, Iterator < Vi > values,
                    OutputCollector < Ko, Vo > output, Reporter reporter)

```

Los parámetros de la función varían un poco respecto con los del Mapper:

- Key: clave de los valores que recibe el Mapper.
- Values: los valores correspondientes a la clave key.
- Output: la salida en formato de parejas del tipo <Ko,Vo>.
- Reporter: se usa para informar del estado de proceso.

Como se puede observar, se ejecuta la función reduce para cada clave que reciba el Reducer.

```

public static class Reduce extends MapReduceBase
    implements Reducer<Text, IntWritable, Text, IntWritable> {
    public void reduce(Text key, Iterator<IntWritable> values,
                      OutputCollector<Text, IntWritable> output,
                      Reporter reporter)
        throws IOException {
        int sum = 0;
        while (values.hasNext()) {
            sum += values.next().get();
        }
        output.collect(key, new IntWritable(sum));
    }
}

```

Figura 13: Ejemplo de implementación de un reduce para un problema wordcount (21). Cuenta cuantas veces aparece cada palabra en un fichero de texto.

En la *Figura 13* se implementa el Reducer para el wordcount del apartado anterior. En este caso reduce recibe todas las entradas (el iterador *values*) correspondientes a una palabra (la clave *key*) y suma todos los valores (que en el map se han configurado a uno). De esta manera se obtiene el número de veces que aparece una palabra en un fichero de texto almacenado en HDFS.

CONFIGURACIÓN DE UN TRABAJO MAPREDUCE

Una vez implementadas las clases Mapper y Reducer ya sólo queda configurar una tarea MapReduce y ejecutarla. La configuración de la tarea se hace con la clase JobConf, donde se configura cuál es la implementación del Mapper y cuál la del Reducer, los tipos de las entradas y salidas de ambas clases y otras configuraciones como los ficheros de entrada o dónde se quiere almacenar la salida.

En la *Figura 14* se puede ver el ejemplo de la implementación de la clase WordCount, que lanza un trabajo MapReduce con las clases implementadas en los ejemplos de los apartados anteriores.

La clase JobConf es la clase principal a la hora de configurar un trabajo MapReduce vía código y puede configurar las siguientes opciones (*Tabla 14*):

Nombre de la configuración	Función de JobConf	Descripción
Job name	setJobName(string)	Define el nombre del trabajo
Set Jar	setJar(string)	Indica el Jar a usar para el trabajo MapReduce.
Mapper class	setMapperClass(class)	Indica la clase Mapper
Reducer class	setReducerClass(class)	Indica la clase Reducer
Input format	setInputFormat(class)	Indica la clase de para leer la entrada
Output format	setOutputFormat(class)	Indica la clase para escribir la salida
Output key class	setOutputKeyClass(class)	Indica de que tipo serán las claves en la salida del Mapper
Output value class	setOutputValueClass(class)	Indica de que tipo serán los valores en la salida del Mapper

Hadoop

Combiner	setCombinerClass(class)	Indica la clase que hará de Combiner
Partitioner	setPartitionerClass(class)	Indica la clase que hará de Partitioner
Set secondary Sort	setOutputValueGroupingComparator(class)	Indica la clase para hacer una ordenación alternativa
Set number of Mappers	setNumMapTasks(int)	Configura el número de Mappers
Set number of Reducers	setNumReduceTasks(int)	Configura el número de Reducers

Tabla 14: Algunas de las opciones que acepta la clase JobConf para configurar un trabajo MapReduce.

```

public class WordCount {
    public static void main(String[] args) throws Exception {
        JobConf conf = new JobConf(WordCount.class);
        conf.setJobName("wordcount");
        conf.setOutputKeyClass(Text.class);
        conf.setOutputValueClass(IntWritable.class);
        conf.setMapperClass(Map.class);
        conf.setCombinerClass(Reduce.class);
        conf.setReducerClass(Reduce.class);
        conf.setInputFormat(TextInputFormat.class);
        conf.setOutputFormat(TextOutputFormat.class);

        FileInputFormat.setInputPaths(conf, new Path(args[0]));
        FileOutputFormat.setOutputPath(conf, new Path(args[1]));

        JobClient.runJob(conf);
    }
}

```

Figura 14: ejemplo de configuración y ejecución de una tarea MapReduce con las clases de la Figura 13 y de la Figura 14. (21)

Hadoop

La Figura 15 muestra cómo sería el *dataflow* del proceso (con un ejemplo bastante simplificado) wordcount implementado en este apartado paso a paso:

1. Se añade el fichero original a HDFS (se almacena en tres bloques en tres nodos distintos).
2. El Mapper se ejecuta en cada nodo sobre el bloque correspondiente, generando una lista de parejas.
3. La salida de cada Mapper se envía al nodo que hará de Reducer y se hace la fase de Shuffle y Sort.
4. El Reducer suma los valores de cada clave y deja el resultado en un fichero HDFS.

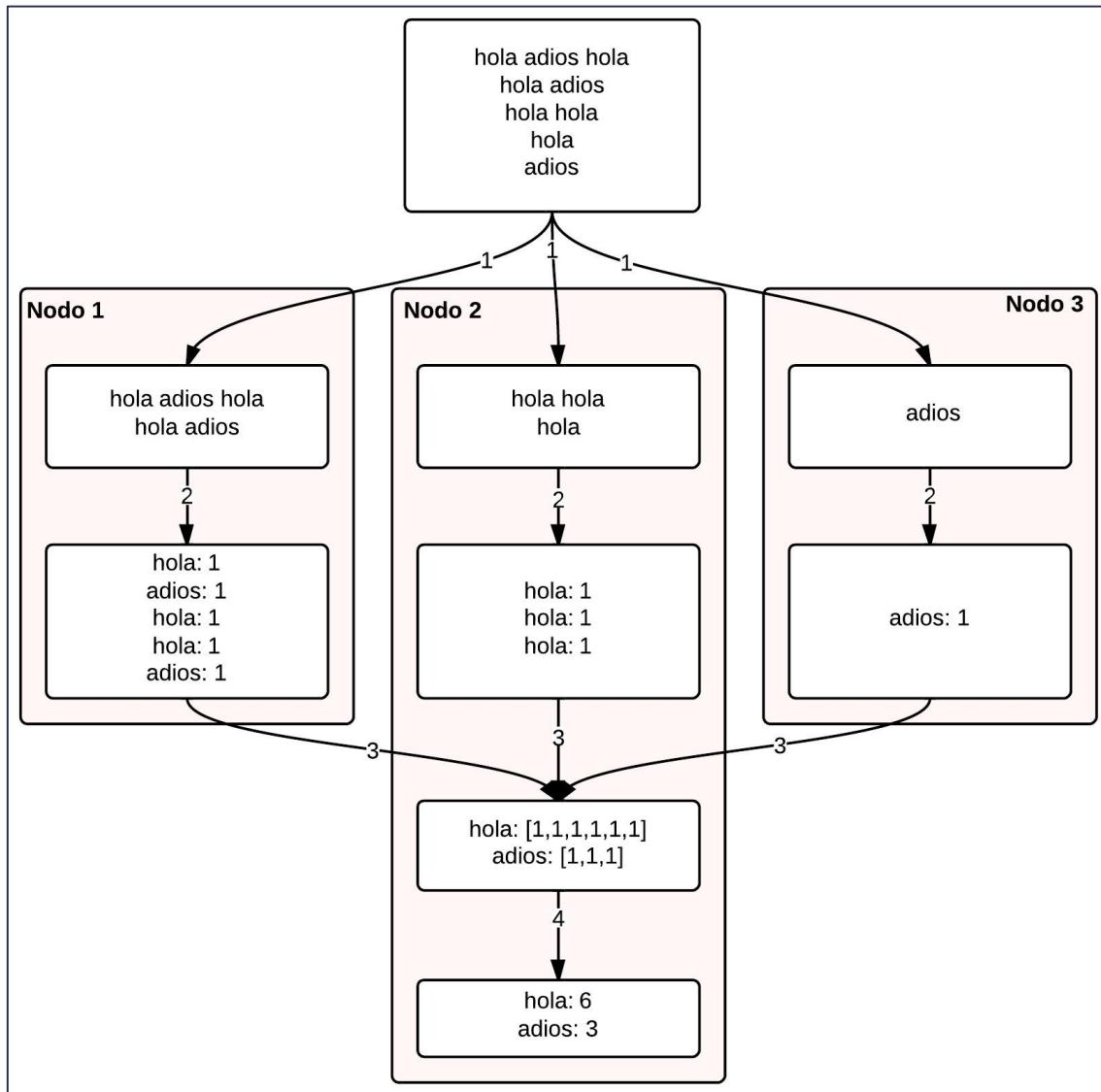


Figura 15: Ejemplo simplificado de la ejecución del WordCount implementado.

4.2. HADOOP 2.0

La segunda versión de Hadoop parte con la base de Hadoop 1.0 y añade -y modifica también- algunas características de sus módulos para tratar de resolver algunos de los problemas que tenía y mejorar el rendimiento del sistema. El proyecto Hadoop 2.0 está dividido esta vez en cuatro módulos:

- **Hadoop Common**
- **Hadoop Distributed File System (HDFS)**
- **Hadoop YARN**: un *framework* para la gestión de aplicaciones distribuidas y de recursos de sistemas distribuidos.
- **Hadoop MapReduce**: el sistema de procesamiento principal, que esta vez se ejecuta sobre YARN.

En este apartado se estudia la versión 2.2.0 [19], la última en el momento de escribir esta memoria.

4.2.1. HDFS 2.0

Los cambios introducidos en HDFS han sido pocos pero significativos. Se intenta combatir la principal debilidad de la primera versión: el NameNode como punto de fallo único en el sistema. Esto evita que un sistema HDFS tenga alta disponibilidad, ya que un fallo en el NameNode hace que el sistema deje de funcionar. Otra novedad introducida es la HDFS Federation, que permite tener múltiples espacios de nombres en HDFS.

El resto de las características que ofrece la primera versión de HDFS se han mantenido prácticamente intactas, desde la arquitectura con NameNodes y DataNodes a la monitorización a través de una interfaz web o la ejecución de comandos por consola.

4.2.1.1. ALTA DISPONIBILIDAD

La alta disponibilidad del NameNode se puede conseguir de dos maneras: a través del Quorum Journal Manager o usando Network File System.

QUORUM JOURNAL MANAGER

En este tipo de arquitectura se configura, aparte del NameNode principal, un segundo NameNode que está en modo espera o standby, llamado precisamente Standby NameNode. Este servicio permanece inactivo a la espera de un fallo en el NameNode activo, que es el encargado de realizar las tareas de gestión y administración del sistema.

Para mantener la coherencia de los datos entre los dos NameNodes y mantenerlos sincronizados se crea un grupo de servicios, llamados JournalNodes, cuya función es la de actuar como diarios de todas las operaciones que el NameNode activo va realizando. Este conjunto de JournalNodes se llama Quorum Journal Manager.

El funcionamiento de un sistema HDFS con Quorum Journal Manager es el que se muestra en la *Figura 16*. El NameNode comunica a un grupo de JournalNodes (no hace falta que lo haga con todos ya que entre ellos se sincronizan) todos los cambios que se van realizando en el sistema de ficheros -es decir, en los DataNodes-. El Standby NameNode, por su parte, va leyendo el estado del sistema a través de los JournalNodes de manera que cuando se produce un evento de fallida pueda actuar rápidamente como NameNode activo.

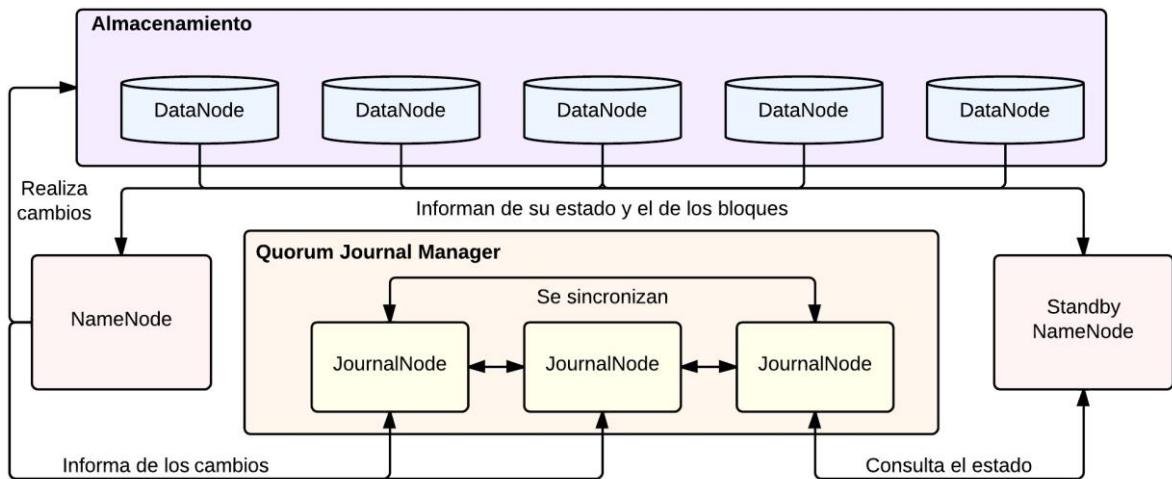


Figura 16: Esquema de los servicios de HDFS con Quorum Journal Manager y la comunicación entre ellos.

Para asegurar la coherencia del sistema de ficheros se toman ciertas medidas:

- Sólo puede haber un **NameNode** activo ya que la existencia de más de uno podría provocar fallos de sincronización y una ruptura del sistema de ficheros. Los **JournalNode** sólo dan el permiso para realizar cambios a uno de los **NameNode**s; de manera que cuando hay un fallo en el activo, el inactivo pasa a reemplazarlo en el rol de servicio activo.
- El **NameNode** inactivo solamente pasa a activo cuando se ha asegurado de que ha leído todos los cambios contenidos en los **JournalNode**s.
- Los **DataNode**s están configurados con las direcciones de los dos servicios de **NameNode**, tanto del activo como del inactivo, para enviar constantemente información de los bloques e informar de su estado. Con esto se logra que la transición entre **NameNode**s sea lo más rápida posible, ya que ambos tienen toda la información necesaria para realizar las tareas de un servicio activo.
- Un **JournalNode** es un servicio que requiere de pocos recursos por lo que puede ser configurado en nodos con otros servicios del clúster -**NameNode**, **DataNode**, **JobTracker**, etc.-.
- Se recomienda tener, como mínimo, tres servicios **JournalNode** para tener un Quorum preparado para la pérdida de uno de los nodos. Para poder aumentar el número de caídas toleradas se debe incrementar el número de servicios en cantidades impares (3, 5, 7...), ya que el Quorum tolera la caída de $(N - 1)/2$ nodos, donde N es el número total de servicios **JournalNode**.

El uso del Quorum Journal Manager es totalmente transparente para el usuario y la transición del **Standby NameNode** a servicio **NameNode** principal puede ser configurada como automática o manual. Para que esta transición sea automática se requiere de **ZooKeeper** -explicado en 5.1.2. **ZooKeeper**- para detectar los fallos en el **NameNode** y monitorizar su estado.

Se han añadido comandos nuevos a los ya existentes para poder administrar la alta disponibilidad del sistema (*Tabla 15*):

Acción	Comando
Cambiar el estado de un NameNode manualmente	<code>hdfs haadmin [-transitionToActive -transitionToStandby] <serviceID></code>
Provocar el fallo de un NameNode y que sea	<code>hdfs haadmin -failover <serviceID> <serviceID></code>

Hadoop

reemplazado por otro	
Determinar si un NameNode está activo o no	hdfs haadmin -getServiceState <serviceID>
Comprobar el estado de un NameNode	hdfs haadmin -checkHealth <serviceID>

Tabla 15: Comandos para administrar la alta disponibilidad de un sistema HDFS con Quorum Journal Manager.**NETWORK FILE SYSTEM**

Este método para lograr la alta disponibilidad en HDFS también cuenta con un NameNode principal y un Standby NameNode, realizando las mismas funciones que en el caso del Quorum Journal Manager. La diferencia principal es que en lugar de usar un Quorum para la sincronización entre los NameNodes se utiliza un dispositivo de almacenamiento conectado mediante una Network File System. NFS es un protocolo de sistemas de ficheros en red que permite a diferentes ordenadores acceder a ficheros en remoto.

Garantizando el acceso de los NameNodes a este dispositivo a través de NFS, el sistema funciona de manera muy parecida a como lo hace con el Quorum. Cada vez que el NameNode activo realice cambios sobre el sistema de ficheros lo dejará indicado en un fichero de ediciones, almacenado en un directorio del dispositivo NFS. De esta manera el nodo en standby puede ir consultando los cambios realizados en el espacio de nombres del sistema de ficheros. El esquema de trabajo realizado por los nodos sería muy parecido al de la *Figura 16* pero cambiando el Quorum por un dispositivo conectado a través de una red NFS.

Tal y como pasa con el Quorum Journal Manager, solo puede haber un nodo activo a la vez y uno en standby solo cambia de estado a activo cuando está seguro de haber leído todos los cambios del fichero de ediciones.

4.2.1.2. HDFS FEDERATION

Una división por capas y explicación simplificada de la arquitectura de HDFS -y generalizando de cualquier sistema de ficheros- sería la siguiente:

- **Espacio de nombres:** es la parte lógica del sistema y la que el usuario más acostumbra a ver. El conjunto de directorios, ficheros y bloques así como su estructura jerárquica. El espacio de nombres de HDFS permite las operaciones para crear, borrar, modificar y conseguir la localización de los bloques.
- **Servicio de almacenamiento:** es la parte más física -a nivel de software- del sistema y acostumbra a ser transparente al usuario. Tiene dos partes:
 - **Administración de bloques:** realizado por el NameNode.
 - **Almacenamiento:** realizado por el DataNode.

La primera versión de HDFS trabajaba con un espacio de nombres único e igual al de un sistema de ficheros Unix (con la excepción de la operación para conseguir los bloques). Con la aparición de HDFS Federation esto cambia, ahora se puede tener varios espacios convirtiendo HDFS en escalable a nivel horizontal.

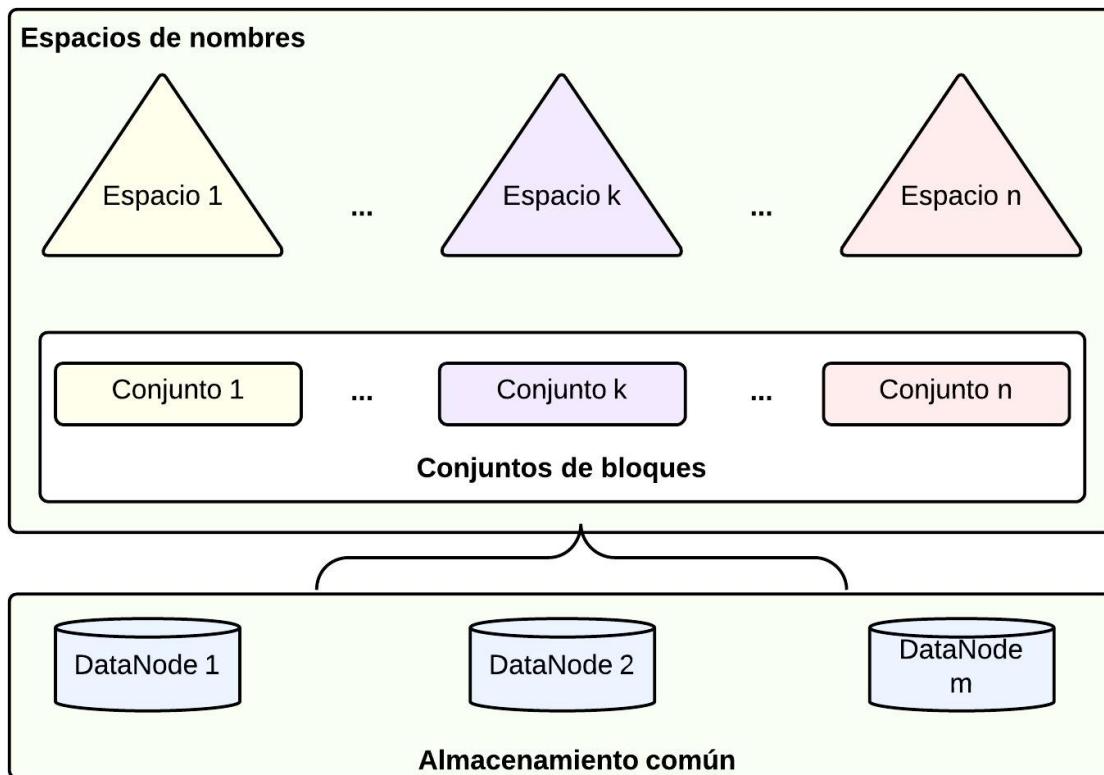


Figura 17: Arquitectura HDFS con una federación de n espacios de nombre y m DataNodes.

La arquitectura de una federación HDFS es la que se muestra en la Figura 17, con un NameNode para cada espacio y con el conjunto de DataNodes como almacenamiento común. Mientras los DataNodes no están afectados por el conjunto de espacios, pues siguen almacenando bloques de ficheros independientemente del espacio al que pertenezcan; los NameNode trabajan de forma totalmente independiente entre ellos, sin requerir coordinación entre ellos.

Las características que aporta la HDFS Federation, aparte de los espacios de nombres independientes son:

- **Conjunto de bloques exclusivos:** es un conjunto de bloques que pertenecen exclusivamente a un espacio de nombres y, por lo tanto, administrados por un solo NameNode. De esta manera el fallo en un NameNode no evitaría a los DataNodes de seguir sirviendo bloques de otros espacios. Al conjunto de espacio de nombres y bloques exclusivos se le llama volumen del espacio de nombres.
- **Escalabilidad de espacios horizontales:** añadir datos en un sistema HDFS con un solo NameNode puede tener escalabilidad horizontal en el almacenamiento pero no si hablamos de los metadatos. Al tener varios NameNodes habrá una mejor escalabilidad en cuanto a los espacios.
- **Rendimiento:** por la misma razón, las peticiones de lectura y escritura serán servidas más rápidamente al tener que procesar menos información los NameNodes.
- **Aislamiento:** al tener varios espacios de nombres y cada uno con su propio NameNode, se puede configurar diferentes categorías e incluso usuarios para aislarlos en distintos espacios, de manera que no interfieran entre ellos.

En un principio puede parecer que esta nueva característica también es una buena manera de ofrecer alta disponibilidad en el sistema, pero nada más lejos de la realidad. Aunque haya varios NameNodes y la caída de uno no implica la inaccesibilidad de todo los datos almacenados en HDFS -solamente la de los

Hadoop

bloques almacenados en su espacio- sí que hay pérdida de datos y sigue siendo un punto de fallo único en el sistema.

4.2.2. MAPREDUCE 2.0

MapReduce 2.0 -o MRv2- es la capa que más cambios ha sufrido con la segunda versión de Hadoop. Se ha mantenido todas las características que identifican a MapReduce a nivel de usuario: las fases de un proceso; pero se ha renovado por completo su arquitectura y los servicios que la componen.

4.2.2.1. ARQUITECTURA YARN

YARN es un motor de gestión de recursos y aplicaciones o procesos distribuidos y es la principal adición de Hadoop 2.0. Actualmente prácticamente solo lo usa MapReduce pero en un futuro puede ser usado para otros *frameworks* para gestionar las aplicaciones. La principal idea detrás de YARN es la de hacer una gestión más óptima de los recursos de un clúster a la hora de realizar un proceso MapReduce.

Hasta ahora el JobTracker realizaba dos funciones destacadas: la gestión de recursos y la planificación y monitorización de los trabajos. Con YARN, el JobTracker deja paso a dos servicios nuevos que se encargan cada uno de una de estas tareas y aparece también el NodeManager. Los servicios trabajan sobre Containers, un concepto abstracto que se utiliza para agrupar los diversos recursos de un sistema -procesadores, memoria, disco, red, etc.- en una misma noción.

- **ResourceManager:** es un servicio global para todo el clúster y que se encarga de gestionar los recursos del sistema. Tiene dos partes:
 - **Scheduler:** es el responsable de asignar los recursos a cada aplicación. Está diseñado específicamente para realizar solamente tareas de planificación, por lo que no es encarga de monitorizar ni relanzar aplicaciones caídas. Se encarga de dividir los recursos del clúster a través de diferentes colas, aplicaciones y puede tener varias implementaciones de su política -modificables a modo de *plug-in*-, como el *CapacityScheduler* o el *FairScheduler*.
 - **ApplicationsManager:** se responsabiliza de aceptar las peticiones de creación de trabajos y de crear y asignar un ApplicationMaster a cada una de ellas.
- **ApplicationMaster:** es un servicio único para cada aplicación -es decir, para cada trabajo MapReduce-. Su principal cometido es el de negociar con el ResourceManager la gestión de los recursos y la de trabajar con los NodeManagers para ejecutar y monitorizar los trabajos.
- **NodeManager:** es un agente que se ejecuta en cada máquina y es el responsable de los Containers. Se encarga de gestionar los recursos asignados al Container y reportar su estado al Scheduler.

El proceso que siguen estos servicios a la hora de crear un proceso MapReduce se puede ver en la *Figura 18* y sigue los siguientes pasos:

1. El usuario ejecuta un proceso y se crea una petición al ApplicationsManager para crear un proceso.
2. El ApplicationsManager habla con el NodeManager para crear un ApplicationMaster.
3. El NodeManager crea el ApplicationMaster.
4. El ApplicationMaster negocia con el Scheduler del ResourceManager los recursos que serán asignados a los Containers.
5. Una vez sabe los recursos, en ApplicationMaster se comunica con los NodeManagers correspondientes para que lancen el proceso MapReduce en un Container con una máquina virtual de Java.
6. El NodeManager lanza y monitoriza el proceso MapReduce sobre un Container.
7. Los Container van reportando su estado al ApplicationMaster, que se encarga de coordinar los procesos.

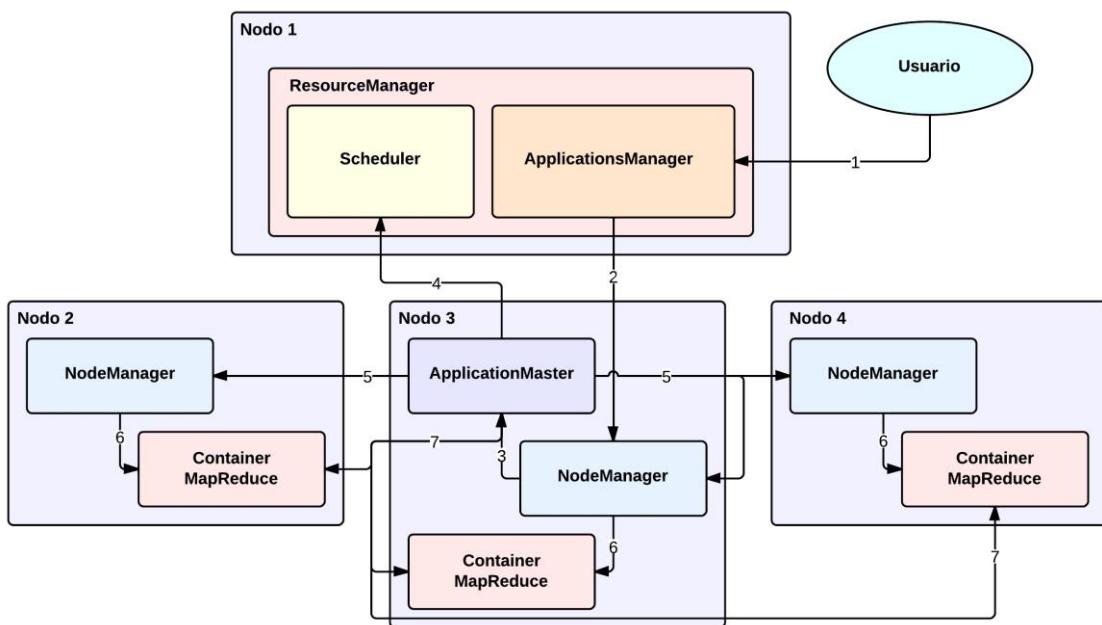


Figura 18: Proceso de creación y administración de un proceso MapReduce con YARN.

4.2.2.2. NUEVA API

Con MRv2 también se ha cambiado la API antigua -con el nombre mapred- a una nueva API más fácil de usar -llamada mapreduce-. La filosofía sigue siendo la misma, es decir implementar las interfaces que ejecutarán los procesos de las diferentes fases del trabajo MapReduce. En este punto se comentan algunos de los cambios más significativos de la nueva API.

Para la interfaz Mapper se ha cambiado la firma de la función map:

```
void map(Ki key, Vi value, Context context)
```

La clase Context sustituye al OutputCollector y al Reporter y asume sus funciones: la de escribir la salida -con la llamada context.write(K,V)- y la de reportar el funcionamiento y el estado del proceso. También permite la configuración del Mapper, reemplazando el método configure.

En la interfaz Reducer -y también en el Combiner- la clase Context sustituye también el OutputCollector:

```
void reduce(Ki key, Iterable <Vi> values, Context context)
```

Los valores de la clave también se pasan de forma distinta, ahora se usa una clase Iterable en lugar del Iterator de la API antigua.

4.2.2.3. COMPATIBILIDAD CON MAPREDUCE 1.0

MRv2 mantiene la compatibilidad de API a la hora de programar aplicaciones de manera que las aplicaciones de MRv1 deberían ser completamente funcionales sin tener que realizar cambios. Además, también se aprovechan de las características de YARN ya que a nivel de arquitectura están en diferentes niveles.

5. HERRAMIENTAS HADOOP

En este apartado se describen el resto de proyectos que completan el ecosistema Hadoop, ofreciendo una buena cantidad de herramientas y aplicaciones para distintos usos. Se han clasificado según el uso que tienen en una arquitectura Big Data, como la explicada en *3.5. Arquitectura Big Data*.

Las herramientas que se explican en este apartado son las que en el punto *1.3. Trabajo conjunto* están asignadas al autor de esta memoria. Todas las herramientas que aparecen en este apartado pertenecen a proyectos *open source* de Apache.

5.1. UTILIDADES

Las utilidades y herramientas descritas en este punto no se pueden clasificar en ninguna de las fases de una arquitectura Big Data. Sin embargo son proyectos esenciales ya que ofrecen soporte y funcionalidades a los desarrolladores de aplicaciones Hadoop de cualquiera de las etapas definidas y están, sobretodo, pensadas para la comunicación y sincronización de procesos.

5.1.1. AVRO



Avro es un sistema para la serialización de datos y uno de los principales métodos para transportar información entre las herramientas y aplicaciones Hadoop. Entre otras características, Avro provee a las aplicaciones que la utilizan de un formato de serialización binario, compacto y de rápido transporte siendo compatible con estructuras de datos complejas; un método para escribir datos de forma persistente a través de ficheros; y la implementación del protocolo RPC (Remote Procedure Call), para la comunicación entre ordenadores con una arquitectura maestro-esclavo. Las librerías que ofrece Avro son de uso muy sencillo ya que no requieren de generación de código ya sea para leer o escribir datos o para usar una comunicación de tipo RPC.

El funcionamiento de Avro es muy sencillo, al serializar datos Avro crea un esquema JSON (*Figura 19*) que lo acompañará durante todo el “trayecto”. De esta manera cuando estos datos sean leídos, la aplicación destino tendrá toda la información necesaria para interpretar los datos que le llegan. Esto hace a Avro totalmente compatible con lenguajes dinámicos.

```
{"namespace": "ejemplo.avro",
"type": "record",
"name": "Usuario",
"fields": [
    {"name": "nombre", "type": "string"},
    {"name": "numero_favorito", "type": ["int", "null"]},
    {"name": "color_favorito", "type": ["string", "null"]}
]}
```

Figura 19: Ejemplo de un esquema JSON para un tipo de datos de nombre “Usuario” con tres valores: nombre de tipo string, número favorito de tipo pareja de entero con posibilidad de ser null y color favorito de tipo string y también con posibilidad de ser null.

Cuando se escriben ficheros con datos serializados pasa exactamente lo mismo, su esquema es almacenado para que cualquier aplicación que lea los datos sepa cómo interpretarlos y deserializarlos

Herramientas Hadoop

mediante Avro. Con la comunicación RPC pasa exactamente lo mismo, los esquemas de los datos a serializarse se intercambian en el proceso de *handshaking* al inicio de la comunicación.

El tener siempre el esquema de datos junto con la información serializada es una ventaja en las situaciones en las que la aplicación destino espera recibir otro tipo de datos o esquema distinto al obtenido. Al tener los dos esquemas, el recibido y el esperado, la aplicación puede decidir los siguientes pasos a realizar.

Como se ha comentado, para la definición de los esquemas de datos Avro utiliza JSON, un tipo de codificación muy común y utilizada que lo hace altamente compatible con muchos lenguajes de programación. La especificación del esquema se basa en la declaración de objetos JSON que definen los datos a través de tipos primitivos -null, bool, int, long, float, double, bytes y string- y con la posibilidad de añadir seis tipos complejos: [20]

- **Records:** son objetos de nombre “*record*” que tienen los siguientes atributos:
 - **Name:** nombre del objeto.
 - **Namespace:** espacio que completa el nombre (en el ejemplo de la Figura 19: ejemplo.Avro.usuario).
 - **Doc:** documentación del esquema para el usuario (es opcional).
 - **Aliases:** una lista de nombres alternativos para el objeto (es opcional).
 - **Fields:** una lista de objetos JSON que tendrán los atributos de nombre, documentación, tipo y valor por defecto.
- **Enums:** de nombre “*enum*”, sirven para representar enumeraciones y que tienen los siguientes atributos que los definen:
 - **Name:** nombre del enum.
 - **Namespace:** espacio que completa el nombre.
 - **Doc:** documentación del esquema para el usuario (es opcional).
 - **Aliases:** una lista de nombres alternativos para el objeto (es opcional).
 - **Symbols:** una lista de objetos JSON de tipo string.
- **Arrays:** sirven para representar listas y sólo tienen un atributo “*items*”, que indica el tipo de lista. Su nombre identificativo es “*array*”.
- **Maps:** de nombre “*map*” y solo aceptan un atributo “*values*”, que es el esquema de los valores del mapa. Las claves se presuponen de tipo string.
- **Unions:** el tipo “unión” se representan mediante una lista JSON y representan la disyunción de tipos. Por ejemplo, [“string”, “null”] representa un esquema que tanto puede ser de tipo string o null. Para los tipos complejos no puede haber dos esquemas del mismo tipo salvo en los casos de los record, enum y fixed. Los tipos unions tampoco pueden contener otras unions.
- **Fixed:** representan el tipo binario, su nombre identificativo es “*fixed*” y aceptan los siguientes atributos:
 - **Name:** nombre del fixed.
 - **Namespace:** espacio que completa el nombre.
 - **Aliases:** una lista de nombres alternativos para el objeto (es opcional).
 - **Size:** el tamaño en bytes de cada valor.

Para definir uno de estos esquemas complejos basta con especificar el nombre del tipo en el atributo *type*, como en la *Figura 19* para el tipo record o la *Figura 20* en el caso del enum.

```
{"type": "record",
"name": "Palo",
"symbols": [ "PICA", "CORAZON", "TREBOL", "DIAMANTE" ]}
```

Figura 20: Ejemplo de un esquema para el tipo enum.

Herramientas Hadoop

Actualmente existen APIs oficiales para los lenguajes de programación Java, C, C# y C++, así como librerías Java para adaptar la comunicación de programas MapReduce con Avro.

5.1.2. ZOOKEEPER



ZooKeeper es un servicio centralizado que se encarga de administrar y gestionar la coordinación entre procesos en sistemas distribuidos. El objetivo de Apache con ZooKeeper es el de librarse a los desarrolladores de la tarea de implementar funciones de mantenimiento entre sus procesos, como la sincronización entre estos, y ofrecer alta disponibilidad a través de servicios redundantes.

ZooKeeper permite la coordinación de procesos a través de un espacio de nombres compartido de estructura jerárquica, parecido a un sistema de ficheros, donde cada nodo recibe el nombre de *znode*. Los *znodes* son muy parecidos a los de un sistema de ficheros normal y corriente, cuentan con un *path* o camino con los elementos separados por la barra “/” y cada uno tiene un parente que es el *znode* anterior -salvo la raíz “/” que es su propio parente-. Los *znodes* tampoco pueden ser eliminados si tienen hijos, tal y como pasa con los sistemas de ficheros comunes.

La principal diferencia entre un *znode* y un nodo de un sistema de ficheros es que los primeros pueden tener datos asociados, de manera que hasta los nodos directorios pueden contener información como si fueran ficheros normales y corrientes. Además los datos que contiene cada *znode* así como los metadatos -información del estado, configuración, localización...- está limitada para asegurarse de que ZooKeeper no es usado como si se tratara de un sistema de ficheros.

Para asegurarse de que tenga alta disponibilidad los servicios están replicados a distintos nodos del clúster -se convierten en servidores ZooKeeper- donde está instalado y se mantienen sincronizados a través de logs de transacciones e imágenes de estado en un almacenamiento permanente (como un fichero o una base de datos). Un proceso cliente del servicio ZooKeeper se conecta únicamente a uno de los servidores, obteniendo de él una clave de sesión y desde el que realiza todas las peticiones a través de una conexión TCP. En caso de que la conexión se pierda, el cliente trata de conectarse a otro de los servidores, renovando la sesión.

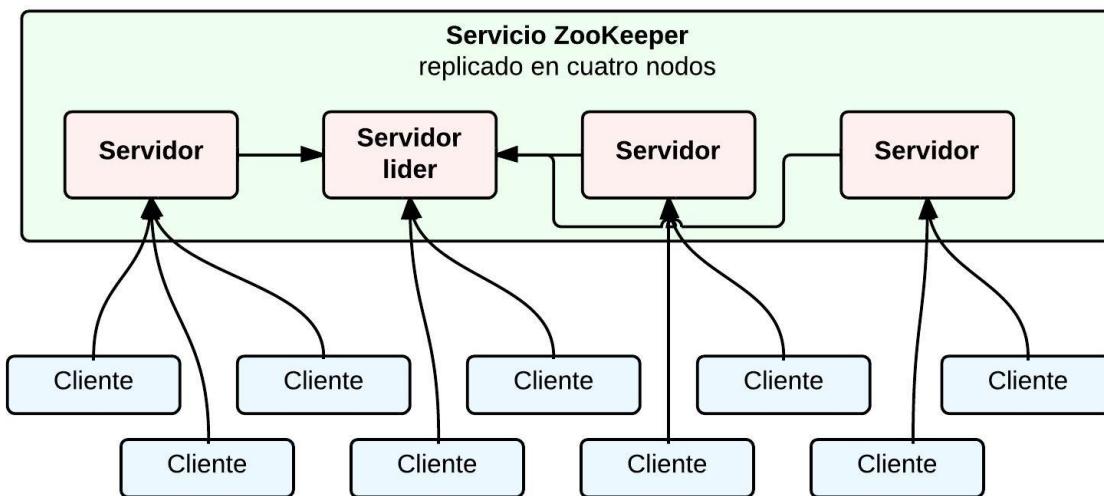


Figura 21: Arquitectura del servicio ZooKeeper. Los servicios están replicados en cuatro nodos y sirven peticiones a varios clientes. Hay un servidor líder que es el que sincroniza a los demás servidores.

La principal razón para evitar que sea usado como un sistema de ficheros para almacenar datos con un volumen considerable es porque los datos se mantienen en la memoria del sistema. Gracias a esto se consigue un rendimiento muy elevado a la hora de servir las peticiones de los procesos cliente pero

Herramientas Hadoop

también limita la cantidad de datos a la memoria del sistema (generalmente del orden de pocos gigabytes).

Para mantener la coherencia entre los datos, cada vez que un cliente hace una petición de escritura a uno de los servidores, este lo comunica a los demás y no contesta la petición hasta llegar a un acuerdo con los demás. Las peticiones de lectura se sirven desde el propio servidor que recibe la petición, ya que no existe el peligro de romper la coherencia de datos. Debido a esto, cuantos más servidores haya mayor será el rendimiento de las lecturas, mientras que menor será el de las escrituras (al tener que llegar a un consenso entre todos).

De esta manera, los desarrolladores de aplicaciones distribuidas tienen una arquitectura y un recurso para mantener sincronizados los servicios de un mismo clúster.

5.1.3. SOLR



Apache Solr es un motor de búsqueda basado en el Apache Lucene, escrito en Java y que facilita a los programadores el desarrollo de aplicaciones de búsqueda. Lucene ofrece indexación de información, tecnologías para la búsqueda así como corrección ortográfica, resaltado y análisis de información, entre otras muchas características. [21]

Una arquitectura típica de Solr sería la de la Figura 22. Se cuenta con un servidor web, para que los usuarios puedan interactuar y realizar distintos tipos de búsquedas -páginas de tiendas en línea, por ejemplo-, con conexión directa con Solr y que consulta datos mediante este en Hadoop. El servidor web puede ser una de las máquinas con servicios de Hadoop ejecutándose en él o un servidor exclusivamente dedicado a servir peticiones de los usuarios.

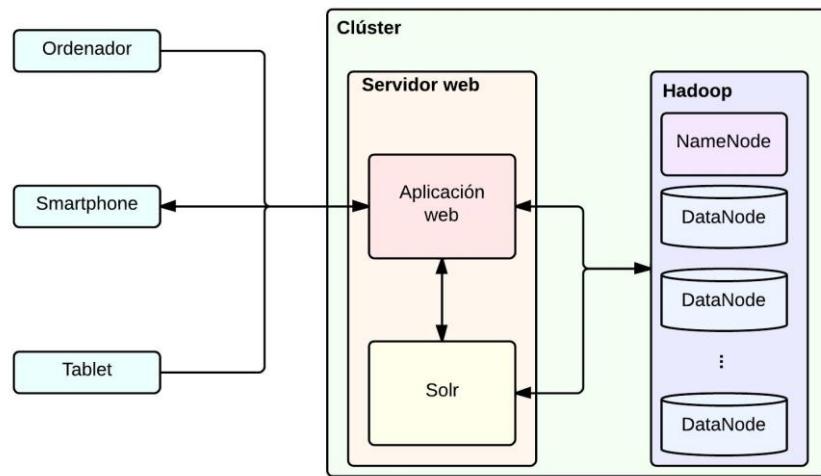


Figura 22: Arquitectura de Solr trabajando con Hadoop.

Solr también permite nativamente replicación y distribución (aunque estas características ya vienen implícitas con Hadoop) y una combinación de ellas mediante ZooKeeper.

Solr es totalmente extensible en la fase de visualización ya que las consultas son RESTful; es decir, que son simples peticiones HTTP a través de una URL y sus respuestas son documentos estructurados - generalmente XML pero también acepta otros formatos como JSON o CSV entre otros-.

Solr cuenta básicamente con dos fases: indexación y consulta. En la fase de indexación se le indican los documentos y se define un esquema sobre el que Solr estará trabajando. El esquema le indica los contenidos del documento que indexará para facilitar y aumentar el rendimiento de las búsquedas. En la fase de consulta simplemente recibe una pregunta y la responde usando el índice generado con el esquema.

5.2. RECOLECCIÓN DE DATOS

Estas herramientas son de las que más han evolucionado gracias a la aparición de Hadoop y a la popularización de sistemas de almacenamiento NoSQL; todo ello fruto de la necesidad de tratar datos no estructurados.

Una de las ventajas que ofrecen las herramientas de recolección de datos es la flexibilidad que tienen; tanto a la hora de configurarse y adaptarse a distintos orígenes y destinos de datos como para trabajar independientemente del sistema donde están montados (es decir, no necesitan Hadoop de manera imperiosa).

5.2.1. CHUKWA



Chukwa [22] es una herramienta principalmente pensada para trabajar sobre logs y realizar análisis. Está construido por encima de HDFS y MapReduce, por lo que hereda su escalabilidad y robustez. La principal motivación a la hora de desarrollar Chukwa fue precisamente que Hadoop no termina de trabajar bien con sistemas de logs ya que está más optimizado para trabajar con pocos ficheros de mayor tamaño; a contraposición de los sistemas de logs, que son directorios con un gran número de ficheros pequeños.

Para cumplir con este propósito Chukwa ofrece un sistema flexible para recolectar datos de forma distribuida y para realizar su procesamiento que, a la vez, es adaptable a las nuevas tecnologías de almacenamiento que van apareciendo (5.3. Almacenamiento).

La arquitectura de Chukwa se compone de cuatro componentes:

- **Agentes:** los procesos que se encargan de capturar datos.
- **Colectores:** reciben los datos de los agentes y lo escriben en un almacenamiento permanente.
- **Trabajos MapReduce** para trabajar con los datos.
- **HICC:** es una interfaz web para visualizar datos.

Con esta arquitectura Chukwa abarca prácticamente todas las capas de un sistema Hadoop ya que se aprovecha de la base de este -HDFS para el almacenamiento y MapReduce para el procesado- y le añade utilidades que le faltan para completar la arquitectura. De todas maneras Chukwa está considerada por muchos una herramienta de recolección de datos gracias a que tiene un diseño flexible y muchas veces se adapta solamente para este motivo.

5.2.1.1. AGENTES Y ADAPTADORES

Los agentes son los encargados de capturar datos y enviarlos a un colector. Para poder capturar datos desde diversas fuentes dinámicamente, ya que las orígenes de datos ofrecen datos que pueden variar su formato dependiendo de la máquina y el momento en que se capturan, los agentes tienen la capacidad de ser configurados dinámicamente con módulos que se ejecutan dentro del proceso y que son los verdaderos responsables de la recolección de datos.

Los datos emitidos por los adaptadores se envían por paquetes llamados *Chunks*. Estos paquetes están formados por una secuencia de bytes -los datos a enviar- y metadatos -que describen los datos-. Los metadatos suelen estar generados por el agente pero hay un par de ellos que si es necesario que los defina el usuario: el nombre del clúster y los tipos de los datos que se están enviando. En la *Tabla 16* se listan los metadatos de un Chunk.

Campo	Significado	Generado por
Source	El origen -host- de los datos del chunk	Agente
Cluster	Host del clúster asociado	Usuario
Datatype	formato de los datos	Usuario
Sequence ID	Cantidad de bytes enviados desde el inicio de la transmisión (incluyendo los del chunk actual)	Adaptador
Name	Nombre del origen de datos	Adaptador

Tabla 16: Posibles campos de los metadatos de un chunk.

El campo Sequence ID se usa además para continuar la transmisión de datos si se produce algún fallo.

Los adaptadores acostumbran a estar directamente relacionados con una fuente de datos y hay de diversos tipos:

- **FileAdaptor**: envía un fichero entero en un solo chunk.
- **LWFTAdaptor**: envía un fichero de forma binaria -sin importar el contenido- en varios Chunks.
- **FileTailingAdaptor**: envía un fichero ignorando su contenido y realizando *tails* continuamente.
- **CharFileTailingAdaptorUTF8**: hace lo mismo que el FileTailingAdaptor pero se asegura de que los Chunks terminan en un salto de línea.
- **DirTailingAdaptor**: no emite datos por sí mismo. Recibe por parámetro el nombre de otro adaptador y un directorio. Recorre recursivamente el directorio y sus subdirectorios y utiliza el adaptador especificado para enviar los datos.
- **ExecAdaptor**: ejecuta un programa cada cierta cantidad de tiempo -configurable con un parámetro- y envía su salida.
- **UDPAdaptor**: emite los datos recibidos desde un puerto del ordenador.

Una vez un agente se está ejecutando, Chukwa ofrece una serie de comandos para controlar su comportamiento (*Tabla 17*) a través del puerto 9093 [23].

Comando	Propósito
add	Añade un adaptador a un agente
close	Cierra el socket del agente
help	Muestra una lista de los comandos disponibles
list	Lista los adaptadores que se están ejecutando
reloadCollectors	Vuelve a leer la lista de colectores
stop	Para el adaptador indicado de manera abrupta
stopAll	Para todos los adaptadores de manera abrupta
shutdown	Para un adaptador indicado
stopagent	Para el agente

Tabla 17: Comandos que puede recibir un agente de Chukwa.

5.2.1.2. COLECTORES

Los datos de los agentes son enviados a los colectores vía HTTP y estos son los encargados de escribir en HDFS los chunks. A medida que van recibiendo chunks, los colectores los escriben en un fichero sink dentro de HDFS, que es un fichero con la secuencia de datos serializados. Periódicamente los colectores cierran y renombran el sink, marcando los datos como disponibles para ser procesados y empiezan a escribir otro fichero de cero.

5.2.2. FLUME



Flume es una herramienta distribuida para la recolección, agregación y transmisión de grandes volúmenes de datos. Ofrece una arquitectura basada en la transmisión de datos por streaming altamente flexible y configurable pero a la vez simple. Al tener un origen de datos configurable, se adapta prácticamente a cualquier tipo de situación: monitorización de logs, descarga de información de redes sociales o mensajes de correo electrónico, entre muchas otras. Los destinos de los datos también son altamente configurables de manera que el uso de Flume no va ligado exclusivamente con HDFS o incluso Hadoop.

La arquitectura de Flume [24] está basada en agentes, que son procesos encargados de recolectar datos y enviarlos a su destino. A su vez, el flujo de datos viene determinado por una unidad llamada evento que, como pasaba con los chunks de Chukwa, está formado por datos y metadatos. Un agente tiene tres componentes (Figura 23):

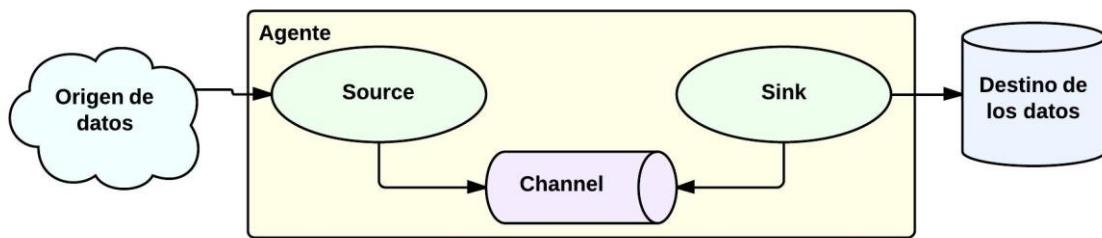


Figura 23: Componentes de un agente de Flume.

- **Source:** el source es el encargado de recibir los datos, convertirlos en eventos y escribirlos en el channel. Hay varios tipos de sources dependiendo de cuál es el origen de datos:
 - **Avro source:** recibe los datos a través de un puerto mediante la utilidad de serialización de datos Avro. Los datos que recibe ya son eventos.
 - **Thrift source:** utiliza otra herramienta externa, Thrift, para ir recibiendo datos.
 - **Exec source:** ejecuta un comando Unix y espera a que genere datos de manera continua. Captura estos datos y los escribe en un channel. En caso de que el proceso Unix fallara y dejase de ejecutarse, el Exec source dejaría de emitir datos.
 - **Netcat source:** escucha a un puerto y convierte cada línea de texto que recibe por ese puerto en un evento. Recibe este nombre porque su comportamiento es muy parecido al del comando de Unix netcat.
 - **Sequence Generator Source:** genera eventos de manera continua con un contador que empieza en 0 e incrementa de uno en uno. Es usado mayormente para realizar pruebas.
 - **HTTP source:** recibe eventos a través del protocolo HTTP mediante los métodos GET (se aconseja que sea únicamente para experimentación) y POST.
 - **Custom Source:** es un source implementado por el usuario y por lo tanto totalmente personalizable.
- **Channel:** es un almacenamiento pasivo que recibe los eventos del source y espera a que el sink lo consuma, es decir que los sources añaden eventos mientras que los sinks los retiran. Dependiendo del tipo de tolerancia a fallos y del rendimiento que se desea hay distintos tipos de channels:

Herramientas Hadoop

- **Memory channel:** los eventos se almacenan en una cola en memoria con un tamaño máximo. Es el channel ideal para obtener rendimientos elevados pero no tiene demasiada tolerancia a errores ya que los datos se pierden al reiniciar el agente.
- **JDBC channel:** los eventos se almacenan en una base de datos permanente -es compatible con Derby-. Al usar un sistema de bases de datos no se obtiene un rendimiento tan elevado pero sí que permite la recuperación de eventos bajo fallos. Un contraefecto de este channel es
- **File channel:** ofrece las mismas características que un channel JDBC pero almacena los eventos en ficheros temporales.
- **Custom channel:** es un channel implementado por el usuario y totalmente personalizable.
- **Sink:** se encarga de leer los eventos del channel y dependiendo de su tipo enviarlo a diferentes sitios:
 - **HDFS sink:** escribe los eventos en un fichero HDFS. Hay varios parámetros configurables: ruta del fichero, nombre y extensión, cuando se cierra cada fichero o cuántas réplicas debe tener mínimo cada bloque.
 - **Logger sink:** escribe los events en un log con el nivel de INFO.
 - **Avro sink:** envía los eventos al host y puertos indicados mediante Avro.
 - **Thrift sink:** como el anterior pero usando Thrift.
 - **File roll sink:** escribe los eventos en un fichero del sistema de ficheros local.
 - **Null sink:** descarta todos los eventos.
 - **Custom sink:** es un sink implementado por el usuario y totalmente personalizable.

Además de estos componentes del agente, Flume también ofrece una utilidad para facilitar la captura y generación de eventos llamado cliente. Un cliente es un proceso que trabaja en el origen de datos, desde fuera del agente, y que se encarga de recolectar los datos para enviarlo al agente.

Gracias a la flexibilidad de Flume, es posible interconectar varios agentes -por ejemplo, usar un Avro Sink con un Avro Source- formando diversas configuraciones que añaden nuevas características a una aplicación.

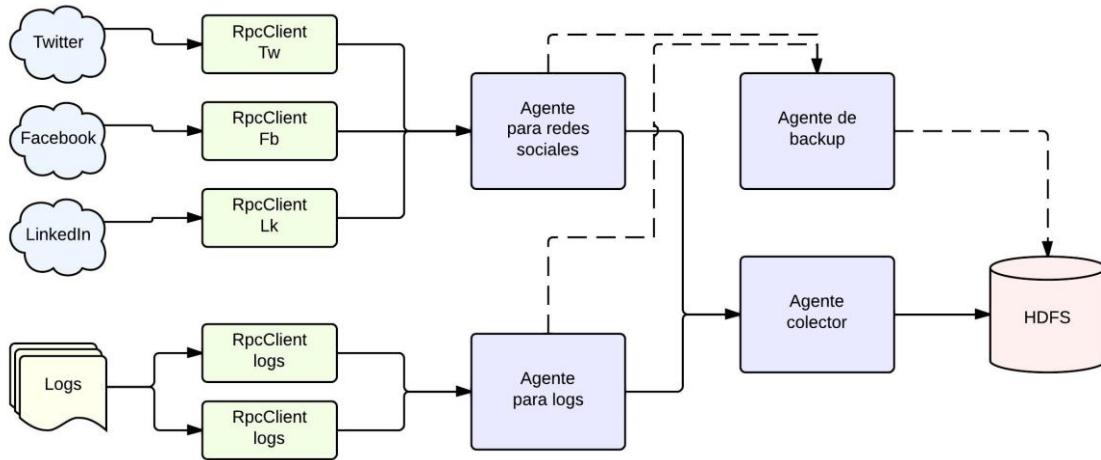


Figura 24: Arquitectura Flume que descarga información de las redes sociales Twitter, Facebook y LinkedIn además de leer ficheros de log.

En la *Figura 24* vemos un ejemplo de interconexión de diversos agentes junto con muchos clientes. En esta arquitectura se usa varios clientes para descargar información de distintas fuentes de información - redes sociales y logs-. Los clientes envían los eventos ya formateados a dos agentes -uno para redes

Herramientas Hadoop

sociales y otro para logs- para distribuir la carga y se usa un tercer agente a modo de colector para juntar todos los eventos y escribirlos finalmente en HDFS. Además, se tiene un último agente para obtener alta disponibilidad en el agente de colector. Este agente está activo, esperando eventos como los demás, pero los otros componentes de la arquitectura sólo le enviarán información cuando el agente activo se desconecte. Otro ejemplo de arquitectura lo podemos observar en la Figura 25; en este caso el agente que recibe datos de los clientes hace de balanceador de carga, repartiendo la distribución de datos entre otros dos agentes. Estos, además, escriben mediante dos sinks cada uno -para HDFS y para Hive-.

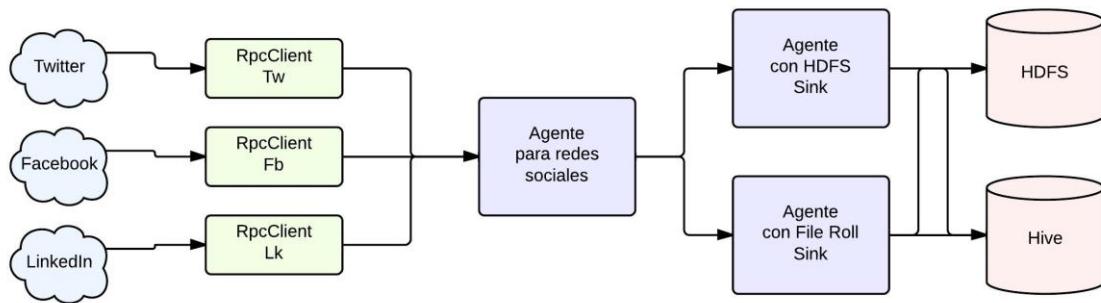


Figura 25: Arquitectura Flume que usa un agente para balancear la carga de datos recibidos. Los agentes finales escriben tanto en HDFS como en Hive.

En la *Figura 26* observamos la configuración de un agente de Flume con los siguientes componentes:

- **Source:** con el identificador r1, es de tipo avro y escucha en localhost en el puerto 11011 para recibir eventos avro.
- **Channel:** con identificador c1, es de tipo memory, y tiene una capacidad para almacenar 1000 eventos y enviarlos de 100 en 100.
- **Sink:** con identificador k1, escribe en HDFS un fichero de tipo DataStream, dentro directorio que se le indica -en los metadatos del evento- con el parámetro outputDirectory y a su vez dentro de otro subdirectorio que tiene por nombre una fecha -usando un timestamp recibido también con los metadatos del evento-. El fichero tendrá por nombre tweets-XXXXXX.txt; donde XXXXXX es un número identificativo puesto por el propio sink.

Como se puede observar, un agente puede tener más de un componente para cada tipo y en el propio fichero de configuración es donde se realiza la interconexión de los componentes. También

```

a1.channels = c1
a1.sources = r1
a1.sinks = k1

a1.channels.c1.type = memory
a1.channels.c1.capacity = 1000
a1.channels.c1.transactionCapacity = 100

a1.sources.r1.channels = c1
a1.sources.r1.type = avro
a1.sources.r1.bind = localhost
a1.sources.r1.port = 11011

a1.sinks.k1.type = hdfs
a1.sinks.k1.channel = c1
a1.sinks.k1.hdfs.path = %{outputDirectory}
a1.sinks.k1.hdfs.fileType=DataStream
a1.sinks.k1.hdfs.filePrefix = %Y-%m-%d/tweets
a1.sinks.k1.hdfs.fileSuffix = .txt
  
```

Figura 26: Configuración de un agente Flume.

5.3. ALMACENAMIENTO

Aunque Hadoop ya cuenta con su propio sistema de almacenamiento, HDFS, existen varios proyectos que complementan, e incluso trabajan conjuntamente, con el sistema de ficheros para aumentar las características y la flexibilidad de Hadoop entorno a las necesidades de sus usuarios.

5.3.1. CASSANDRA



Cassandra

Cassandra es una base de datos NoSQL, mayormente desarrollada por Datastax aunque empezó como un proyecto de Facebook, escrita en Java y open-source -también es un proyecto Apache-. Entre sus características se encuentra la de ofrecer alta disponibilidad de los datos junto con una gran capacidad para escalar linealmente, además de ser tolerante a fallos -no tiene un punto de fallo único- y compatible con hardware de bajo presupuesto o infraestructuras en la nube -cloud-.

5.3.1.1. MODELO DE DATOS

Cassandra es una base de datos distribuida orientada a columnas -inspirada en BigTable de Google y en Dynamo de Amazon-, pensada especialmente para datos semi-estructurados (aunque también se puede adaptar a los estructurados y no estructurados) y que contiene varios espacios de claves con el siguiente esquema:

- Los espacios de claves están formados por un número determinado de familias de columnas -también conocidas como supercolumnas-.
- Cada familia de columnas está identificada con una clave y tiene por valor un grupo indeterminado de filas.
- Cada fila, también identificada por una clave, tiene por valor distintas columnas (que pueden añadirse dinámicamente y por lo tanto no hay un número definido de columnas). Las columnas de cada fila son independientes de las que haya en las otras filas.
- Cada columna es una tupla {nombre, valor, timestamp}. Cada vez que se modifica un valor también se cambia el timestamp, por lo que se puede hacer un seguimiento de los cambios de un valor.

Este modelo de datos es propenso -y es una de las razones para las que fue diseñado- a tener los datos semi estructurados -podría ser un buen esquema para almacenar ficheros XML, HTML o JSON- y que los datos estén dispersos. Esto significa que debido a la diferencia entre fila y fila, haya una desproporción entre estas en el número de columnas. Cuando se crea el espacio de claves -que en las bases de datos tradicionales podría equivaler a una base de datos- también se debe establecer qué familias de columnas -que podrían equivaler a las tablas- habrá en ese espacio.

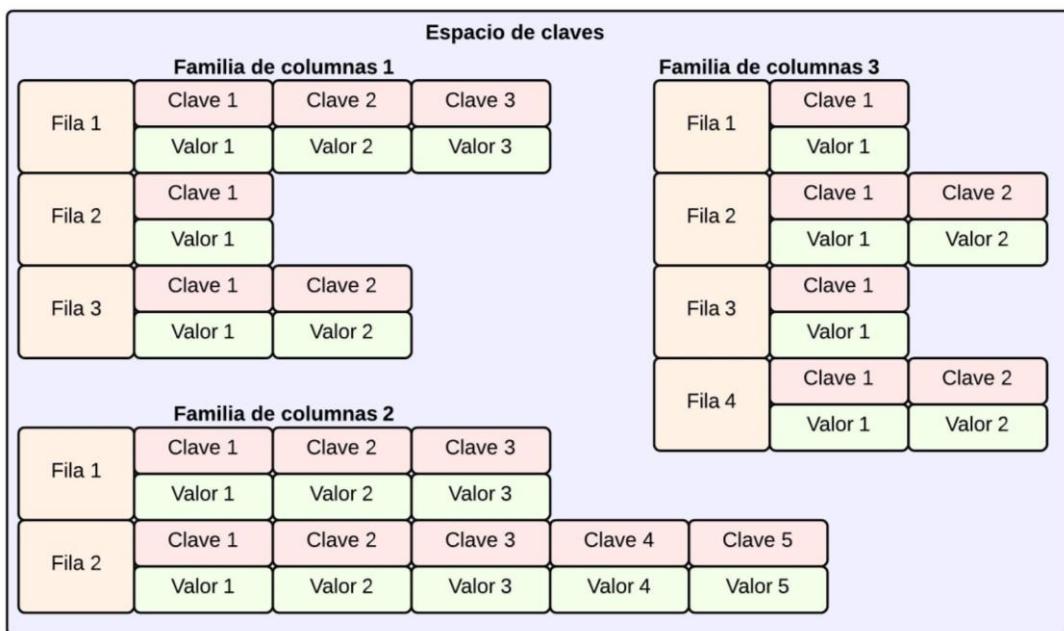


Figura 27: ejemplo de un espacio de nombres de Cassandra.

5.3.1.2. ARQUITECTURA

Su arquitectura es parecida a una red peer-to-peer (P2P) donde todos los nodos tienen la misma importancia jerárquica, sin ninguno que haga de maestro o coordinador. De esta manera todos los nodos tienen exactamente la misma función y se evita que haya un punto de fallo único en el sistema. Los datos se distribuyen entre todos los nodos y estos se sincronizan mediante el protocolo Gossip, un protocolo P2P que sirve para sincronizar información sobre el estado y la localización de los datos en cada nodo de la red. La información transferida a través de este protocolo es también almacenada en disco para poder usarla en caso de reinicio del nodo. La información es intercambiada por los nodos cada segundo, manteniendo una alta tasa de sincronización.

Cada vez que hay actividad -de escritura- en uno de los nodos este lo transcribe en su log de actividad para asegurar que haya coherencia. Los datos además son almacenados en una estructura en memoria -memtable- y, una vez se sobrepasa el tamaño de esta, son escritos en un fichero llamado SSTable.

Cada escritura es automáticamente particionada y replicada en el clúster para asegurar la alta disponibilidad de los datos. El Partitioner es el encargado de decidir cómo se distribuirán los datos y es configurable por el usuario. Por otro lado, la estrategia de réplicas determina dónde y cuántas réplicas se almacenaran en el clúster y debe ser configurada al crear el espacio de claves correspondiente.

Cuando un usuario realiza una operación de lectura o escritura sobre uno de los nodos -todos pueden recibir indistintamente peticiones-, el nodo conectado pasa a ser el coordinador para esa operación en particular. La función del coordinador es básicamente la de decidir cuáles son los nodos que deberán contestar la petición del usuario.

5.3.1.3. CASSANDRA QUERY LANGUAGE

Cassandra tiene su propio lenguaje para realizar consultas sobre los datos: Cassandra Query Language (CQL). Para hacerlo lo más usable posible, CQL es prácticamente igual que un lenguaje SQL, incluso en el

[Herramientas Hadoop](#)

concepto de tablas con filas y columnas, a pesar de que el modelo de datos de Cassandra sea tan diferente. La principal diferencia es que CQL no permite realizar consultas con joins ni subconsultas.

5.3.2. HIVE



Hive [25] es una herramienta para data warehousing que facilita la creación, consulta y administración de grandes volúmenes de datos distribuidos en forma de tablas relacionales. Cuenta con un lenguaje derivado de SQL, llamado Hive QL, que permite realizar las consultas sobre los datos. Por esta misma razón, se dice que Hive lleva las bases de datos relacionales a Hadoop. A su vez, Hive QL está construido sobre MapReduce, de manera que se aprovecha de las características de éste para trabajar con grandes cantidades de datos almacenados en Hadoop. Esto también provoca que Hive no ofrezca respuestas en tiempo real.

La arquitectura de Hive se compone de los siguientes componentes: [26]

- **Interfaz de usuario:** el método de entrada del usuario para realizar las consultas. Actualmente hay una interfaz de línea de comandos y una interfaz web.
- **Driver:** recibe las consultas y se encarga de implementar las sesiones, además de recibir también consultas vía interfaces JDBC y ODBC.
- **Compilador:** parsea la consulta y realiza análisis semánticos y otras comprobaciones de lenguaje para generar un plan de ejecución con la ayuda del metastore.
- **Metastore:** almacena toda la información -metadatos- de la estructura que mantienen los datos dentro de Hive -es decir, tiene el esquema de las bases de datos, tablas, particiones, etc.-.
- **Motores de ejecución:** se encargan de llevar a cabo el plan de ejecución realizado por el compilador

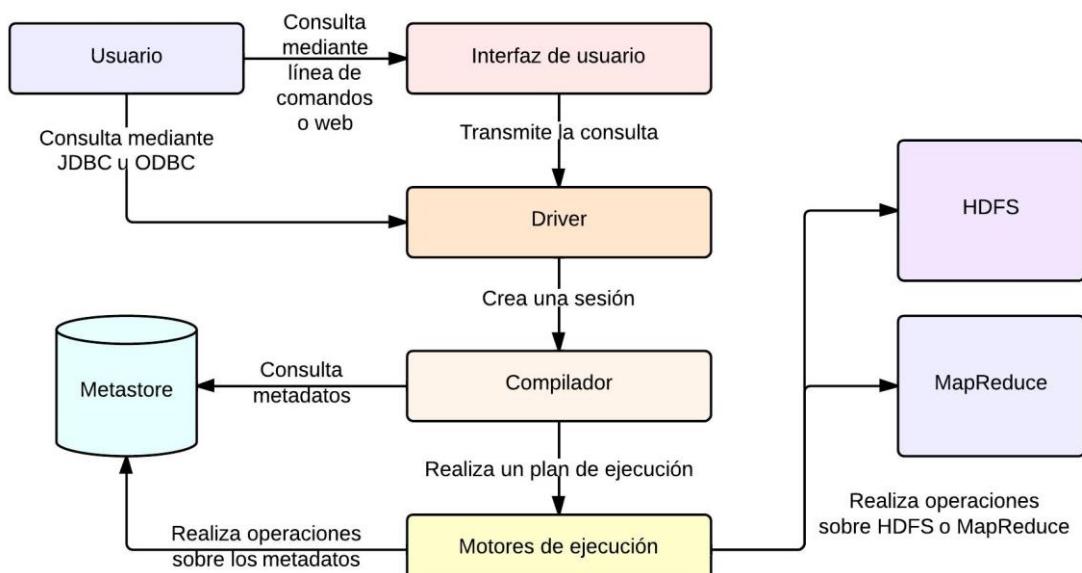


Figura 28: Arquitectura de Hive elaborada en fases o etapas.

Una vez el usuario ha introducido su consulta, el Driver la recibe y se encarga de crear una sesión y enviarla al compilador. El compilador comprueba la validez de la consulta e idea el plan de ejecución a llevar a cabo. Para realizar estas dos tareas, consultará con el metastore los metadatos necesarios para realizar una división adecuada de la consulta a realizar a trabajos MapReduce -una consulta puede necesitar más de un MapReduce, dependiendo del tipo de consulta o el esquema de los datos-. El plan es un conjunto de fases donde cada una de ellas es un trabajo MapReduce, una operación sobre los

Herramientas Hadoop

metadatos o una operación sobre HDFS. Finalmente los motores de ejecución reciben el plan y se encargan de ejecutar en el orden correcto las distintas fases. Los resultados parciales de cada fase son almacenados en ficheros temporales dentro de HDFS para que la siguiente fase pueda leerlos y para asegurarse de la consistencia de datos y evitar que otras consultas puedan leer datos no definitivos-. Al finalizar la consulta, se pasa el resultado a un fichero HDFS definitivo.

Hive cuenta con dos servicios

5.3.2.1. METASTORE Y MODELO DE DATOS

Los datos de Hive están organizados en tres categorías distintas:

- **Tablas:** son como las tablas de las bases de datos relacionales. Se permite hacer operaciones de filtrado, proyección, joins y uniones. Las tablas se almacenan en un solo directorio y se pueden crear tablas externas -con ficheros de datos ya existentes fuera del directorio-. Las filas están organizadas en columnas con un tipo asignado -tal y como pasa con las tablas relationales-.
- **Particiones:** las tablas pueden particionarse según conveniencia. Cada partición se almacena por separado, propiciando una mayor flexibilidad y rendimiento a la hora de trabajar y hacer consultas con los datos. Por ejemplo, se puede partitionar una tabla para que el contenido esté almacenado según fechas.
- **Cubos:** a su vez, las filas de cada partición pueden estar divididos en cubos -o buckets, en inglés- que se almacenan en ficheros distintos dentro del directorio de la partición.

Las columnas aceptan los tipos básicos -enteros, números decimales, cadenas de texto, fechas y booleanos-, tipos más complejos como maps o vectores y, además, da la posibilidad a los usuarios de definir sus propios tipos -programando sus propios inspectores para usar un SerDe-. Un SerDe es una utilidad para serialización y deserialización de datos; en Hive pueden usarse SerDes para leer los datos (para leer datos en JSON o XML y traducirlos al esquema de una tabla de Hive, por ejemplo).

El servicio metastore proporciona dos funcionalidades vitales para una herramienta de data warehousing: abstracción de datos y explotación de datos. De esta manera se abstrae al usuario de tener que indicar los tipos de datos, su localización y demás información cada vez que se trabaje con los objetos de Hive.

Hive tiene tres tipos de objetos definidos en los metadatos:

- **Bases de datos:** es un espacio de nombres para las tablas que puede usarse para realizar unidades administrativas. Por defecto hay una creada con el nombre “default”, que es con la que se trabaja si no se indica lo contrario.
- **Tablas:** los metadatos para la tabla incluye las columnas y sus tipos, información sobre el propietario y la relacionada con el SerDe para leer y escribir los datos. También incluye la localización de los datos e información de los cubos.
- **Particiones:** las particiones pueden tener su propia estructura de columnas, SerDe y almacenamiento. De esta manera se consigue aislar los cambios de las particiones.

5.3.2.2. HIVE QUERY LANGUAGE

Hive QL es prácticamente una copia de la sintaxis de los lenguajes SQL, usando la misma estructura para la creación de tablas, la carga de datos o al realizar selecciones de datos dentro de las tablas, entre otras. Adicionalmente permite que los usuarios creen funciones propias, en forma de script MapReduce.

5.4. DESARROLLO DE PROCESOS PARA EL TRATAMIENTO DE DATOS

Hadoop también dispone de un buen número de herramientas para tratar los datos. Ya sea para realizar esquemas de trabajos o workflows o para gestionar el flujo de los datos en un caso de uso -dataflows-. También cuenta con herramientas que ofrecen librerías y funciones ya implementadas para trabajar con los datos.

5.4.1. MAHOUT



Mahout es una librería Java que contiene básicamente funciones de aprendizaje y que está construida sobre MapReduce. [27] Al usar MapReduce está pensada para trabajar con grandes volúmenes de datos y en sistemas distribuidos -aunque también está diseñado para funcionar en sistemas no Hadoop y no distribuidos [28]-. Otro de los objetivos de Mahout es contar con el apoyo de la comunidad, para ir ofreciendo cada día más y más funciones e ir depurando los errores que van surgiendo. Actualmente está en continuo desarrollo pero cuenta con una base muy sólida.

Actualmente Mahout da soporte a cuatro casos de uso:

- **Minería de recomendaciones:** aprende a través del comportamiento de los usuarios qué es lo que les podría gustar.
- **Clustering:** busca agrupaciones dado un conjunto de documentos para poder diferenciarlos y clasificarlos.
- **Clasificación:** aprende de un grupo de documentos ya categorizados cómo son los documentos pertenecientes de cada categoría. De esta manera puede clasificar en un nuevo documento.
- **Minería de ítems frecuentes:** dado un grupo de ítems identifica cuáles son los más comunes en frecuencia de aparición.

Todos estos casos de uso forman parte de problemas de aprendizaje típicos. Entre las funciones cuenta con algunos de los algoritmos más conocidos en el mundo del aprendizaje: los clasificadores Naive Bayes y Random Forest, arboles de decisión o funciones de clustering como k-means o Fuzzy k-means.

Usar mahout es tan sencillo como ejecutar un proceso Java Mahout indicándole que algoritmo usar, la entrada de datos y un directorio donde escribir la salida. Por esto mismo, el usuario no tiene que implementar una sola línea de código para hacer uso de esta herramienta.

5.4.2. OOZIE



Oozie es un planificador de workflows para sistemas que realizan trabajos o procesos Hadoop. Proporciona una interfaz de alto nivel para el usuario no técnico o no experto y que gracias a su abstracción permite a estos usuarios realizar flujos de trabajo complejos. [29]

Oozie funciona como un motor de workflows a modo de servicio que permite lanzar, parar, suspender, retomar y volver a ejecutar una serie de trabajos Hadoop -no solamente MapReduce- basándose en ciertos criterios, como temporales o de disponibilidad de datos. Los flujos de trabajo Oozie son grafos no cíclicos directos -también conocidos como DAGs- donde cada nodo es un trabajo o acción con control de dependencia, es decir, que una acción no puede ejecutarse a menos que la anterior haya terminado. Los flujos tienen dos tipos de nodos: [30]

- **Nodos de acción:** ejecutan una tarea de procesamiento como un trabajo MapReduce, Pig, una acción HDFS o incluso otro workflow de Oozie. Se puede extender para que acepte otros tipos de acciones.
- **Nodos de control:** definen el principio y el final de un flujo de trabajo -nodos de start, fail y end- así como mecanismos para el flujo de ejecución -nodos de decisión, división y unión-.

```
<workflow-app name='wordcount-wf' xmlns="uri:oozie:workflow:0.1">
  <start to='wordcount'/>
  <action name='wordcount'>
    <map-reduce>
      <job-tracker>${jobTracker}</job-tracker>
      <name-node>${nameNode}</name-node>
      <configuration>
        <property>
          <name>mapred.mapper.class</name>
          <value>org.myorg.WordCount.Map</value>
        </property>
        <property>
          <name>mapred.reducer.class</name>
          <value>org.myorg.WordCount.Reduce</value>
        </property>
        <property>
          <name>mapred.input.dir</name>
          <value>${inputDir}</value>
        </property>
        <property>
          <name>mapred.output.dir</name>
          <value>${outputDir}</value>
        </property>
      </configuration>
    </map-reduce>
    <ok to='end'/>
    <error to='kill'/>
  </action>
  <kill name='kill'>
    <message>Something went wrong: ${wf:errorCode('wordcount')}</message>
  </kill/>
  <end name='end' />
</workflow-app>
```

Figura 29: Ejemplo de definición de un workflow Oozie.

Herramientas Hadoop

Los flujos de datos están escritos en un lenguaje similar a XML llamado hPDL, como el de la Figura 29. En el ejemplo se define un workflow con el estado inicial start, que pasa a un nodo de acción de nombre wordcount. Al nodo de acción se le indica que es un MapReduce y cuál es la clase Java que realizará la tarea de Mapper y cual la de Reducer, así como la entrada y salida del proceso. Finalmente se le indica que si el resultado es correcto vaya al nodo de control llamado end y si se produce algún error vaya a kill, que mandará un mensaje con el código de error.

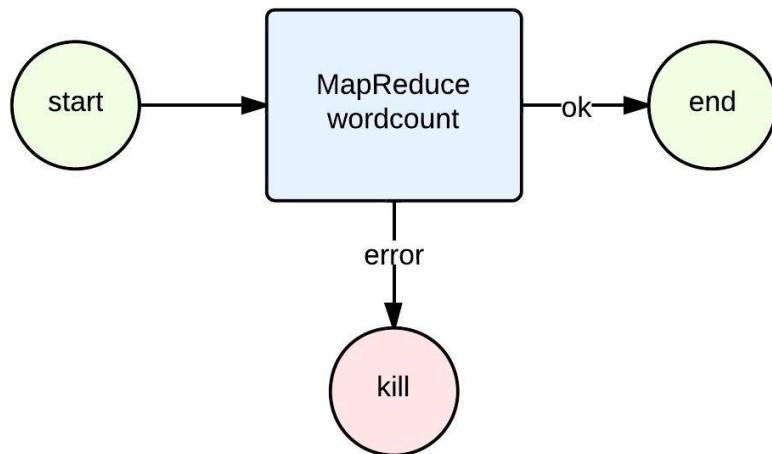


Figura 30: grafo del workflow definido en el ejemplo de la Figura 29.

5.4.3. PIG



Es una herramienta para analizar grandes volúmenes de datos mediante un lenguaje de alto nivel -PigLatin- que está diseñado para la paralelización del trabajo. Mediante un compilador se traducen las sentencias en PigLatin a trabajos MapReduce sin que el usuario tenga que pasar a programar ni tener conocimientos sobre ellos. Así pues PigLatin es un lenguaje fácil de programar ya que se trata de un lenguaje textual y convierte las paralelizaciones en dataflows o flujos de datos, conceptos más sencillos para los usuarios no expertos o sin conocimiento en el mundo de la paralelización. Además también se encarga de optimizar de manera automática los programas escritos por el usuario, respetando la coherencia de los datos, antes de traducirlos a trabajos MapReduce. Adicionalmente, también permite la creación de funciones por parte del usuario para realizar procesamientos de propósito especial y que no se incluya en las operaciones básicas de PigLatin. [31]

Obviamente Pig puede ejecutarse tanto en clústeres Hadoop -modo MapReduce- o en servidores sin una instalación Hadoop, es decir, sin ningún tipo de sistema distribuido -modo local-. Adicionalmente tiene dos modos para trabajar:

- **Modo interactivo:** se trabaja en un terminal grunt -una línea de comandos- y permite lanzar los comandos y sentencias una a una y de manera interactiva. Se lanza mediante el comando `pig` en una línea de comandos y con el argumento “-x” se le indica si se quiere lanzar en modo local o mapreduce.
- **Modo batch:** si se le pasa por argumento al comando “`pig`” un fichero de script, se ejecutará el dataflow que contenga el fichero. También acepta modo local y mapreduce.

Las sentencias PigLatin son operaciones sencillas que reciben una relación -es decir, una colección de datos- y producen otra colección -esto es cierto salvo por las operaciones de lectura y escritura sobre un sistema de ficheros-. Las relaciones se pueden guardar en variables. A continuación se listan algunas de las sentencias más comunes de PigLatin:

- Para leer datos de una fuente de información existe el operador **LOAD**.
- El operador **FILTER** permite trabajar con tuplas o filas de datos. Se usa juntamente con el operador **FOREACH** cuando se quieren recorrer las columnas de cada tupla o fila.

```
A = LOAD 'student' USING PigStorage() AS (name:chararray, age:int, gpa:float);
B = FOREACH A GENERATE name;
DUMP B;
(John)
(Mary)
(Bill)
(Joe)
```

Figura 31: ejemplo de las sentencias LOAD, FOREACH y DUMP.

- A la hora de agrupar datos de diferentes relaciones en una sola se usa **GROUP**. También se pueden hacer joins con las siguientes operaciones: **inner JOIN** y **outer JOIN**.
- Para mezclar datos o contendios de dos relaciones distintas se usa **UNION**.
- Para dividir una relación en varias, se usa **SPLIT**.
- Para almacenar datos de manera persistente se usa el operador **STORE**.
- A la hora de debugar PigLatin ofrece las siguientes sentencias:
 - **DUMP:** muestra el contenido de una variable.
 - **DESCRIBE:** muestra el esquema de una variable.

Herramientas Hadoop

- **EXPLAIN:** explica lógica o físicamente la ejecución MapReduce que ha realizado la transformación en la relación.
- **ILLUSTRATE:** muestra paso a paso la ejecución de un conjunto de sentencias.

Una cosa importante de PigLatin es que solo ejecuta las sentencias cuando se llama a un DUMP o STORE -básicamente las funciones que producen una salida de datos-. Sin una de estas sentencias lo que hará Pig será analizar y validar las sentencias del script pero no ejecutarlas. [32]

5.5. ADMINISTRACIÓN

Hadoop es un proyecto muy grande y muy ambicioso que a veces puede parecer un poco caótico y complicado de entender y administrar. Por eso mismo han salido varias aplicaciones con el objetivo de facilitar no solo la administración sino también el uso y la monitorización de los sistemas Hadoop.

5.5.1. HUE



Hue es una herramienta que proporciona a los usuarios y administradores de las distribuciones Hadoop una interfaz web para poder trabajar y administrar las distintas herramientas instaladas. De esta manera Hue ofrece una serie de editores gráficos, visualización de datos y navegadores para que los usuarios menos técnicos puedan usar Hadoop sin mayores problemas. Hue es una herramienta principalmente desarrollada por Cloudera -6.1. Cloudera-, razón por la que cuenta con algunas características de su distribución, como un editor para Impala.

Entre las herramientas con las que Hue permite trabajar tenemos las siguientes:

- **File Browser:** permite navegar entre el sistema de ficheros HDFS como si de un sistema de ficheros Unix se tratará. Facilita mucho la navegación ya que es rápido y permite realizar las operaciones más básicas: creación y eliminación de ficheros y directorios, mover datos, modificar los permisos y ver el contenido de los ficheros.

Type	Nombre	Tamaño	Usuario	Grupo	Permisos	Fecha
	.		hue	hue	drwxr-xr-x	July 29, 2013 03:14 AM
	..		hdfs	supergroup	drwxrwxrwx	July 24, 2013 04:34 AM
	.Trash		hue	hue	drwx-----	October 18, 2013 05:00 PM
	.staging		hue	hue	drwx-----	October 18, 2013 02:06 AM
	oozie		hue	hue	drwxrwxrwt	July 29, 2013 03:14 AM
	oozie-oozi		hue	hue	drwxr-xr-x	July 29, 2013 03:15 AM

Ilustración 7: Navegador para HDFS de Hue.

- **Metastore manager:** es una herramienta que ayuda en la administración de las bases de datos Hive. Permite crear o eliminar bases de datos o tablas, modificar el esquema o visualizar los datos que contienen.

Nombre de tabla
comments
comments1024
comments256
log_id
patterns
tweets

Ilustración 8: Metastore manager de Hue.

- **HBase browser:** como el metastore manager, es una herramienta para visualizar las estructuras de HBase.

Herramientas Hadoop

- **Job browser:** ofrece un listado de los trabajos MapReduce que se han ejecutado o se están ejecutando en el clúster. Indica su estado (completado, porcentajes o errores) así como la distribución del trabajo entre los nodos y otras estadísticas de tiempo.

The screenshot shows the 'Job Browser' interface in Hue. At the top, there are filters for 'Estado del trabajo' (Todos los estados), 'Mostrar trabajos retirados', 'Nombre de usuario' (hue), and a 'Filtro de texto' input field. Below the filters is a table with columns: ID, Nombre, Estado, Usuario, Maps, Reduces, Cola, Prioridad, Duración, and Fecha. The table lists five completed jobs (succeeded) with their respective details.

ID	Nombre	Estado	Usuario	Maps	Reduces	Cola	Prioridad	Duración	Fecha
1382010592519_0194	SELECT query, count (*) FROM comme..query(Stage-1)	succeeded	hue	100.0	100.0	default	N/D	3m 2s	10/21/13 00:14:37
1382010592519_0009	select * from logs_rtd where ..timestamp)(Stage-1)	killed	hue	100.0	100.0	default	N/D	27m 12s	10/18/13 01:58:48
1382010592519_0008	select CTIME, count(*) from logs_r.. CTIME(Stage-1)	succeeded	hue	100.0	100.0	default	N/D	8m 51s	10/18/13 01:58:04
1382010592519_0006	select count(*) from tweets(Stage-1)	succeeded	hue	100.0	100.0	default	N/D	4m: 17s	10/18/13 01:34:24
1382010592519_0001	select * from logs_rtd where ..timestamp)(Stage-1)	succeeded	hue	100.0	100.0	default	N/D	7m 7s	10/17/13 04:51:57

Ilustración 9: Job Browser de Hue.

- **User admin:** para administrar los usuarios de Hue y grupos a los que pertenecen.
- **Beeswax Hive UI:** ofrece un sencillo editor gráfico -con alguna característica como autocompleteo- para realizar consultas sobre Hive. No sólo da el editor sino también muestra los resultados en la misma web -con la opción de descargarlos en distintos formatos- y ver el estado del trabajo en tiempo real. También permite ver un historial e las consultas realizadas - así como de los resultados obtenidos- y guardar las consultas que el usuario crea convenientes.

The screenshot shows the Beeswax Hive UI. At the top, there are tabs for 'Editor de consultas', 'Mis consultas', 'Consultas guardadas', 'Historial', and 'Ajustes'. Below the tabs, it says 'Resultados de la consulta: Consulta sin guardar'. On the left, there's a sidebar with 'DESCARGAS' and options to 'Descargar como CSV' or 'Descargar como XLS'. On the right, there's a table with columns: Resultados, Consulta, Registro, and Columnas. The table shows the results of a query with four rows, each containing a value for 'query' and '_c1'.

Resultados	Consulta	Registro	Columnas
▲	query	▲ _c1	
0	#ipod	9887232	
1	#apple	40277416	
2	#iphone	980720	
3	#ipad	1961440	

Ilustración 10: Resultados de una consulta Hive con Hue.

- **Impala:** es exactamente el mismo editor que para Hive pero orientado a Impala.
- **ZooKeeper UI:** permite navegar entre los znodes y su jerarquía y ver la información y el estado del sistema.
- **Job designer:** para crear trabajos MapReduce de manera gráfica a través de un cuestionario.
- **Oozie editor:** permite crear workflows de Oozie de manera gráfica, ejecutarlos y ver la información sobre el estado de ejecución.
- **Pig editor:** permite crear dataflows y scripts de Pig y seguir la evolución de su ejecución.

6. DISTRIBUCIONES HADOOP

En este apartado se analizan algunas de las distribuciones Hadoop más populares, concretamente las estudiadas por el autor de la memoria. Es el último paso de la fase de estudio teórico y el objetivo es realizar una comparación teórica entre todas las soluciones estudiadas. Al final de este apartado se muestra la tabla comparativa explicada que se ha realizado entre los dos autores del proyecto.

6.1. CLOUDERA

Cloudera es una empresa americana fundada el año 2008 por los creadores y parte del equipo original del proyecto Hadoop. Se dedica únicamente a ofrecer soluciones Hadoop para Big Data y es una de las compañías líderes y punteras en este campo. Aparte de las soluciones Big Data, Cloudera también se dedica a ofrecer soporte para sus productos y cuentan con un sistema de entrenamiento y certificaciones profesionales llamado Cloudera University.

Es una de las compañías clave en el desarrollo de Hadoop, desarrollando nuevas herramientas open-source y colaborando en el desarrollo continuo de nuevas versiones del ecosistema; ya sea activamente o apostando por las últimas versiones de esta.

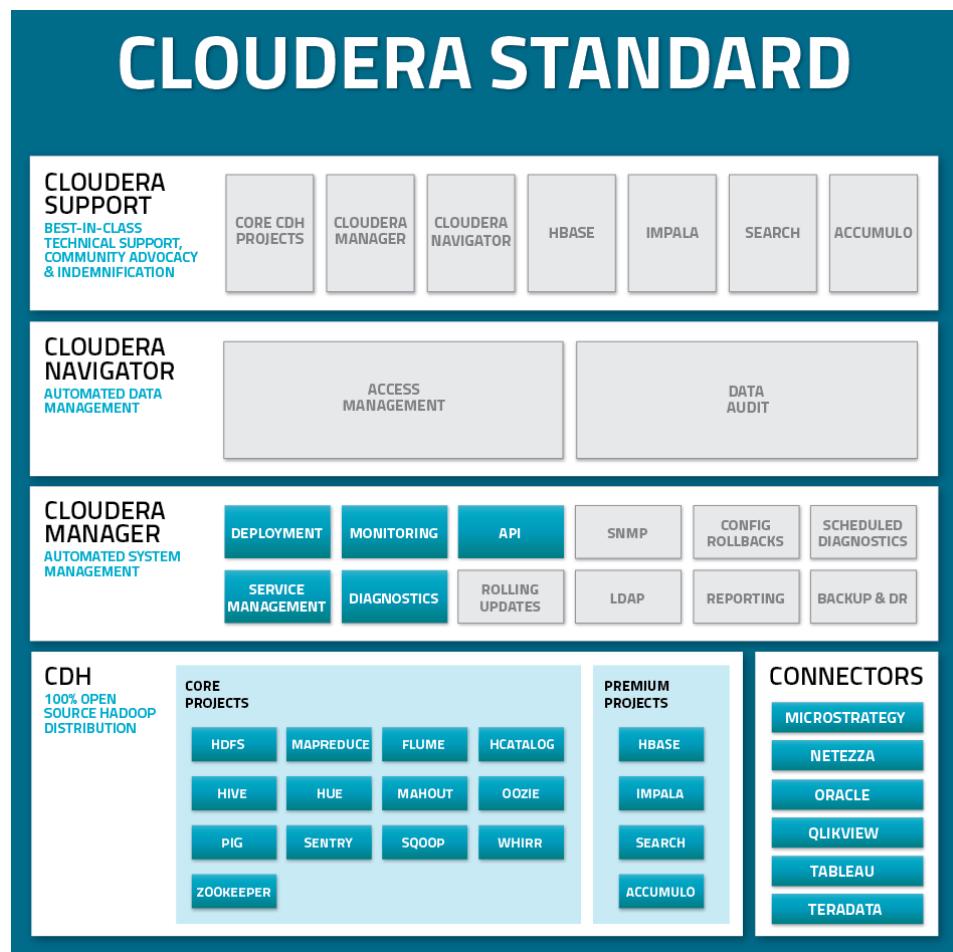


Ilustración 11: Esquema de la arquitectura de CDH. [33] Los elementos coloreados son los que se ofrecen de serie con la versión gratuita mientras que los grises son elementos de pago.

Distribuciones Hadoop

6.1.1. CLOUDERA DISTRIBUTION INCLUDING APACHE HADOOP

La base sobre la que está construida su distribución es Cloudera Distribution including Apache Hadoop (CDH), que actualmente se encuentra en su versión 4 -lanzada en febrero de 2013-. Es una distribución Hadoop 2.0 completamente open-source que ofrece el núcleo de la segunda versión de Hadoop -HDFS, YARN (MRv2)- así como compatibilidad con MapReduce (MRv1). También incluye las herramientas que forman parte del ecosistema como Flume, Sqoop, HBase, Hive, Pig, Oozie o Hue.

Con el lanzamiento de CDH4, Cloudera lanzó oficialmente las primeras versiones estables de dos herramientas que añaden mucho valor a su distribución: Impala y Search.

6.1.1.1. CLOUDERA IMPALA

Impala nace como respuesta a una necesidad creada por un vacío en la concepción de Hadoop y las diferentes herramientas que lo engloban. A la hora de trabajar con datos y tablas relacionales, Hive es la herramienta adecuada pero tiene un hándicap en ciertas situaciones: trabaja sobre MapReduce. Esto hace que al lanzar consultas que deberían ejecutarse de una manera rápida, Hive lance un proceso Java que puede tardar unos segundos extra además de ser costosa. Por lo tanto no es una herramienta ágil para hacer análisis de datos sobre bases de datos de un tamaño reducido y cuando se busca tener una respuesta a las consultas en tiempo real.

Cloudera Impala es una herramienta que trabaja con el lenguaje de consultas SQL y que permite realizar consultas en tiempo real, ya que trabaja de forma independiente a MapReduce, eludiendo la sobrecarga que supone. También se beneficia de los metadatos de Hive, de manera que es totalmente compatible con sus tablas y puede hacer consultas en bases de datos distribuidas.

A pesar de esto, Impala no es un sustituto de Hive, que sigue siendo más eficiente a la hora de realizar trabajos de transformaciones -ETL- sobre grandes volúmenes de datos. Además, Impala aún no cuenta con algunas características como funciones definidas por el usuario o la serialización/deserialización a través de frameworks de distintos formatos de datos. [34] De hecho Impala y Hive son totalmente complementarios ya que en cierta medida -y gracias a Impala- CDH ofrece una arquitectura muy cercana a las estudiadas en el punto 3.6.4 *Arquitectura mixta*, ya que permite trabajar con grandes volúmenes de datos gracias a Flume, Hadoop o Mahout y almacenar estos resultados en una tabla Hive para ser consultados con Impala, obteniendo un tiempo de respuesta más reducido.

La arquitectura de Impala y el modo de funcionamiento es muy parecido a la de cualquier otra herramienta de Hadoop [35]. Se cuenta con un servicio, *impalad*, que se ejecuta en todos los nodos del clúster y cuya función es la de recibir y servir las diferentes consultas de los usuarios. Una vez uno de los nodos recibe una consulta, se convierte en el nodo coordinador y distribuye la carga de trabajo sobre los otros nodos impalad. Finalmente recibe los resultados parciales de los demás nodos y construye la respuesta final de la consulta. Se puede realizar balanceo de carga entre los diferentes nodos indicándoselo mediante la interfaz.

Para monitorizar el estado de los nodos hay un servicio adicional llamado *statestored*, que recibe periódicamente mensajes de los nodos impalad para saber cuáles de ellos están disponibles para realizar nuevos trabajos. De esta manera se evita que los nodos impalad coordinadores intenten enviar trabajo a los nodos caídos u ocupados. Este servicio no es obligatorio para el funcionamiento normal de Impala, en su ausencia el clúster trabaja con normalidad aunque no tenga la robustez y la seguridad que ofrece el *statestored*.

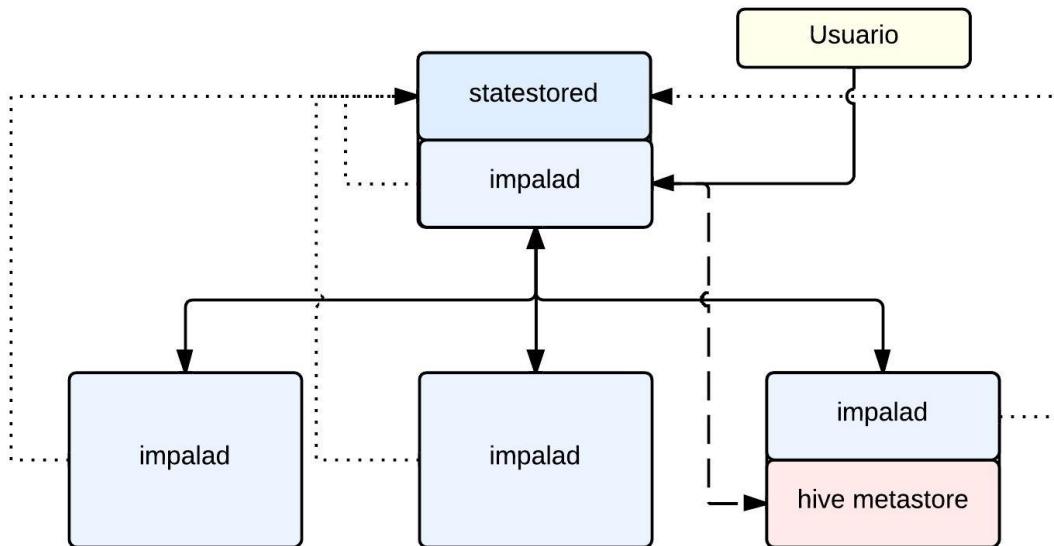


Figura 32: arquitectura Impala con cuatro nodos impalad. El nodo que recibe la consulta se convierte en el coordinador, consulta el metastore de Hive para conocer la estructura de los datos y sincroniza el trabajo de los demás impalad. Paralelamente todos los nodos envían información de su estado al statestored.

Como se ha dicho, Impala funciona con un lenguaje SQL -más concretamente con un subconjunto de Hive QL-, por lo que es familiar para los usuarios de bases de datos relacionales. Para trabajar con Impala hay varias opciones: desde consola de comandos -parecida a la de Hive, por ejemplo-, desde la interfaz web Hue o mediante conectores JDBC u ODBC para conectar distintos tipos de herramientas de visualización.

6.1.1.2. CLOUDERA SEARCH

Cloudera Search es una herramienta que permite al usuario realizar búsquedas interactivas de texto en los datos almacenados en HDFS. [36] Funciona sobre la utilidad Apache Solr y permite a los usuarios no técnicos realizar búsquedas y explotación de datos, aumentando la productividad. Al contrario de Hive, Impala o Pig no están ligados a datos estructurados o semi.estructurados y tampoco requiere de implementación de código, como en el caso de MapReduce. De esta manera se puede navegar por los datos sin tener que realizar trabajos MapReduce y sin el tiempo añadido que estos suponen. [37]

Search se aprovecha de un buen número de herramientas ya existentes en Hadoop para realizar estas consultas: HDFS, MapReduce, Flume, Hue, ZooKeeper, HBase, Cloudera Manager, Oozie, Impala, Hive, Avro, Sqoop y Mahout.

6.1.2. CLOUDERA MANAGER

Es la herramienta que ofrece Cloudera para administrar el clúster y la distribución Hadoop. A través de una interfaz web permite al usuario instalar el clúster desde cero -ya sea mediante los repositorios oficiales de Cloudera o unos propios-, añadir o quitar nodos y servicios, realizar la configuración de cualquier aspecto de la distribución Hadoop y monitorizar los distintos servicios que se ejecutan -a través de un sistema de logs o de gráficas de rendimiento de los recursos del sistema-. También se encarga de realizar comprobaciones periódicas del estado del clúster, ya sea de los nodos conectados o de los servicios activos.

6.1.3. CLOUDERA NAVIGATOR

Navigator es la herramienta ofrecida por Cloudera para realizar tareas de administración, monitorización y análisis sobre los datos y su evolución a lo largo del tiempo; dado que a medida que pasa la complejidad de los datos crece exponencialmente. [38] Entre sus características principales se encuentran:

- Realizar auditorías de datos y controlar el acceso a estos. Controla los permisos de acceso, lectura y escritura de los ficheros almacenados en HDFS, las entradas de HBase y los metadatos de Hive.
- Explorar el sistema de datos, buscando los datos disponibles y como están estructurados, para encontrar la forma idónea de trabajar con ellos.
- Trazar la evolución de los conjuntos de datos desde su origen para comprobar la validez de estos.

6.1.4. PRODUCTOS

Cloudera empaqueta su distribución CDH en una serie de productos que se adaptan a las necesidades de cada uno:

- **Cloudera Standard:** es una distribución gratuita que incluye CDH con Cloudera Manager limitado a opciones básicas de instalación, administración y monitorización.
- **Cloudera Enterprise:** incluye el resto de características de la distribución CDH, como una versión completa del Manager, Navigator y el soporte de sus expertos.
- **Subscripciones:** estas subscripciones añaden soporte para las distintas tecnologías incluidas en CDH.
 - **RTD** (real-time delivery) para HBase.
 - **RTQ** (real-time query) para Impala.
 - **RTS** (real-time search) para Search.
 - **BDR** (back-up and disaster recovery) para realizar tareas de recuperación y seguridad de datos.
 - **Navigator Subscription** para el Navigator.

También ofrecen una versión de Cloudera Enterprise llamada Cloudera Quickstart que incluye también plazas para los cursos -de administración, desarrollador y analista- ofrecidos en Cloudera University.

6.2. DATASTAX

DataStax es una empresa fundada el 2010 especializada en ofrecer y distribuir soluciones de software basadas en la base de datos NoSQL Cassandra, de la cual son unos de los principales desarrolladores. De todas las soluciones Hadoop estudiadas, la ofertada por DataStax es la más diferente al introducir como principal motor de almacenamiento Cassandra, como se verá más adelante. El apoyo a Cassandra está motivado por algunos de los principales fallos de la primera versión de Hadoop como el punto de fallo único en el NameNode. También ofrecen servicios de soporte y de entrenamiento para los desarrolladores de sus plataformas así como de consultoría para las empresas que quieran introducirse en el mundo de Big Data.

6.2.1. DATASTAX ENTERPRISE

DataStax Enterprise es una solución Big Data centrada en la base de datos NoSQL Cassandra, la cual sirve como principal fuente de almacenamiento, a diferencia de las demás soluciones Hadoop que usaban principalmente HDFS. A pesar de no trabajar con el sistema de ficheros nativo de Hadoop la solución de Datastax es totalmente compatible con las demás herramientas del ecosistema Hadoop, aunque requieran del propio HDFS. Es decir que DataStax Enterprise es totalmente compatible con Hive, Pig, Oozie y otras utilidades.

6.2.1.1. CASSANDRA FILE SYSTEM (CFS)

Cassandra File System (CFS) es un sistema de ficheros desarrollado para ser compatible con HDFS (en su primera versión) y para incluso sustituirlo. Está pensado para reemplazar los servicios de NameNode, Secondary NameNode y DataNode por utilidades que se apoyan en Cassandra. Los principales objetivos de CFS son eliminar los problemas que se arrastran debido a la arquitectura de HDFS -principalmente los puntos de fallo único como el NameNode- y para que las utilidades como Hive o Mapreduce pudieran ser compatibles con DataStax Enterprise.

Gracias a la arquitectura P2P de Cassandra desaparecen los puntos de fallo únicos del sistema, ya que todos los nodos ejercen el mismo rol. A cambio, se añade un modelo de datos -almacenado en Cassandra- para los metadatos que contendrán la información que antes usaba el NameNode. [39]

El sistema de ficheros está modelado en un espacio de claves con dos familias de columnas:

- **sblocks**: contiene los metadatos que sustituyen las funciones de los DataNodes. Es decir, es el conjunto de bloques de un fichero y su contenido.
- **inode**: contiene los metadatos de un NameNode: nombre del fichero, usuario, grupo, permisos, tipo y la lista de id de los bloques del fichero. Para los id de los bloques usa un TimeUUID, ordenando los bloques de forma secuencial.

De esta manera se evita tener una arquitectura más compleja maestro-esclavo y se simplifica la configuración de la red.

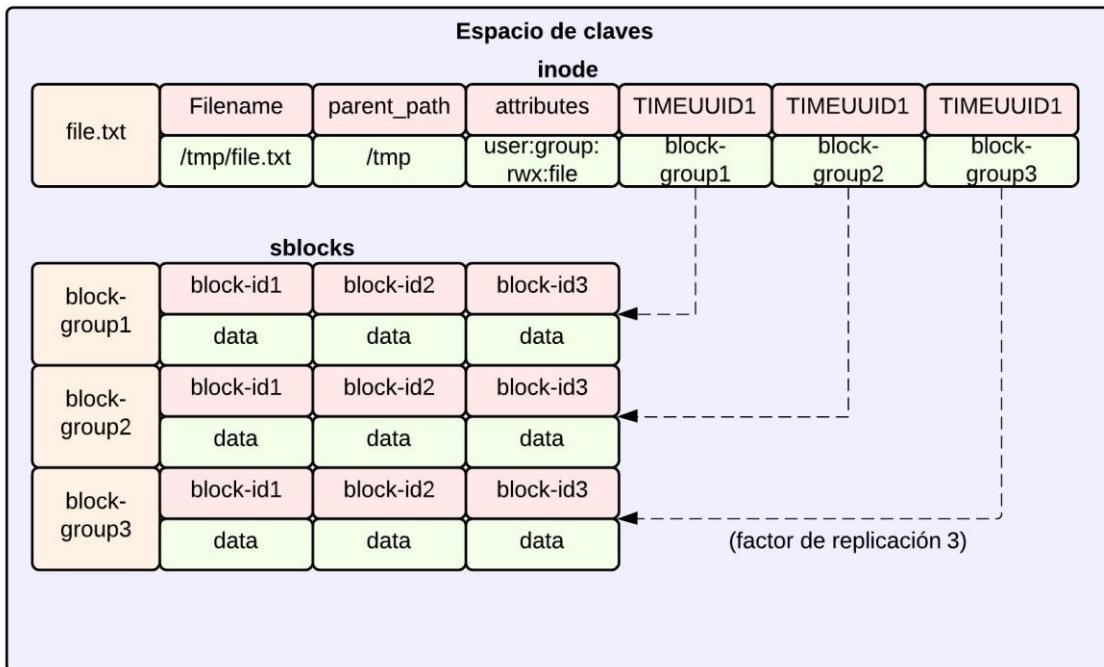


Figura 33: espacio de claves correspondiente a DFS para un fichero `file.txt` (dividido en 3 bloques) en un clúster con factor de replicación 3.

Cuando se añade un fichero en CFS, se escribe en inode los metadatos estáticos del fichero (nombre, permisos, etc.) y, dependiendo de la configuración HDFS como el tamaño de bloque, se divide el fichero en bloques y se añaden en sblocks -una fila por replicación-. Los bloques son comprimidos mediante un sistema de compresión de Google y una vez son añadidos en sblocks, se escriben los identificadores en la fila correspondiente del inode. [40]

Uno de los problemas de CFS es que a diferencia de con HDFS, no se puede tener un factor de replicación distinto para cada fichero. Una de las soluciones posibles a este problema es tener más de un espacio de claves, cada uno con su propio factor de replicación. Esto significa tener varios sistemas de ficheros.

6.2.1.2. DATASTAX OPSCENTER

OpsCenter es la herramienta de DataStax para administrar y monitorizar la distribución y la infraestructura. A través de una interfaz web, los usuarios pueden realizar tareas de administración como la instalación y configuración de nuevos servicios, realizar trabajos a través de los nodos y crear dashboards para visualizar el rendimiento de los diferentes nodos del clúster y poder realizar análisis de funcionamiento y de carga de trabajo.

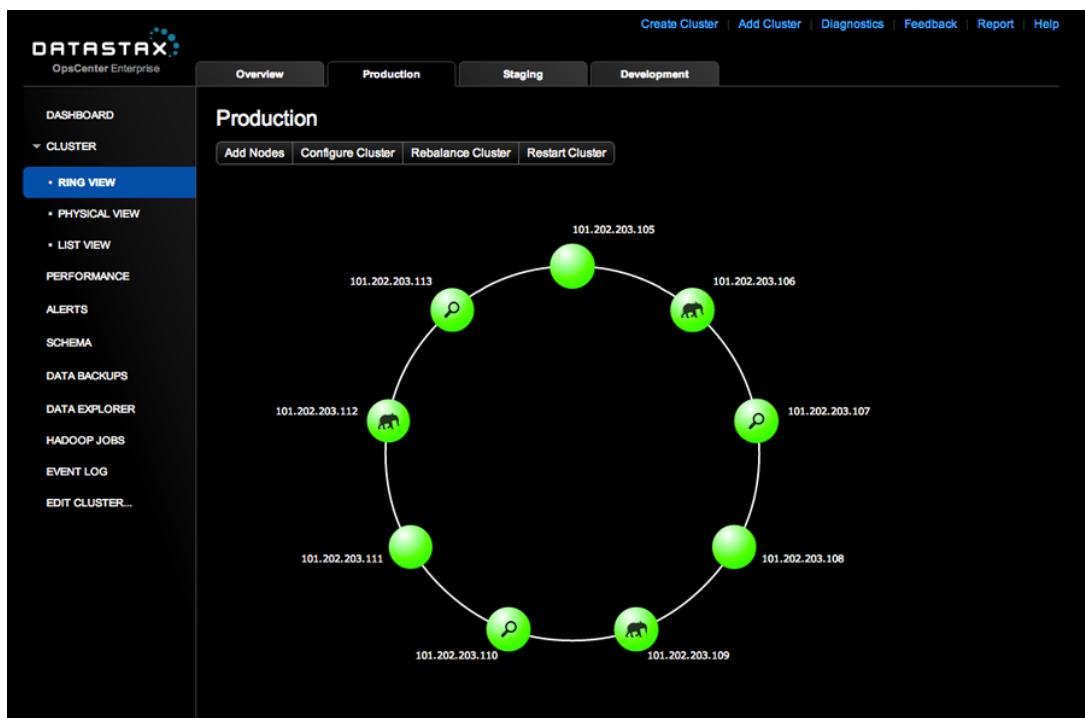


Ilustración 12: pantalla de OpsCenter. Muestra el esquema de la disposición del clúster.

6.2.1.3. DATASTAX DEVCENTER

DevCenter es una herramienta de desarrollo pensada para realizar programas y scripts en CQL para trabajar sobre Cassandra y DataStax Enterprise en general. Entre sus características están las de navegar fácilmente entre los distintos esquemas de clústers, espacios de claves y bases de datos; realizar conexiones sencillas entre las bases de datos y la herramienta de desarrollo o funciones de editor de programación como resaltado de sintaxis, auto completado o validación de la sintaxis.

6.2.1.4. DATASTAX COMMUNITY EDITION

DataStax también cuenta con una versión gratuita de su distribución. Las principales diferencias entre ambas versiones es que la gratuita no ofrece compatibilidad por defecto con Hadoop (es de configuración manual), trae una versión limitada de OpsCenter y todas las actualizaciones y versiones que recibe son de la comunidad, nunca desde DataStax. Además tampoco se puede actualizar la versión de la comunidad a Enterprise. [41]

Distribuciones Hadoop

The screenshot shows the Apache DevCenter interface with several windows open:

- Connections:** Shows a connection to "cassandra-1.2.10 [Available]".
- worksheet.cql:** A CQL script window containing code to create tables and insert data into Cassandra. The code includes creating a keyspace "cassandra_community", creating tables "cassandra_users" and "cassandra_mvps", and inserting various user records with their details and site preferences.
- insert.cql:** A CQL script window showing a single INSERT statement for a user record.
- setup.cql:** A CQL script window showing a single UPDATE statement for a user record.
- CQL Scripts:** A sidebar listing the CQL script files: "create_schema.cql", "insert.cql", "schema_upgrade_v1.cql", "setup.cql", and "worksheet.cql".
- Schema:** A tree view of the Cassandra schema for "cassandra-1.2.10" showing the "cassandra_community" keyspace, its tables ("cassandra_users", "cassandra_mvps"), columns, and indexes.
- Outline:** A tree view of the current session's activity, showing the history of queries run, including CREATE KEYSPACE, INSERT INTO, and UPDATE statements.

Ilustración 13: Pantalla de desarrollo de DevCenter.

6.3. PIVOTAL

Pivotal es una compañía fundada el Abril de 2013 pero que cuenta con muchos años de experiencia detrás ya que fue fundada por EMC, desde su división de Big Data, Greenplum; y en colaboración con VMWare. Está centrada en soluciones Big Data -con Pivotal HD- y en soluciones cloud -con Pivotal CF, que en el momento de escribir la memoria aún no ha sido lanzado-. La mayoría del equipo de Pivotal -unos 200 al fundar la compañía- eran prácticamente todos parte de Greenplum, hoy en día desaparecida y sustituida por Pivotal dentro de EMC.

6.3.1. PIVOTAL HD ENTERPRISE

Pivotal HD Enterprise es la solución propuesta por Pivotal construida sobre Hadoop 2.0 y con un buen número de añadidos para complementar las ya existentes. Como se puede ver en la *Ilustración 14* cuenta con HDFS como almacenamiento principal y con MRv2 más las principales herramientas de análisis de datos: Pig, Hive, Mahout... [42]

Los principales añadidos de Pivotal HD Enterprise son la herramienta de administración y monitorización Command Center, el paquete HAWQ, que incluye una herramienta de consulta de datos MPP y una librería de funciones para el análisis de datos; y el paquete GemFire. Adicionalmente se incluyen otras herramientas como el Data Loader, para cargar datos a HDFS; o Spring, para realizar workflows.

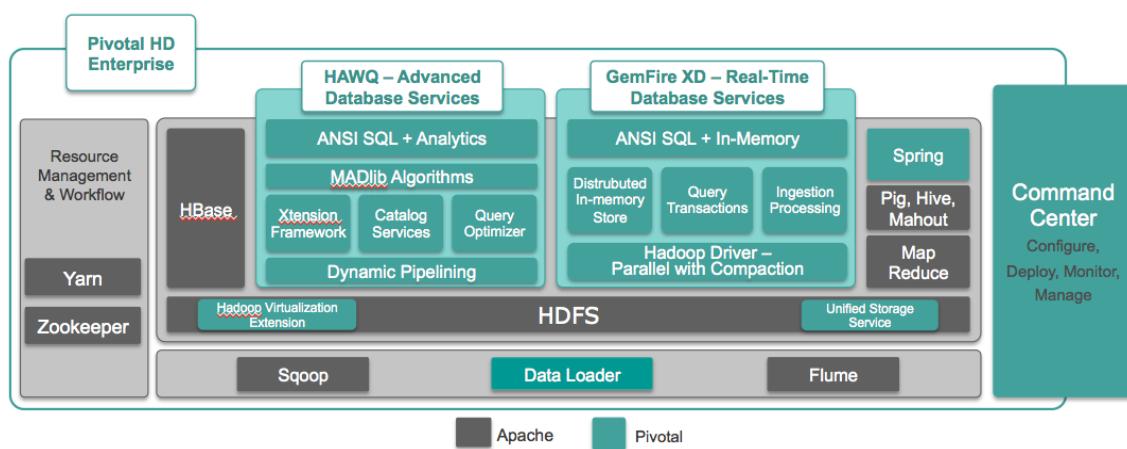


Ilustración 14: Arquitectura de la solución Pivotal HD Enterprise.

- **Data Loader:** es otra herramienta para la recolección de datos, que permite paralelizar la escritura de los datos capturados a HDFS.
- **Unified Storage Service:** permite visualizar como si fuera un solo espacio de nombres diversos sistemas de ficheros, permitiendo realizar las operaciones típicas -crear, copiar, mover o eliminar- sin la necesidad de crear copias intermedias.
- **Hadoop Virtualization Extension:** permiten a Hadoop ser consciente de la topología de virtualización y le permite escalar los nodos virtuales dinámicamente.
- **Spring:** es una herramienta que proporciona APIs simplificadas para usar HDFS, MapReduce, Hive y otras utilidades Hadoop. También permite la creación de workflows mediante el uso de estas APIs.
- **Command Center:** es una interfaz web que permite instalar, configurar, monitorizar y administrar todos los componentes Hadoop.

6.3.2. HAWQ - ADVANCED DATABASE SERVICES



Advanced Database Services (ADS) es un paquete que engloba diversas herramientas y utilidades, destacando por encima de todas HAWQ. Con HAWQ se añade una herramienta MPP para la consulta de datos en HDFS formando una arquitectura mixta entre MapReduce y MPP. Al igual que Impala de Cloudera no usa MapReduce para realizar las consultas y en cambio usa un lenguaje SQL real, con lo que facilita la conexión de las bases de datos en Hadoop con las herramientas de visualización y transformación de datos ya existentes.

HAWQ está preparado para funcionar con clústeres de cientos de nodos y con bases de datos del orden de varios petabytes. Ofrece un motor de consultas de datos que engloba la división de consultas complejas en varios pasos y la paralelización de estas a través de los nodos. Además gracias a Pivotal Xtension Framework HAWQ es capaz de acceder a los datos almacenados en diversas herramientas Hadoop como HDFS, HBase o Hive; con un rendimiento parecido al de una consulta en HAWQ.

La otra característica clave que ofrece el paquete es una librería de algoritmos para el análisis y proceso de datos llamada MADlib, e incluye funciones de estadística, matemáticas y de aprendizaje. Estas funciones están implementadas sobre HAWQ y se pueden ejecutar como si fueran un comando SQL más. MADlib es una librería de código libre y está pensada para realizar operaciones en bases de datos distribuidas y escalables. [43]

6.3.3. GEMFIRE XD - REAL-TIME DATABASE SERVICES

Mientras HAWQ trabaja con bases de datos en disco, GemFire XD proporciona a los desarrolladores la posibilidad de hacer aplicaciones que aprovechen la velocidad de la memoria principal de los nodos. Elimina la barrera de la limitación de este tipo de memoria mediante un proceso que gestiona y coordina los nodos del clúster para trabajar con gigabytes de información -lo que actualmente se suele tener en las memorias de los servidores-. Como HAWQ, es compatible con un lenguaje SQL y permite trabajar también con HDFS e información almacenada en disco.

El objetivo de GemFire XD es la de ofrecer aplicaciones en tiempo real gracias a un almacenamiento con una arquitectura que solamente comparte un recurso del clúster: la red. [44] De esta manera la capa de aplicación no tiene que preocuparse de cómo se están administrando los datos, simplemente se conecta y trabaja con ellos a través de una interfaz HashMap.

La arquitectura de GemFire XD da al clúster donde se está ejecutando una infraestructura parecida a la que se obtiene con las soluciones cloud estudiadas en el apartado 3.7.3. Cloud. Añadir un nodo es tan fácil como conectarlo al clúster e inmediatamente se obtiene una escalabilidad lineal sin influir en el rendimiento de los demás. Además es capaz de estar ejecutándose a la vez que se hace una actualización, tanto de hardware como de software, en algunos de los nodos.

6.3.4. PIVOTAL ANALYTICS WORKBENCH (AWB)

Es un servicio ofrecido por Pivotal que consiste en un clúster de 1.000 nodos con 24 petabytes de almacenamiento. Es usado principalmente para realizar pruebas y experimentar sobre Pivotal HD y poder hacer demostraciones de grandes clústeres a las empresas. [45]

6.3.5. PIVOTAL DATA COMPUTING APPLIANCES (DCA)

Pivotal también ofrece a sus clientes una arquitectura *appliance* con Pivotal HD. [46] La arquitectura DCA está compuesta por módulos clasificados por roles:

- **Módulos para bases de datos:**
 - **Pivotal Database Standard:** están optimizados para el almacenamiento de datos y ser altamente escalables. Proporcionan una buena arquitectura para almacenar, hacer pequeñas consultas y operaciones de red para la comunicación entre los demás módulos.
 - **Pivotal Database Compute:** como su nombre indica están optimizados para realizar tareas de computación sobre un cierto volumen de datos. Son los adecuados para realizar minería de datos, aprendizaje y consultas más pesadas que requieran un rendimiento elevado.
- **Módulos para Pivotal HD:**
 - **Pivotal HD Server:** combinan recursos para MapReduce y un almacenamiento tipo SATA de gran capacidad para HDFS.
 - **Pivotal HD Compute:** ofrecen más recursos para MapReduce, quitando la capa de almacenamiento para HDFS. Al poder conectar un dispositivo externo a modo de almacenamiento permite conectarse con otros sistemas.
- **Módulos Pivotal Data Integration Accelerator (DIA):** están pensadas para añadir aplicaciones de administración y análisis de datos de terceros. Ya sea para realizar tareas de visualización o de recolección de información.

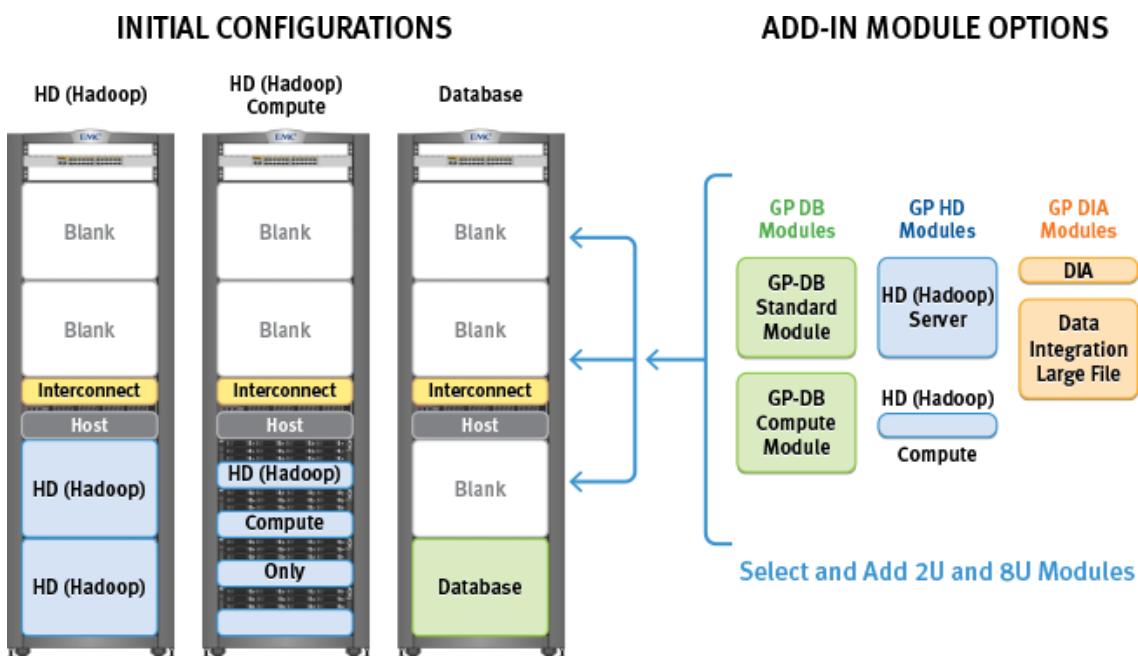


Ilustración 15: Esquema de la distribución de módulos de la appliance de Pivotal.

Gracias a estos módulos, el usuario puede configurar y escalar su clúster de manera sencilla, optimizando los racks para las tareas específicas que deban realizar y adaptándolo al presupuesto que se tenga. La Ilustración 15 muestra un esquema de una appliance Pivotal DCA.

6.3.6. VERSIÓN DE COMUNIDAD

Pivotal HD cuenta con una versión gratuita -Community- pero está limitada sólo a una distribución Hadoop 2.0 y a algunas herramientas y utilidades aportadas por Pivotal como el Data Loader y Command Center. Además de una limitación para los clústeres de 50 nodos. Esta versión sin embargo no incluye HAWQ, GemFire XD ni el paquete de funciones para análisis de datos. [47]

6.4. COMPARATIVA

En este apartado se muestra la comparativa realizada entre las distintas distribuciones estudiadas con el objetivo de escoger cuál es la que más encaja en las necesidades del proyecto. Los aspectos valorados han sido puntuados del 1 (peor) a 3 (mejor) según los criterios del proyecto:

- **Presencia en el mercado:** la aceptación y el uso que tiene la distribución en el mercado entre las empresas. Es un factor difícil de valorar pero importante. Significado de las valoraciones:
 - **1:** es una solución con poca presencia y que es recién llegada a la industria Big Data.
 - **2:** es una distribución usada por empresas importantes pero que no cuenta con ninguna gigante que la distribuya con su solución.
 - **3:** es usada por empresas importantes en el sector que distribuyen soluciones con la distribución como base o núcleo.
- **Trabajado en Everis:** un aspecto importante es el grado de conocimiento que tienen en Everis de las tecnologías que incluye cada distribución. Uno de los objetivos del proyecto es aportar conocimiento a la empresa por lo que cuanto más desconocida sea una distribución, mejor. Significado de las valoraciones:
 - **1:** es una de las herramientas usuales en los proyectos de Everis.
 - **2:** se ha trabajado en algún proyecto.
 - **3:** se desconoce profundamente.
- **Versión gratuita:** debido a que no se pueden conseguir licencias para el proyecto, un punto importante es cuál es el coste de cada distribución y que incluye la versión gratuita de cada una. Significado de las valoraciones:
 - **1:** incluyen una distribución Hadoop y poca cosa más.
 - **2:** incluyen gran parte de las herramientas pero con limitaciones.
 - **3:** la versión gratuita incluye gran parte de las herramientas que se quieren estudiar.
- **Productividad:** hace referencia a las facilidades que ofrece una distribución a la hora de trabajar con ella. Significado de las valoraciones:
 - **1:** no otorga ninguna facilidad para acelerar el desarrollo de aplicaciones para analizar los datos almacenados más allá de las distribuidas con el núcleo de Hadoop.
 - **2:** facilita el desarrollo de aplicaciones distribuidas gracias a herramientas que aíslan al usuario de la capa más técnica del sistema, como configuraciones o trabajar a través de línea de comandos.
 - **3:** facilita en gran medida el desarrollo de aplicaciones distribuidas, sin necesidad de conocer el modelo de programación MapReduce.
- **Rendimiento:** evalúa las adiciones de cada distribución a la base Hadoop para mejorar su rendimiento de cada a realizar procesos con los datos almacenados. Significado de las valoraciones:
 - **1:** no ofrece ningún cambio ni herramienta adicional para aumentar el rendimiento del sistema.
 - **2:** añade algunas mejoras para aumentar el rendimiento de las herramientas ya existentes de Hadoop.
 - **3:** añade algunas mejoras para aumentar el rendimiento del sistema. Añade además nuevas herramientas para mejorar el rendimiento de algunos tipos concretos de tareas.
- **Tolerancia a fallos:** puntúa los extras de cada distribución para solucionar los problemas de las soluciones Hadoop con algunos puntos poco tolerables a fallos. Significado de las valoraciones:
 - **1:** no añade nuevas características que aumenten la tolerancia a fallos respecto a las que ya existen en el proyecto Hadoop base.

Distribuciones Hadoop

- **2:** añade mejoras a Hadoop para hacerlo más tolerante a fallos y eliminar SPOF, pero sólo a HDFS o MapReduce, no contempla otras opciones (como por ejemplo HBase).
- **3:** añade nuevas funcionalidades a Hadoop para aumentar la tolerancia a fallos y/o elimina puntos de fallos únicos (SPOF).

	Cloudera	Pivotal HD	DataStax	IBM InfoSphere	MapR	Hortonworks
Presencia en el mercado	3	2	1	2	3	3
Trabajado en Everis	3	3	3	3	3	3
Versión gratuita	3	1	2	1	1	2
Productividad	2	2	2	3	1	1
Rendimiento	3	3	1	2	2	2
Tolerancia a fallos	2	2	3	2	3	2
Total	16	13	12	13	13	13

Tabla 18: Comparativa entre soluciones Hadoop.

En el punto de presencia en el mercado, tanto Cloudera como Hortonworks son distribuciones muy usadas ya que la primera, aparte de ser la distribución puntera en desarrollo Hadoop, también es la escogida por Oracle para su solución Big Data. La segunda en cambio es la escogida por Microsoft y Teradata. Pivotal tiene detrás a una gran multinacional como EMC y aunque es una recién llegada, su solución no deja de ser la que ofrecía EMC antes -con el nombre de Greenplum-.

Ninguna de las distribuciones ha sido trabajada en la empresa aparte de Cloudera. Aunque desde Everis se consideró oportuno darle la misma puntuación que las demás porque prácticamente solo se habían usado las herramientas base de Hadoop.

Las versiones gratuitas de las distribuciones de Pivotal HD, IBM y MapR no llevan prácticamente nada que sea añadido del núcleo de Hadoop, mientras que Hortonworks y DataStax añaden algunas herramientas, aunque con limitaciones. La versión de Cloudera en este sentido es la mejor ya que salvo suscripciones de soporte y una limitación de clúster a 50 nodos -que no afecta al proyecto ya que es bastante elevada-.

Por los apartados que quedan, IBM está centrada sobre todo a ofrecer una gran productividad al usuario a través de herramientas como BigSheets -consultas y trabajos mediante hojas de cálculo-, BigIndex -búsqueda de texto a través de índices- o Jaql -un lenguaje de consulta-; además de las herramientas de monitorización, administración y programación de tareas típicas que las demás distribuciones también añaden. Pivotal y Cloudera por su lado se centran en mejorar el rendimiento de la solución añadiendo herramientas como Impala y Search, en el caso de Cloudera; y HAWQ y GemFire XD en el de Pivotal.

DataStax se centra en arreglar la tolerancia a fallos del sistema de almacenamiento usando Cassandra -que elimina el punto de fallo único mediante un sistema P2P-. Con el mismo objetivo, MapR sustituye HDFS por MapR-FS, un sistema de ficheros con una filosofía muy similar a la de Cassandra, y añadiendo alta disponibilidad en las tareas MapReduce.

Finalmente, la elección de la distribución a usar en el clúster que se instalará para el proyecto recae sobre Cloudera. No solamente por ganar en la comparación por puntos -y en aspectos realmente importantes para el proyecto como la versión gratuita o el rendimiento- sino también por ser la empresa que más dedica esfuerzos en el desarrollo de Hadoop, participando activamente en su implementación y apostando siempre por las últimas versiones.

7. CASO DE USO PRINCIPAL

En este apartado se explica el diseño del caso de uso que se ha desarrollado sobre el clúster y la distribución de Cloudera. Se explica la arquitectura escogida y con qué herramientas se va a realizar cada etapa y el proceso de cada una de estas. Finalmente se detallarán algunas de las conclusiones sacadas de la implementación del caso de uso. El objetivo de este caso de uso no es obtener unos resultados lógicos sino el de comprobar cómo funciona Cloudera -en especial Hadoop- en términos de productividad, programación y monitorización de las actividades que se realizan.

7.1. CASO DE USO

El caso de uso escogido consiste en la descarga a través de las redes sociales de información relacionada con una empresa. Una vez recolectada la información se almacena en Hadoop para realizar procesos de análisis para determinar cuáles son las palabras y patrones más comentados cada día para poder realizar un análisis de sentimiento muy sencillo.

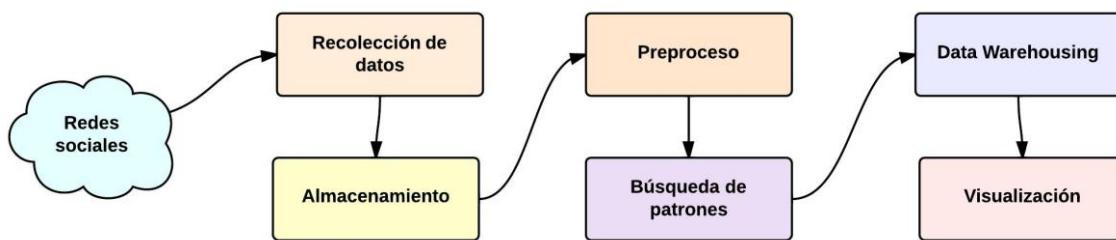


Figura 34: Flujo de información del caso de uso.

La Figura 34 muestra el flujo de la información del caso de uso y las diferentes etapas a implementar. Primero de todo se tiene una capa de recolección de datos que se conecta a una o varias redes sociales y descarga la información. La información descargada se almacena en alguna base de datos o sistema de ficheros para poder realizar un pre proceso de la información. Este pre proceso consiste en la purificación del texto para poder realizar mejor los algoritmos de búsqueda de patrones. Los resultados de la búsqueda se almacenan en una herramienta de data warehousing para poder ser visualizados.

7.2. ARQUITECTURA DE LA SOLUCIÓN

La Figura 35 muestra las herramientas seleccionadas para cada etapa del caso de uso:

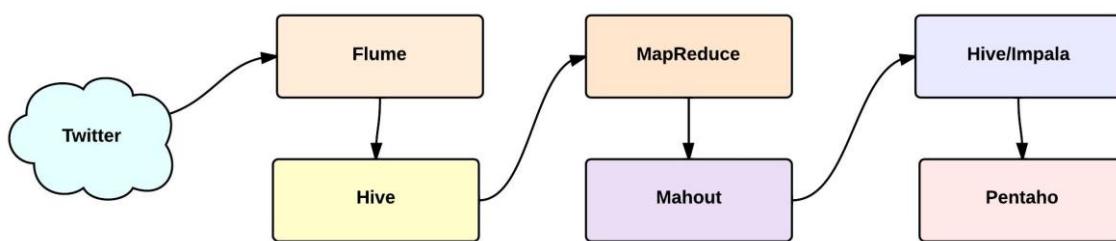


Figura 35: Herramientas seleccionadas para la realización del flujo de trabajo del caso de uso.

Para descargar datos de una o varias redes sociales se necesita de una herramienta altamente flexible y que sea capaz de ingerir muchos datos no estructurados en muy poco tiempo. De todas las herramientas estudiadas la que encaja mejor en ese perfil es Flume, ya que permite conectar fuentes de

Caso de uso principal

información personalizadas por el usuario y realizar diferentes arquitecturas para coleccionar grandes volúmenes de información.

Por otro lado, la red social desde la que se descargará la información es Twitter, puesto que es la red social que más facilidades -y menos protección de privacidad- da a la hora de descargar información. Además se tiene un sistema de búsqueda por hashtag que facilita mucho encontrar información un tema determinado.

Ya que los datos de Twitter son semi estructurados y después de pasar por Flume se estructurarán, en este caso de uso no se trabajará directamente con datos no estructurados como tal -el texto de los tweets es información no estructurada pero el tweet en sí está estructurado en varios campos (fecha, usuario, localización, texto, etc.)- y, por lo tanto, se puede trabajar con una herramienta de bases de datos como Hive, que además es compatible con Impala, la herramienta que se ha escogido para realizar data warehousing.

Finalmente, las dos etapas de procesos se realizan con MapReduce para la primera, ya que cada tweet ocupará una línea y su texto -la información procesada- sí que no está estructurada. El segundo proceso, el de la búsqueda de patrones, se realiza con Mahout que está pensado para esta tarea (y precisamente incluye un algoritmo MapReduce que busca patrones en un texto).

7.3. PROCESO

A continuación se detalla el proceso realizado para cada capa de la arquitectura Big Data diseñada. La explicación de la estructura y diseño del código desarrollado para los diferentes apartados se puede consultar en el apartado *F. Código* del anexo.

7.3.1. RECOLECCIÓN DE DATOS

La capa de recolección de datos se ha diseñado para ser lo más flexible posible -permitiendo realizar búsquedas paralelas- y para poder cambiar las búsquedas sin tener que re implementar ni una sola línea de código ni recompilar nada.

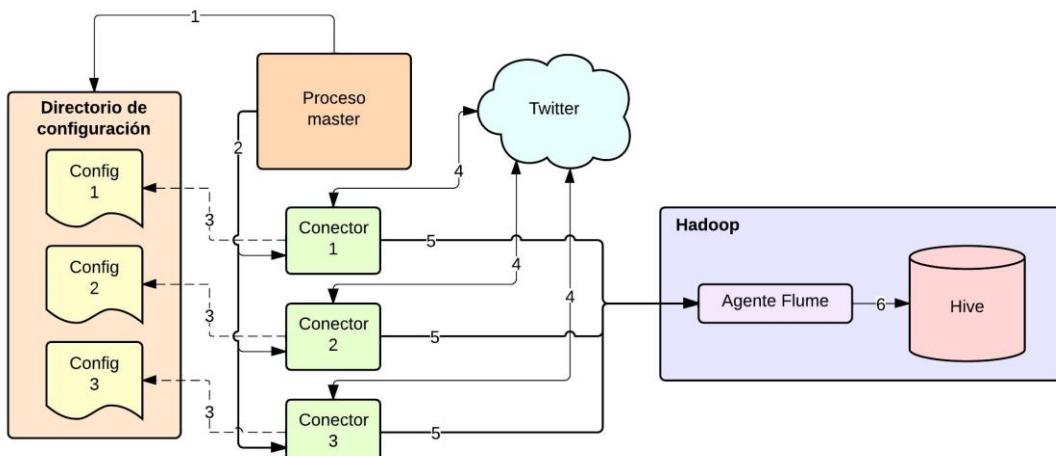


Figura 36: Arquitectura para la capa de recolección de datos.

La arquitectura de esta fase se compone de varios componentes:

- **Ficheros de configuración:** para cada búsqueda en paralelo habrá un fichero de configuración en el directorio de configuración que se le indique al proceso maestro. Este fichero indica el tipo de conector -a qué red social y qué modo de búsqueda: paginación o streaming; así como las claves

Caso de uso principal

de autentificación-, la búsqueda en sí -las palabras y las fechas con las que acotarla- y a qué agente de Flume envía los resultados obtenidos y a qué directorio se almacenará. También indica qué tipo de cliente será el que envíe datos al agente.

- **Proceso maestro:** lee el directorio de configuración y crea un cliente para cada fichero encontrado.
- **Conector:** recibe la configuración de su fichero de configuración y se conecta a la red social correspondiente para enviarle los datos al agente mediante un cliente Avro.
- **Agente:** recibe los datos de los clientes y los envía a HDFS o Hive.

```
application=twitter
connectorType=pagination
consumerKey=8HMreGQYx5e0nrhQU7Q
consumerSecret=BDBJI4g7VzifiXa4JnsnI11yH4RJpqDPIt6Rovw
accessToken=1041941390-aEQUM6cs9aDR5TRQCUVGhHnBU8bfIXa0MnU
accessTokenSecret=Vq0iHslofIDCHuZDzcacOzxTDRBTnKs7LjhWNRY
query="#apple"
count=100
since=2013-05-13
until=2013-05-14
clientType=rpcclientfacade
hostname=ip
port=11011
outputDirectory=apple
```

Figura 37: Ejemplo de configuración de un conector que buscará en twitter, por paginación, el hashtag #apple desde el 13 al 15 de mayo y lo enviará al agente Flume ip:11011 a través de un rpcclientfacade (cliente Avro).

La información descargada de Twitter, para cada tweet, es la siguiente:

- Identificador único.
- Texto.
- Identificador del usuario que lo ha publicado.
- Nombre de usuario que lo ha publicado.
- Desde qué aplicación ha sido publicado.
- La posición -si tiene- en forma de latitud y longitud.
- La fecha y la hora de ubicación.

7.3.2. ALMACENAMIENTO

Los tweets descargados por Flume se pasan a una tabla Hive con el esquema mostrado en la Figura 38:

- **id:** tipo string. Identificador del tweet.
- **latitude:** tipo double.
- **longitude:** tipo double.
- **posted:** tipo timestamp. Fecha y hora del tweet.
- **query:** tipo string. Búsqueda realizada por el conector para encontrar el tweet.
- **source:** tipo string. Fuente o aplicación desde la que el tweet ha sido publicado.
- **text:** tipo string. El texto posteado en el tweet.
- **userid:** tipo double. Identificador del usuario que ha posteado.
- **username:** tipo string. Nombre del usuario que ha posteado.

Columnas	Ejemplo
Nombre	Tipo
id	double
latitude	double
longitude	double
posted	timestamp
query	string
source	string
text	string
userid	double
username	string

Figura 38: Esquema de la tabla Hive que contiene los tweets.

Para facilitar la consulta de los datos se realiza un particionado usando la fecha del tweet, de manera que para cada día hay una partición de la tabla. Como los datos están en Hive, se pueden realizar consultas tipo SQL para hacer las consultas que se deseen. Por ejemplo, la consulta:

```
SELECT query, COUNT(*)
FROM comments
GROUP BY query
```

Agrupa todos los tweets de la tabla *comments* por su campo *query* y cuenta el total de entradas para cada grupo. El resultado se puede observar en la Figura 39.

Resultados	Consulta	Registro	Columnas
			query _c1
0	#ipod		9887232
1	#apple		40277416
2	#iphone		980720
3	#ipad		1961440

Figura 39: Número de tweets para cada consulta realizada.

En la *Figura 41* se puede ver unos ejemplos de los tweets descargados -en esta ocasión se buscaba algo más que los productos de la empresa Apple-.

Caso de uso principal

userid	username	latitude	longitude	posted	source	query
1.540524E7	Liam Parker	NULL	NULL	2013-10-14 01:40:24	web	#apple
6.03762961E8	gus	NULL	NULL	2013-10-14 01:40:29	Twitter for iPhone	#apple
6.7049467E7	Hussnain munawar	NULL	NULL	2013-10-14 01:40:40	twitterfeed	#apple
4.34058333E8	Katherine Hudson	NULL	NULL	2013-10-14 01:40:54	OS X	#apple
3.3076165E8	Kevin Bray	NULL	NULL	2013-10-14 01:41:06	Twitter for iPhone	#apple
2.46701806E8	appnews	NULL	NULL	2013-10-14 01:41:14	IFTTT	#apple
7.0478756E7	Junior Senior	NULL	NULL	2013-10-14 01:41:28	web	#apple
3.730835E8	Nia	NULL	NULL	2013-10-14 01:41:38	iOS	#apple
1.922033702E9	Sandra Flood	NULL	NULL	2013-10-14 01:42:00	TweetAdder v4	#apple
9.18145513E8	jbtvtools.com	NULL	NULL	2013-10-14 01:42:00	TweetAdder v4	#apple
1.03349993E9	•RayRay•	NULL	NULL	2013-10-14 01:42:38	Instagram	#apple
1.667725519E9	Tee Array	NULL	NULL	2013-10-14 01:42:38	HootSuite	#apple
4.5126062E7	NineZero	NULL	NULL	2013-10-14 01:42:47	Google	#apple
4.5126062E7	NineZero	NULL	NULL	2013-10-14 01:42:48	Google	#apple
4.5126062E7	NineZero	NULL	NULL	2013-10-14 01:42:50	Google	#apple
4.5126062E7	NineZero	NULL	NULL	2013-10-14 01:42:50	Google	#apple

Figura 40: Muestra de los tweets capturados por Flume y ya almacenados en Hive.

id	text
3.8953612475590246E17	Reinforcements for my ailing-long-lived-on-life-support iPhone 3GS have arrived. A new iPod Touch. Exciting. iOS7 here I come. #apple #ios
3.8953614671730688E17	RT @applenws: Google's 'Ingress' coming to iOS in 2014 http://t.co/BYZIGosWjt #apple
3.8953619443590349E17	Google's 'Ingress' coming to iOS in 2014: Google's popular mobile augmented-reality game called Ingre... http://t.co/xh6LK3C10I #Apple
3.8953625458219418E17	mac book conference in the library foyer #apple
3.8953630225506304E17	@BizNasty2point0 congratulations on filling your quota for the season #apple http://t.co/BE5rB8V9Eg
3.8953633517196902E17	iOS 7 app crash rates are higher for iPhone 5S than other Apple smartphones - Mobile Burn http://t.co/xVXa05T8Tq #apple iOS 7 app crash r...
3.8953639634967757E17	Apple TV HD digital media receiver with Wi-Fi and AirPlay #Apple #TV #HD #digital #media #receiver #WiFi #AirPlay http://t.co/GKGLPEjXU
3.8953643864925389E17	The fam @ #apple picking!! http://t.co/h2U9YJjTvp
3.8953652821061222E17	Great #Mac #Italian language learning software http://t.co/Ng0l1s657i #apple #education #backtoschool #easy
3.8953652825255117E17	This is pretty neat: http://t.co/vNFfzq4t59 Learn #French spelling on your #mac #apple #backtoschool
3.8953668754381619E17	Shots!! #jagor #apple #sourz #strawberry #sourz #sambuca #lush #sunday #night #drinking #love #it... http://t.co/94fluBrJQ7
3.895366902030295E17	New #ios7 shirts are up!Extremely limited!#ios #iphone5s #apple \$aapl #iphone #ipod #ipadhttp://t.co/FhiZAIgUKI http://t.co/4iGAuq0Rd3
3.8953672525480346E17	Apple TV adds ESPN channel http://t.co/cpdajr4reO #Apple
3.8953673104713318E17	For Apple's new iPad, it's (r)evolution, baby http://t.co/4VLYm7tp6r #Apple
3.8953673827814605E17	Apple announces iPhone 5s: Touch ID fingerprint security, 64-bit A7 CPU, new gold... http://t.co/nZmB9iyGxA #Apple
3.895367399935959E17	Apple Finally Has Its Own Blue Screen of Death http://t.co/2VTQSott8b #Apple

Figura 41: Muestra de los tweets capturados por Flume y ya almacenados en Hive.

7.3.3. PROCESAMIENTO

Una vez almacenados los tweets, se realizan dos procesados sobre ellos: el primero para seleccionar solo el texto y pasar unos diccionarios para depurar el texto. El segundo procesado es el que realiza la búsqueda de patrones sobre el texto seleccionado y procesado.

La depuración de texto se realiza para que al buscar patrones los resultados sean más exactos. Aunque el objetivo de realizar este caso de uso no es para obtener resultados sino para evaluar Hadoop como herramienta Big Data, se ha intentado seguir un proceso lógico a la hora de realizar la implementación del caso de uso. En la depuración se aplican los siguientes pasos sobre el texto:

1. Eliminación de caracteres no alfanuméricos. Ya que lo que se busca son patrones de palabras, los símbolos de puntuación y otros caracteres no interesan en absoluto, además que podrían llegar a causar problemas.
2. Filtrado de tweets con palabras no deseadas. Debido a que Twitter es una red social muy popular suele haber muchos comentarios que pueden ser considerados spam, basura o publicidad viral. Por esto mismo se buscan los tweets que incluyan palabras clave para eliminarlos. No se eliminan cuando se capturan en la fase de recolección de datos por dos motivos: se quería obtener la mayor cantidad de datos para ver el comportamiento de la solución y puede ser que en un futuro interesa realizar estudios sobre la cantidad de spam, por lo que pasaría a ser información de valor.
3. Eliminación de palabras sin valor. Se eliminan las palabras que no contienen ningún valor para el estudio a realizar. En esta categoría entran por ejemplo los artículos, pronombres, preposiciones, etc.
4. Sustitución de palabras por una forma simple. Algunas palabras, como los verbos, pueden aparecer en varios textos con distintas formas pero con el mismo significado (como las conjugaciones verbales). En este paso se buscan estas palabras y se sustituyen por una palabra simple para enfatizar su valor (en el caso de los verbos, por su infinitivo).

```
every bar should have iphone charger apple justsaying
ios7 apple seene iphone makes creating interactive 3d
photos fun easy iphone apple
appleactumac iphone 5s 5c debuting prepaid carrier
cricket october 25 apple
financial calculator pv fv mortgage investment discounts
financial app iphone apple ipad
```

Figura 42: ejemplo de cómo queda el texto de un tweet después de pasar por el pre proceso.

Para realizar este pre proceso sobre el texto se han implementado unas clases diccionario que leen un fichero -llamado diccionario- que contienen las listas de palabras a buscar y realizan los cambios comentados al texto que se les indica. Además, los diccionarios se ejecutan sobre un MapReduce que tiene como entrada el texto de los tweets y como salida deja en un fichero temporal el texto depurado después de ejecutar los diccionarios.

Caso de uso principal

El segundo procesado se realiza sobre el texto depurado y consiste en la utilización de la librería Mahout para encontrar los patrones más frecuentes. La función que se utiliza se llama Frequent Pattern Growth o FPM. Para ejecutar Mahout en este caso no hace implementar ningún código ya que con la llamada de

```
bin/mahout fpg \
    -i /input/file \
    -o patterns \
    -k 50 \
    -s 2
```

Figura 44: Llamada por línea de comandos del algoritmo FPM de Mahout. En este caso lee el texto de /input/file y escribe los resultados en patterns.

```
quiero comprarme un macbook pro porque me gusta apple
i want to buy a macbook pro because i like apple
you prefer sony vaio instead macbook pro
listening music with my apple ipod touch
apple ipod touch are expensive
apple expensive
```

Figura 45: Ejemplo de entrada para la función de búsqueda de patrones de Mahout.

```
Key: apple: Value: ([apple],5), ([apple, macbook, pro],2),
      ([apple, ipod, touch],2), ([apple, macbook],2), ([apple,
      ipod],2), ([apple, expensive],2)
Key: expensive: Value: ([apple, expensive],2)
Key: ipod: Value: ([apple, ipod, touch],2), ([apple, ipod],2)
Key: macbook: Value: ([macbook, pro],3), ([macbook],3),
      ([apple, macbook, pro],2), ([apple, macbook],2)
Key: pro: Value: ([macbook, pro],3), ([apple, macbook, pro],2)
Key: touch: Value: ([apple, ipod, touch],2)
Count: 6
```

Figura 43: Resultado del proceso de Mahout de búsqueda de patrones con la entrada de la Figura 45.

la Figura 44, donde se le indica el fichero de entrada y el de salida, es más que suficiente. En la Figura 45 se puede observar un ejemplo de entrada para el algoritmo FPM y en la Figura 43 la salida que generaría. Esta salida consiste en una lista de claves -las palabras que más aparecen en el texto- junto con una lista de valores para cada clave -las palabras con las que más aparecen-.

7.3.4. DATA WAREHOUSING

Los resultados de Mahout se introducen de nuevo en una tabla de Hive para poder ser consultados a modo de Data Warehouse. La elección de Hive es porque la herramienta Impala -exclusiva de Cloudera- es totalmente compatible al funcionar con los mismos metadatos. Los tiempos de Impala una vez cargada la tabla de resultados en memoria son mucho más rápidos -del orden de segundos, por los

[Caso de uso principal](#)

minutos que tarda una consulta Hive-, haciendo que la visualización sea mucho más interactiva entre el usuario y la aplicación.

[Go to parent directory](#)

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
_SUCCESS	file	0 B	3	128 MB	2013-10-18 05:38	rw-r--r--	cloudera	supergroup
part-r-00000	file	82.19 KB	3	128 MB	2013-10-18 05:38	rw-r--r--	cloudera	supergroup
part-r-00001	file	80.22 KB	3	128 MB	2013-10-18 05:38	rw-r--r--	cloudera	supergroup
part-r-00002	file	84.31 KB	3	128 MB	2013-10-18 05:38	rw-r--r--	cloudera	supergroup
part-r-00003	file	84.50 KB	3	128 MB	2013-10-18 05:38	rw-r--r--	cloudera	supergroup
part-r-00004	file	82.25 KB	3	128 MB	2013-10-18 05:38	rw-r--r--	cloudera	supergroup
part-r-00005	file	80.25 KB	3	128 MB	2013-10-18 05:38	rw-r--r--	cloudera	supergroup

Figura 47: Los resultados de un proceso Mahout son como los de un MapReduce. Se almacenan en un directorio con un fichero de `_SUCCESS` si la ejecución ha sido correcta y con un fichero con los resultados de cada Reducer.

```
mahout seqdumper -i /directorio/HDFS/con/los/resultados/mahout
```

Figura 46: Comando para leer los resultados de un proceso Mahout. Tiene como salida algo parecido a la Figura 48.

7.4. RESULTADOS

A continuación se muestran algunos de los resultados obtenidos en la búsqueda de patrones más comunes. Los resultados obtenidos corresponden a tweets procesados en un par de horas buscando la marca Apple, y se han seleccionado algunos casos con algún interés.

Key: 4s: Value: ([apple, iphone, 4s],157), ([apple, iphone, 5, case, by, 4, cover, blue, 4s, dots, inspireuart, iphoncase, polka],95)

Key: accommodate: Value: ([apple, ipad, be, mini, 2, display, retina, thicker, might, slightly, accommodate],148)

Key: 4g: Value: ([apple, 99, ebay, deal, 4g],193), ([apple, ipad, 7, mini, 99, white, verizon, ebay, fi, 16gb, silver, deal, 4g, buynow, 399, 9in, cellular, wi],112)

Key: amazing: Value: ([apple, amazing],269), ([apple, osx, design, version, amazing, try, product, litos, solidworks, waiting, whit],116)

Key: approved: Value: ([apple, iphone, 5c, 5s, mobile, china, approved],74))

Key: arrived: Value: ([apple, 5s, just, iphone5s, arrived],37)

Key: brilliant: Value: ([apple, new, enter, ios7, sell, around, lets, world, made, ny, built, ui, meet, brilliant, comingsoon, mapapp, simplified],58)

Key: calculators: Value: ([apple, iphone, ipad, apps, fitness, calc, calculators, military, physical, usmc],102)

Key: canadian: Value: ([apple, s, blackberry, headquarters, d, appadvice, drive, hosts, recruitment, canadian, near],67), ([apple, s, blackberry, headquarters, d, appadvice, drive, hosts, recruitment, canadian],67)

Key: cases: Value: ([apple, cases],162), ([apple, 5, have, now, 2, ipod, 99, touch, right, ebay, awesome, deal, 4g, cases, freeshipping, griffin],81)

Key: china: Value: ([apple, china],527), ([apple, other, china],372), ([apple, osx, language, us, way, need, other, around, tool, travel, china],335)

Key: computer: Value: ([apple, computer],558), ([apple, computer, deals],306), ([apple, book, computer, deals, life, times, ii, simplicity, sophistication],116), ([apple, ipad, 7, case, mini, inch, computer, deals, cover, smart, sleep, 9, leather, auto, blason, wake],102)

Key: connectivity: Value: ([apple, iphone, is, how, 5, its, has, out, know, shit, any, help, wifi, find, connectivity, fault, techies],95), ([apple, iphone, is, how, 5, its, has, out, know, shit, any, help, wifi, find, connectivity, fault],95), ([apple, iphone, is, how, 5, its, has, out, know, shit, any, help, wifi, find, connectivity],95)

Key: incredible: Value: ([apple, incredible],137), ([apple, jobs, was, steve, incredible],81), ([apple, s, new, be, here, world, headquarters, model, scale, tomorrow, incredible, productdesign, voted],56),

Figura 48: Resultados de procesar los tweets correspondientes a dos horas con el sistema implementado en Hadoop. Algunas de los patrones más interesantes están resaltados en negrita.

Una lectura rápida de estos resultados nos deja entrever que durante esos días la actividad de sobre Apple se centró en los siguientes aspectos:

- Al parecer en Twitter se realizan muchos anuncios de fundas para los dispositivos móviles de Apple (se habla mucho de cases para los distintos modelos).
- Estos tweets son de unos días antes del anuncio de un producto nuevo (iPad 5 y iPad mini 2), por lo que había una gran cantidad de rumores acerca de las pantallas que iban a tener.
- También salió la noticia de que Apple realizaba anuncios cerca de la zona donde BlackBerry tiene sus oficinas para captar nuevos empleados.
- En general la valoración sobre la compañía y sus productos es positiva (palabras clave como incredible o amazing así lo atestiguan).
- también se habla mucho de la salida de los iPhones en China.

7.5. CONCLUSIONES

Al empezar a desarrollar la primera solución con Hadoop -Cloudera, más específicamente- puede parecer al principio un poco confuso, por la cantidad de herramientas nuevas que hay y por los nuevos paradigmas en los que el desarrollador se encuentra. Por esto mismo, y aunque se haya realizado un estudio teórico de la solución y de las distintas herramientas, durante el desarrollo del caso de uso hubo varias modificaciones en el diseño. Esto es normal teniendo en cuenta todas las novedades y la curva de aprendizaje de herramientas como Flume y MapReduce. Una vez se entra en materia el desarrollo se vuelve más ameno y sencillo incluso que cuando se implementa una solución más tradicional, gracias a las diferentes herramientas de monitorización.

Otro aspecto a tener en cuenta de Hadoop es la potencia que tiene gracias a su escalabilidad, se pueden diseñar casos de uso hasta ahora impensables con infraestructuras de coste medio o bajo como los usados en este proyecto y obtener resultados muy competitivos comparados con una solución de más presupuesto. Hay que recordar que todo el software usado es gratuito -aunque Cloudera tenga versión de pago- y que la infraestructura que se ha usado no es la más adecuada -cuatro máquinas virtuales ejecutándose en un mismo servidor y compartiendo recursos físicos, que en casos como el disco son cuellos de botella importantes al trabajar en paralelo-.

Finalmente destacar la gran flexibilidad a la hora de diseñar procesos de Hadoop. Cualquiera de las capas puede diseñarse independientemente de las otras e incluso pueden cambiarse por completo. En la capa de recolección de datos, con la herramienta Flume, puede extenderse la funcionalidad añadiendo más redes sociales o más búsquedas de manera muy sencilla (añadiendo más clientes) y sin tener que realizar cambios en el almacenamiento. Una vez almacenada la información, al no tener que desechar datos y tenerlos sin estructurar, podrían añadirse nuevos análisis -como el de spam comentado con anterioridad-. Esta flexibilidad permite que las soluciones estén totalmente preparadas para los cambios que se puedan producir en un futuro.

Por lo tanto la conclusión es positiva ya que Hadoop cumple con las cinco V comentadas en el apartado 3.1. *Las cinco V*.

8. CASO DE USO EXTRA

Durante la realización del proyecto surgió la posibilidad de realizar una pequeña aplicación para analizar unos ficheros de logs. Los ficheros contenían entradas con información diversa -tiempo de ejecución, estado, identificador de usuario, conexión, etc.- de diferentes funciones realizadas por distintos usuarios. Cada ejecución podía generar varias entradas desde el inicio -marcado con la palabra clave START- al final -marcado con END-. El objetivo de la aplicación era la de calcular el tiempo de ejecución de las operaciones que aparecían en los logs.

```
2013-10-01 16:29:12,506 INFO [CB_LOE] Production <DEBUG> Function (34177548 @ Write In RTD Resultados): START  
2013-10-01 16:29:12,506 INFO [CB_LOE] Production <DEBUG> Function (34177548 @ Write In RTD Resultados): Initializing DB connection  
2013-10-01 16:29:12,506 INFO [CB_LOE] Production <DEBUG> Function (34177548 @ Write In RTD Resultados): SQL Statement executed  
2013-10-01 16:29:12,507 INFO [CB_LOE] Production <DEBUG> Function (34177548 @ Write In RTD Resultados): Closing DB connection  
2013-10-01 16:29:12,507 INFO [CB_LOE] Production <DEBUG> Function (34177548 @ Write In RTD Resultados): END
```

Figura 49: ejemplo de las entradas en los ficheros de log generadas por una ejecución de una función.

```
2013-10-01 16:29:12,506 2013-10-01 16:29:12,507 INFO Function 34177548 Write In RTD Resultados: 0,001
```

Figura 50: Resultado esperado del ejemplo de la Figura 49.

Aprovechando que se tenía que realizar esta pequeña aplicación se diseñaron dos pruebas adicionales a las diseñadas para la fase de pruebas.

8.1. DISEÑO DE PRUEBAS

Las dos pruebas diseñadas para este pequeño caso de uso son las siguientes:

Comparación de una solución MapReduce con una tradicional

Gracias a que el problema se resuelve con un algoritmo sencillo, la intención de esta prueba es realizar dos soluciones; una que use MapReduce -MRv2- y otra que sea más tradicional, es decir una aplicación Java. El objetivo es comparar los resultados obtenidos y los tiempos (escalando el clúster para la solución MapReduce) y, adicionalmente, comparar la productividad a la hora de desarrollar una solución Hadoop.

[Caso de uso extra](#)**Comprobar la incidencia en el rendimiento del tamaño y número de la entrada**

Al tener una gran cantidad de ficheros de logs, el caso de uso se prestaba para realizar esta prueba. Teóricamente MapReduce está pensado para tener un tamaño de entrada grande, tanto en número como en tamaño individual de cada fichero -que idealmente deberían ocupar más de un bloque-. Esto es porque cada vez que se lanza un Mapper sobre un bloque de un fichero se pierde una pequeña cantidad de tiempo inicializando el proceso, de manera que si se ejecutan muchos Mapper esta cantidad de tiempo se acumula y termina siendo grande. De esta manera, la prueba consiste en la ejecución de la solución MapReduce con varios tipos de entrada: una con muchos ficheros de poco tamaño y otra con pocos ficheros de gran tamaño. La última entrada se genera concatenando los ficheros pequeños de la primera entrada.

La implementación de la solución secuencial está documentada en el apéndice *F. Código en el apartado ¡Error! No se encuentra el origen de la referencia..*

8.2. RESULTADOS DE LAS PRUEBAS

La *Tabla 19* contiene los resultados de la ejecución de la solución secuencial tanto para una entrada con muchos ficheros pequeños como para una con pocos ficheros y grandes. Para cada una de las dos pruebas se han realizado cinco ejecuciones para obtener resultados más fiables. Como se puede observar el proceso tarda casi una hora y es independiente del formato de la entrada (aunque haya una diferencia de dos minutos no es significativa y podría ser causa del azar, ya que hay varios factores que pueden tener influencia en el rendimiento de una ejecución, aunque sea mínima).

Tamaño de entrada	Número de ficheros	Repetición	Tiempo total del proceso (s)	Media (s)	Media (min)	Tamaño del resultado	# Líneas del resultado
29 GB	1435	1	3473	3383,4	56,39	2,17 GB	25819838
		2	3366			2,17 GB	25819838
		3	3405			2,17 GB	25819838
		4	3321			2,17 GB	25819838
		5	3352			2,17 GB	25819838
29 GB	30	1	3364	3248,6	54,14	2,17 GB	25821836
		2	3271			2,17 GB	25821836
		3	3263			2,17 GB	25821836
		4	3234			2,17 GB	25821836
		5	3111			2,17 GB	25821836

Tabla 19: Resultados de la ejecución de la solución secuencial.

La *Tabla 20* muestra los resultados de la ejecución de la solución MapReduce. La tabla muestra los resultados de escalar la configuración de MapReduce (con 1, 3 o 6 Reducers) y también los resultados de la ejecución para ficheros grandes. Como en el último caso, se realizan cinco ejecuciones para cada prueba para obtener resultados fiables. Como se puede observar el escalado de los Reducers tiene un efecto pequeño (se llega a obtener una mejora de hasta 400 segundos, casi siete minutos). Esta poca influencia es debido a que la mayor parte del tiempo de proceso está en los Mappers, que al lanzar muchos -uno por fichero ya que ninguno de los logs ocupan más de un bloque- también pierden mucho tiempo en ejecutarse.

[Caso de uso extra](#)

Por otro lado, la ejecución con pocos ficheros pero mucho más grandes sí que tiene un efecto más significativo, pues se consigue mejorar el tiempo bajándolo más de la mitad. La explicación lógica es que esta vez los Mappers si tienen trabajo a realizar -tardan 29 segundos cada uno, por los 10 segundos de las ejecuciones anteriores- y se lanzan muchos menos -229 por los 1.435 de antes-. Esto significa que se está aprovechando la arquitectura de MapReduce.

Tamaño de entrada	Número de ficheros	# Maps	# Reducers	Repetición	Tiempo total del proceso (seg)	Media (s)	Tiempo medio por map (s)
29 GB	1435	1435	1	1	3926	3947,6	10
				2	3903		
				3	4052		
				4	3912		
				5	3945		
29 GB	1435	1435	3	1	3778	3776,6	10
				2	3819		
				3	3723		
				4	3769		
				5	3794		
29 GB	1435	1435	6	1	3500	3573,8	10
				2	3632		
				3	3633		
				4	3480		
				5	3624		
29 GB	30	229	6	1	1717	1686,4	29
				2	1683		
				3	1640		
				4	1715		
				5	1677		

Tabla 20: Resultados de las ejecuciones de la solución MapReduce.

8.3. CONCLUSIONES

A priori la comparativa entre la solución secuencial y la MapReduce parece decantarse del lado de la segunda, ya que con ficheros grandes consigue reducir el tiempo a más de la mitad. Pero hay que tener en cuenta varios factores:

- sólo vale la pena la solución MapReduce en caso de tener los ficheros de gran tamaño y, en este caso de uso, esto implica realizar un proceso adicional de consolidación de los ficheros pequeños de log a unos de mayor tamaño. Este proceso no se contabiliza en el tiempo total y es del orden de unos 10 minutos.
- al tiempo añadido anterior hay que sumar el coste de subir los ficheros a HDFS, que es desde donde trabaja MapReduce. Este proceso no solamente incluye el hecho de mover los datos sino también replicarlos. Este tiempo dura alrededor de 40 minutos.

Teniendo en cuenta estos factores, sólo vale la pena usar MapReduce si los datos con los que se va a trabajar pasan por más de un procesamiento, de manera que se amortizaría el coste de consolidar y

[Caso de uso extra](#)

almacenar los datos en HDFS. En este caso particular no ocurría esto, ya que la única intención era realizar el cálculo de tiempo de cada operación.

Otra conclusión que se saca de esta prueba es que indudablemente es aconsejable trabajar con ficheros de grandes tamaños -a poder ser que ocupen varios bloques- en lugar de muchos ficheros pequeños - que no ocupen ni un bloque-.

Por otro lado, la implementación de la solución secuencial ha resultado muy sencilla ya que es simplemente un programa Java que leía línea a línea un fichero. Por otro lado, la implementación del MapReduce ha sido algo más problemática ya que se ha tenido que adaptar el algoritmo al esquema de Mapper y Reducer e implementar un Combiner y un Partitioner.

9. PRUEBAS

Las pruebas son uno de los apartados centrales del proyecto. En este apartado se describen las distintas pruebas que se han realizado sobre el clúster. Se han dividido las pruebas en diferentes apartados según la naturaleza de la herramienta o aspecto a estudiar. Para poder identificar mejor las pruebas y sus conjuntos se representa cada una con su etiqueta o código.

9.1. ADMINISTRACIÓN

Esta sección de las pruebas corresponde a aquellas con las que se interactúa con los aspectos de administración, configuración y monitorización de la distribución, que trae sus propias herramientas para la realización de estas tareas. De esta manera, todas las valoraciones se han realizado sobre Cloudera Manager y han servido también para evaluar su usabilidad y facilidad de uso. Es decir que estas pruebas van más enfocadas a la usabilidad e la herramienta y sus funcionalidades. Hay que tener en cuenta que la versión evaluada es la gratuita, que tiene menos funcionalidades que la de pago.

Nombre	Código	Descripción breve
Instalación clúster	A1	Evaluar el proceso de instalación de CDH4 en un nuevo clúster.
Escalabilidad nodos	A2	Evaluar el proceso de añadir o quitar un nodo en el clúster.
Escalabilidad servicios	A3	Evaluar el proceso de añadir o quitar un servicio.
Monitorización	A4	Se permite monitorizar el estado de salud del clúster.
Configuración	A5	Evaluar el proceso de modificar la configuración de un servicio.

Tabla 21: Listado de las pruebas realizadas en el apartado de administración.

9.1.1. A1 - INSTALACIÓN CLÚSTER

El proceso de instalación del clúster realizado en el proyecto está explicado en el apartado *D. Procesos de instalación* del anexo. Cloudera permite instalar y configurar un clúster a través de un proceso de instalación muy sencillo a través de su cliente web. De cara al usuario es un paso muy sencillo, sin prácticamente ningún requerimiento de conocimiento técnico, con una interfaz paso a paso y que permite hacer las primeras configuraciones de manera muy visual y casi automatizada -descarga e instala todos los paquetes sin que el usuario tenga que preocuparse de las dependencias en ningún caso-.

Los únicos puntos delicados durante el proceso es cuando aparece un error (generalmente por temas de puertos en las distintas máquinas que formarán el clúster) y la configuración previa de los servidores para interactuar entre ellos. En ambas partes sí se requiere de conocimientos técnicos en el ámbito de administración de sistemas, aunque sea un nivel muy básico.

9.1.2. A2 - ESCALABILIDAD DE NODOS

Añadir o quitar un nodo del clúster es igual o más sencillo que el de instalar de cero un clúster entero. Sigue la misma interfaz paso a paso que en la prueba A1 y él mismo se encarga de las dependencias (por ejemplo, al quitar un nodo hay que tener en cuenta los servicios instalados en él).

Al igual que en el caso anterior, se requiere de conocimientos en administración de sistemas ya que el servidor a añadir debe estar configurado para ser visible por el clúster.

Pruebas

9.1.3. A3 - ESCALABILIDAD DE SERVICIOS

En el caso de los servicios es aún más sencillo que los demás. Cloudera Manager deja escoger entre los servicios disponibles y realiza la configuración mediante la interfaz web, dejando escoger la localización de los servicios entre las máquinas del clúster. También se encarga de comprobar que el clúster cumple con las dependencias para poder realizar el despliegue del servicio. Este paso está ejemplificado con pantallas en el apartado *E. Procesos de configuración* del anexo.

En este caso no se requiere de ningún conocimiento técnico más allá de saber qué se está instalando y en qué máquinas es adecuado instalarlo para obtener un buen rendimiento.

9.1.4. A4 - MONITORIZACIÓN

En el apartado de monitorización Cloudera Manager ofrece varios niveles de monitorización:

- **Estado de los nodos:** permite ver cuál es el estado de los nodos conectados al clúster.
- **Estado de los servicios:** para cada herramienta desplegada, lista el estado de los servicios y los posibles errores o avisos que puedan generar.
- **Vista de los logs:** se pueden visualizar los logs generados por los servicios en funcionamiento para poder hacer un mejor seguimiento in situ de las trazas de errores.
- **Monitorización a través de sus propios servicios:** Cloudera Manager cuenta con sus propios servicios -también monitorizables con las opciones anteriores- que permiten hacer un seguimiento del funcionamiento, carga de trabajo y otras opciones de las herramientas desplegadas.

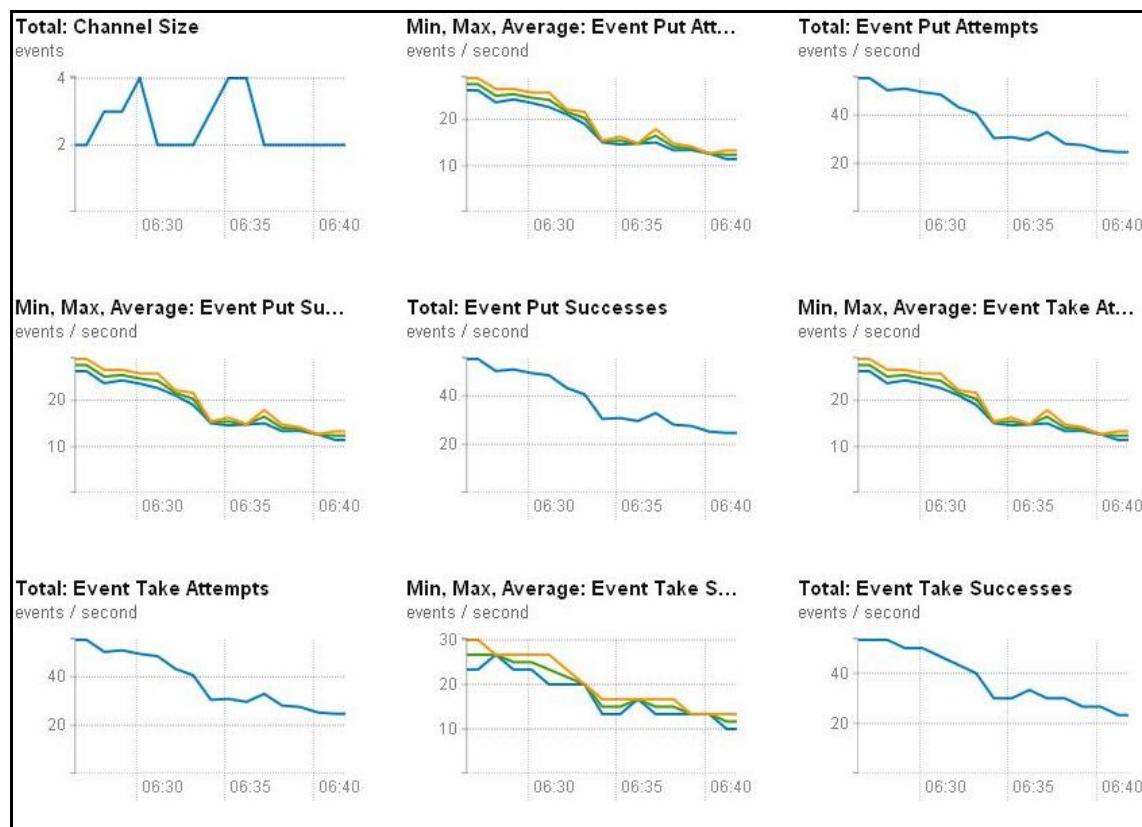


Ilustración 16: Monitorización de los agentes de flume desplegados a través de Cloudera manager.

En cualquiera de estos casos, Cloudera Manager es una herramienta sumamente útil a la hora de monitorizar ya que aísla al usuario o administrador de tener que lidiar con la gran cantidad de logs que

Pruebas

genera Hadoop, así como facilitar la interacción del propio usuario con el clúster para solucionar los errores o eventos que generen contratiempos.

9.1.5. A5 - CONFIGURACIÓN

En el punto de configuración, Cloudera Manager facilita el acceso a todas las opciones de los diferentes servicios mediante una interfaz visual como la que se ve en la *Ilustración 35* en el anexo, que permite configurar un agente de Flume. Básicamente lo que hace de cara al usuario es sustituir los ficheros de configuración -generalmente ficheros en formato XML repartidos por el sistema de ficheros- por un formulario a través de su interfaz web centralizándolo todo en un solo sitio. Además contiene explicaciones de para qué sirve cada aspecto de la configuración.

9.2. HDFS - ALMACENAMIENTO

Las pruebas realizadas sobre HDFS están encaradas sobre todo al funcionamiento automatizado de sus principales virtudes: la replicación de datos y la tolerancia a fallos. También se han realizado pruebas de rendimiento pero a causa del diseño del clúster, limitado en cuanto a número de nodos, no se ha podido realizar pruebas de escalabilidad debido a que el número mínimo de nodos DataNode aconsejado es de tres, el máximo del que se disponía.

Nombre	Código	Descripción breve
Replicación datos	HD1	Se permite añadir un fichero al sistema de ficheros y este se replica automáticamente a otros nodos si el factor de replicación es superior a 1.
Rendimiento	HD2	Calcular con qué velocidad se añade un fichero en el sistema de ficheros.
Tolerancia a fallos	HD3	Si un nodo se cae, y el factor de replicación es superior a 1, los ficheros continúan siendo accesibles.

Tabla 22: Listado de las pruebas realizadas en el apartado de recolección de datos.

9.2.1. HD1 - REPLICACIÓN DE DATOS

Esta prueba se ha realizado en paralelo junto con HD2, para cada fichero que se añadía se comprobaba que se replicaba según el factor de replicación establecido (tres en este caso). En ninguno de los casos se ha dejado de replicar bloques de ningún fichero, además de replicarse con

Total number of blocks: 200
9138498662761311927: 7.110.8.23:50010 View Block Info 7.110.8.22:50010 View Block Info 7.110.8.21:50010 View Block Info
-2761169775562361233: 7.110.8.23:50010 View Block Info 7.110.8.22:50010 View Block Info 7.110.8.21:50010 View Block Info
5137899440932880069: 7.110.8.23:50010 View Block Info 7.110.8.22:50010 View Block Info 7.110.8.21:50010 View Block Info
-6208634936870400560: 7.110.8.23:50010 View Block Info 7.110.8.22:50010 View Block Info 7.110.8.21:50010 View Block Info
-378310171874420866: 7.110.8.23:50010 View Block Info 7.110.8.22:50010 View Block Info 7.110.8.21:50010 View Block Info
-9033515255928099211: 7.110.8.23:50010 View Block Info 7.110.8.22:50010 View Block Info 7.110.8.21:50010 View Block Info
2167797110776843719: 7.110.8.23:50010 View Block Info 7.110.8.22:50010 View Block Info 7.110.8.21:50010 View Block Info
5915687913166392131: 7.110.8.23:50010 View Block Info 7.110.8.22:50010 View Block Info 7.110.8.21:50010 View Block Info
-6755968614403662830: 7.110.8.23:50010 View Block Info 7.110.8.22:50010 View Block Info 7.110.8.21:50010 View Block Info

Figura 51: Lista de bloques y su replicación de un fichero de 25 GB (128MB * 200 bloques = 25.600MB).

9.2.2. HD2 - RENDIMIENTO

El objetivo de esta prueba es medir cuánto tiempo tarda un sistema HDFS con las características de nuestro clúster en añadir ficheros nuevos. La Tabla 23 muestra los resultados obtenidos:

hadoop fs -put		
Tamaño de bloque = 128 MB		
Factor de replicación = 3		
Tamaño de fichero (MB)	Tiempo (s)	Media (s)
128	3,821	4,851
	4,307	
	4,258	
	4,682	
	7,185	

Pruebas

		Pruebas
256	6,192	14,406
	22,338	
	18,446	
	9,166	
	15,888	
384	26,082	26,219
	30,010	
	18,349	
	40,755	
	15,900	
512	46,870	37,941
	31,235	
	29,940	
	40,037	
	41,621	
1.024	67,554	77,013
	69,349	
	76,857	
	83,780	
	87,525	
5.120	569,390	486,078
	495,000	
	458,000	
	475,000	
	433,000	
25.600	2.708,615	2.627,285
	2.631,707	
	2.670,103	
	2.543,800	
	2.582,200	

Tabla 23: Resultados de añadir varios ficheros de diferente tamaño a HDFS.

Como se puede observar, para cada fichero de distinto tamaño se han realizado cinco pruebas para obtener unos resultados más verídicos. En el Gráfico 1 se puede apreciar que los resultados no son linealmente escalables en el caso de los ficheros de menor tamaño -puesto que no ocupan tantos bloques- pero que a medida que el tamaño de fichero crece, sí tiene una proporción linealmente escalable. Una explicación para esto es que el fichero tiene que replicarse y HDFS debe decidir cuál es la mejor disposición de los nodos, trabajo que añade un tiempo considerable a la adición del fichero y por lo tanto que el tiempo aumente con el tamaño del fichero.

Pruebas

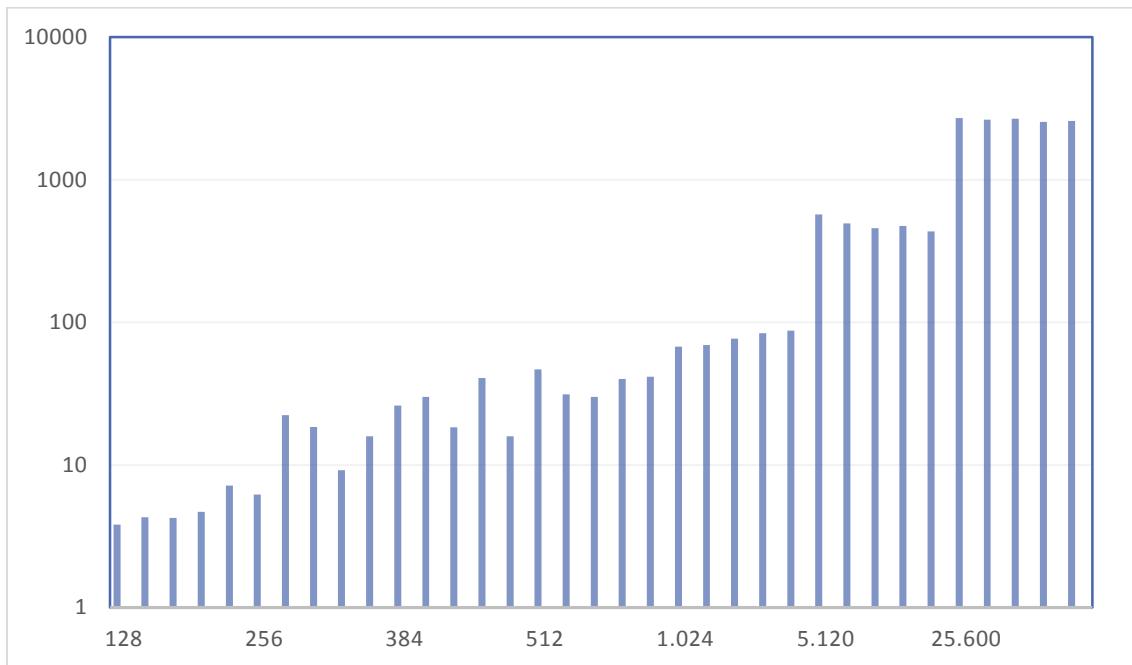


Gráfico 1: Tiempo medio para añadir un fichero a HDFS según su tamaño en escala logarítmica.

9.2.3. HD3 - TOLERANCIA A FALLOS

En esta prueba se han hecho varias comprobaciones de manera paralela aprovechando la configuración del clúster (tres DataNodes):

- La caída de un DataNode no implica la pérdida de datos ya que están replicados.
- Cuando uno de los DataNodes está caído, al añadir un fichero nuevo sus bloques se replican en dos nodos solamente (no cumplirá con el factor de replicación, que es de tres).
- En el caso anterior, cuando el DataNode caído se recupere, se replicarán los datos para cumplir con el factor de replicación.

En referencia al primer apartado de la prueba se han hecho varias pruebas de manera repetida y consecutiva y en ninguna de ellas se han perdido datos:

1. Apagado de un DataNode.
2. Apagado de un segundo DataNode.
3. Apagado del tercer DataNode y encendido de cualquiera de los otros dos.

Acerca del segundo y tercer apartado, en ambos casos el comportamiento del sistema ha sido siempre el esperado, sin pérdida de datos y con la replicación automática de cara al usuario. Se han seguido los siguientes pasos:

1. Apagado de un DataNode.
2. Adición de un fichero (de tamaño suficiente para ocupar tres bloques o más).
3. Comprobación que el fichero se ha replicado a los otros dos DataNodes.
4. Encendido del DataNode restante y comprobación de la réplica de los bloques del fichero a este.

9.3. FLUME - RECOLECCIÓN DE DATOS

Las pruebas referentes a Flume se basan en tres ámbitos principalmente: la integridad del sistema a la hora de transferir datos, el rendimiento de Flume en el clúster instalado dependiendo del tamaño de los datos y la tolerancia a fallos.

Para realizar las pruebas se ha usado el código implementado para el caso de uso de las redes sociales pero en lugar de descargar tweets, los clientes de Flume leen un fichero línea a línea y lo transfieren a los agentes.

Nombre	Código	Descripción breve
Pérdida de datos	F1	Se permite pasar un fichero de un origen a un destino, mediante Flume, manteniendo la integridad del fichero.
Rendimiento	F2	Calcular con que velocidad transmite un fichero de un origen a un destino.
Tolerancia a fallos 1	F3	Si un agente Flume se cae, poder configurar el origen para que automáticamente pase a enviar datos a otro agente Flume. Con canales no persistentes.
Tolerancia a fallos 2	F4	Si un agente Flume se cae, poder configurar el origen para que automáticamente pase a enviar datos a otro agente Flume. Con canales persistentes.

Tabla 24: Listado de las pruebas realizadas en el apartado de almacenamiento.

9.3.1. F1 - PÉRDIDA DE DATOS

Esta prueba se ha hecho en paralelo con la prueba F2 de rendimiento, aprovechando que esta transfería ficheros de tres tamaños distintos: 1, 5 y 25 GB. Como se puede observar en la *Tabla 25*, en ninguna de las transferencias de datos se ha perdido información, por lo que se puede afirmar que Flume es una herramienta fiable y estable en un entorno controlado -ya que esta prueba se ha realizado sin haber fallos de conexiones-. Añadir que esta prueba es significativa puesto que entre los ordenadores que ejecutaban los clientes y el clúster con los agentes estaba toda la infraestructura de la red interna de la empresa, es decir que los ordenadores no estaban conectados directamente.

Para hacer la comprobación de pérdida de datos se ha usado una función SHA1 (una función hash) tanto en el origen como en el destino, para asegurar que los datos recibidos son exactamente los mismos que los de origen.

9.3.2. F2 - RENDIMIENTO

En la prueba de rendimiento se han usado tres ficheros de tamaños distintos para comprobar el rendimiento de Flume con distintos tamaños de datos -de 1, 5 y 25 GB-. Como se puede observar en la *Tabla 25* y en el *Gráfico 2* los tiempos de ejecución son linealmente proporcionales al tamaño del fichero transferido. La explicación a este resultado es que, tal y como pasaba en la prueba HD2, cuanto mayor es el fichero a añadir en HDFS, mayor es el tiempo que se tarda en replicar sus múltiples bloques y por lo tanto el tiempo crece proporcionalmente.

Tamaño del fichero	SHA1 origen	Tiempo (minutos)	Media (minutos)	SHA1 destino
1 GB	34c2fabc3917bd6e524ab2a1e c55903d45753853	65	64	34c2fabc3917bd6e524ab2 a1ec55903d45753853
		71		34c2fabc3917bd6e524ab2 a1ec55903d45753853
		56		34c2fabc3917bd6e524ab2 a1ec55903d45753853
		64		34c2fabc3917bd6e524ab2 a1ec55903d45753853
5 GB	6964de8f020ea58ce75fead2b f9e490422b0c321	308	306	6964de8f020ea58ce75fea d2bf9e490422b0c321
		299		6964de8f020ea58ce75fea d2bf9e490422b0c321
		315		6964de8f020ea58ce75fea d2bf9e490422b0c321
		304		6964de8f020ea58ce75fea d2bf9e490422b0c321
25 GB	c4d89c66db39d21b2be96af4f b8871558295803e	1.481	1.449	c4d89c66db39d21b2be96 af4fb8871558295803e
		1.435		c4d89c66db39d21b2be96 af4fb8871558295803e
		1.365		c4d89c66db39d21b2be96 af4fb8871558295803e
		1.518		c4d89c66db39d21b2be96 af4fb8871558295803e

Tabla 25: tiempos de ejecución de las transferencias de ficheros mediante Flume.

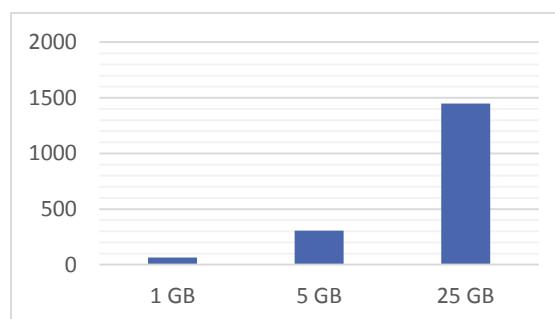


Gráfico 2: Comparación de los tiempos según el tamaño del fichero.

Aprovechando que se disponía de dos ordenadores para realizar el trabajo de cliente y enviar datos a Flume, se realizó la prueba de enviar el mismo fichero desde los dos ordenadores. El resultado esperado es que Flume almacene un fichero en HDFS con los datos duplicados y el doble de tamaño que el fichero original. En la Tabla 26 se pueden observar los resultados: el tiempo para el agente de Flume para recibir dos ficheros de manera paralela es un poco mayor al de recibir uno sólo -un 21% aproximadamente-. De esta prueba también se saca la conclusión que el tiempo de transferencia del fichero -es decir, enviarlo por la red y el proceso que realiza el cliente de Flume- es mayor al de adición a HDFS.

Tamaño del fichero	Tiempo (minutos)	Media (minutos)	Tamaño destino
1 GB	77	78	2 GB
	74		2 GB
	81		2 GB
	79		2 GB

Tabla 26: Tiempo de ejecución de enviar un mismo fichero desde dos clientes a un mismo agente Flume y almacenar los datos en HDFS.

9.3.3. F3 - TOLERANCIA A FALLOS 1

Para comprobar la tolerancia a fallos de Flume se ha diseñado un sistema con dos agentes ejecutándose en dos nodos distintos del clúster. De esta manera, el cliente que envía los datos tiene una lista de los agentes a los que puede enviar datos -como en la Figura 24-. Así el segundo agente funciona a modo de back-up, por si el primero falla. Durante la transferencia de un fichero de 1 GB se apaga el nodo del agente activo para comprobar tres puntos:

- Que el cliente detecta la caída del agente y envía al segundo de manera automática.
- Cuánto tarda en realizar este cambio.
- Cuántos datos se pierden.

Tamaño del fichero	Peso fichero original (kB)	Tiempo (minutos)	Media (minutos)	Peso ficheros destinos (kB)
1 GB	1.037.384	66	67	1.037.045
		68		1.036.684
		65		1.036.914
		70		1.037.018

Tabla 27: Tiempos de ejecución de la prueba F3.

El cliente detecta automáticamente la caída del agente y conecta automáticamente con el otro agente para seguir con la transferencia. Este proceso tarda -comparando los tiempos con los de F2- entre tres y cuatro minutos y además se pierden datos. Esto último es porque los agentes están configurados con un channel en memoria, por lo que todos los eventos que reciben se almacenan en memoria y, al apagar el nodo, se pierden. También cabe indicar que cada agente escribe en un fichero distinto por lo que el resultado de esta prueba son dos ficheros.

9.3.4. F4 - TOLERANCIA A FALLOS 2

Para intentar evitar la pérdida de datos de la prueba anterior, se ha vuelto a ejecutar pero cambiando la configuración de los agentes. Ahora el channel es de tipo file, por lo que almacena los eventos recibidos en un fichero no volátil que perdura aunque se apague el nodo. En la Tabla 28 observamos que el tiempo de ejecución no varía -el channel no tiene influencia- pero que ahora, en lugar de perder datos se duplican, ya que el tamaño de los ficheros resultantes es mayor al del original.

Pruebas

Tamaño del fichero	Peso fichero original (kB)	Tiempo (minutos)	Media (minutos)	Peso ficheros destinos (kB)
1 GB	1.037.384	67	67	1.038.147
		66		1.037.971
		66		1.037.682
		69		1.037.835

Tabla 28: resultados de la prueba F4.

9.4. MAPREDUCE - PROCESOS

Las pruebas sobre MapReduce -versión 1.0- tienen como objetivo evaluar su rendimiento y escalabilidad -entorno al Mapper y al Reducer- además de comprobar que nivel de tolerancia a fallos tiene.

Nombre	Código	Descripción breve
Rendimiento y escalabilidad del Mapper	MR1	Evaluar el rendimiento de MapReduce y su escalabilidad añadiendo más nodos, es decir, permitiendo la ejecución de más Mappers en paralelo.
Rendimiento y escalabilidad del Reducer	MR2	Evaluar el rendimiento de MapReduce en torno a la adición de más procesos Reducer.
Tolerancia a fallos	MR3	Si un nodo se cae durante un trabajo MapReduce, el trabajo continúa y termina correctamente.

Tabla 29: Listado de las pruebas realizadas en el apartado de procesamiento de datos con MRv1.

Para la realización de las pruebas MR1 y MR2 se ha utilizado el algoritmo de Mahout usado en el caso de uso de las redes sociales: Parallel Frequent Pattern Mining o FPM. Este proceso realiza tres trabajos MapReduce de distinta naturaleza en el siguiente orden: [48]

1. **Parallel Counting:** cuenta la aparición de cada palabra en el texto. El trabajo se realiza mayormente en el Mapper, que solamente cuenta las apariciones de las palabras. Por otro lado el Reducer simplemente junta los resultados.
2. **FPM Growth:** en este caso el trabajo realizado es mucho mayor, tanto en el Mapper como en el Reducer:
 - a. **Mapper:** agrupa cada palabra en “transacciones” formando parejas de clave-valor, donde el valor es una lista de palabras dependientes (es decir, palabras que suelen aparecer juntas).
 - b. **Reducer:** agrupa todos los valores con una clave correspondiente,
3. **Aggregator:** simplemente junta los resultados obtenidos del paso anterior. Su carga de trabajo es poca y prácticamente nula en el Reducer.

Estos tres trabajos de tan distintos permiten realizar pruebas y comprobaciones interesantes como se verá más adelante.

La prueba MR3 se ha realizado mediante la ejecución de un algoritmo wordcount ya implementado en la API de MapReduce, puesto que es un trabajo simple y fácil de usar para comprobar la tolerancia a fallos de MRv1.

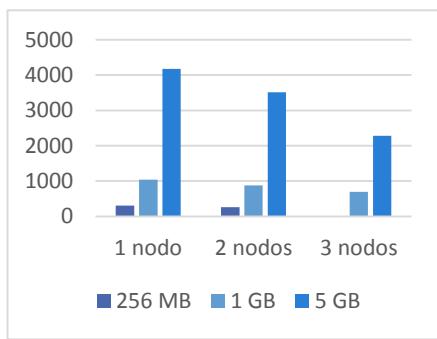
9.4.1. MR1 - RENDIMIENTO Y ESCALABILIDAD DEL MAPPER

En esta prueba se evalúa cuál es el rendimiento de MRv1 a medida que el tamaño de fichero va aumentando y, de paso, cuál es la mejora de rendimiento en términos de escalabilidad al añadir nodos. Esto permitirá al clúster ejecutar más Mappers en paralelo por lo que se podría decir que es una escalabilidad del Mapper. Las ejecuciones mostradas en la Tabla 30 son con un solo Reducer y los tiempos corresponden a la media de cuatro ejecuciones.

Pruebas

Tamaño del fichero (MB)	Número de nodos	Parallel Counting	PFP Growth	Aggregator	Tiempo total (s)
128	3	-	-	-	-
	2	-	-	-	-
	1	56	119	33	229,075
256	3	-	-	-	-
	2	56	143	35	260,487
	1	81	173	36	310,266
384	3	59	254	45	378,737
	2	85	312	47	461,111
	1	113	366	47	549,229
1024	3	121	597	58	692,407
	2	198	663	54	873,968
	1	257	710	54	1.040,983
5.120	3	428	1.753	77	2.280,606
	2	831	2.347	78	3.509,214
	1	1.253	2.850	78	4.179,817

Tabla 30: Tiempos de ejecución del algoritmo de Mahout FPM para ficheros de distinto tamaño.



Como se puede observar en la *Tabla 30*, el aumento de tamaño del fichero de entrada en un clúster con un nodo -es decir, una solución secuencial-, afecta al rendimiento bastante más que en el caso de un clúster con tres nodos de trabajo.

En el *Gráfico 3* se observa perfectamente la ganancia en rendimiento obtenida con la adición de más nodos de ejecución. Se observa además que el speed-up es significativamente mayor a medida que el tamaño de entrada también lo es:

$$\text{speedup } 384 \text{ MB} = \frac{549}{378} = 1,45$$

$$\text{speedup } 1 \text{ GB} = \frac{1.040}{692} = 1,50$$

$$\text{speedup } 5 \text{ GB} = \frac{4.179}{2.280} = 1,83$$

Pruebas

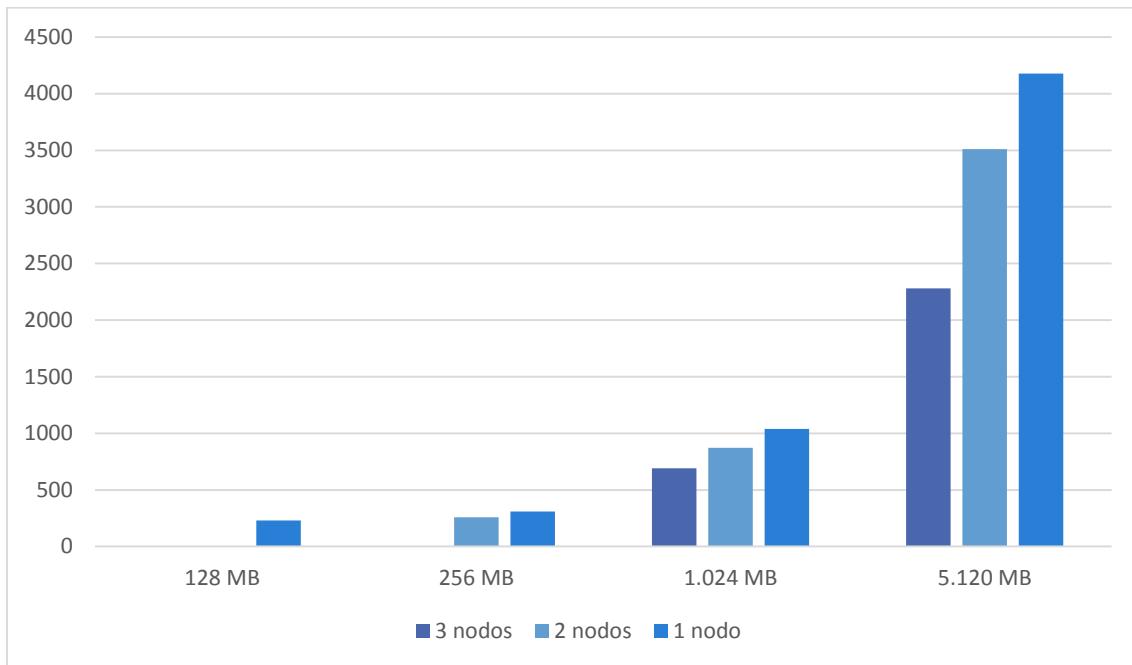


Gráfico 3: Rendimiento de MapReduce según tamaño de entrada y número de nodos.

9.4.2. MR2 - RENDIMIENTO Y ESCALABILIDAD DEL REDUCER

En esta prueba se intenta evaluar la escalabilidad de MapReduce entorno al número de Reducers que se ejecutan en paralelo. Para esto se ha tomado como base la ejecución con tres nodos y un fichero de entrada de 5 GB.

Tamaño del fichero (MB)	Ejecución	Parallel Counting	PFP Growth	Aggregator	Tiempo total (s)
5.120	1	440	960	49	1.481,26
	2	417	964	51	1.461,56
	3	418	985	47	1.467,89
	4	406	1.013	52	1.499,98

Tabla 31: Tiempos de ejecución del algoritmo de Mahout FPM para un fichero de 5 GB y con MapReduce configurado para ejecutar 3 reducers.

Como se puede observar en la Tabla 31, la escalabilidad de MapReduce respecto a los Reducers parece alta. El algoritmo FPM de Mahout es perfecto para poder ver un aspecto: depende del algoritmo que se esté usando. La mayoría de la diferencia de tiempo respecto a las ejecuciones con un solo Reducer viene dada por el segundo MapReduce, donde el trabajo realizado por el Reducer es mucho mayor. Sin embargo de los otros MapReduce también se puede extraer conclusiones interesantes:

El primero, a priori, no se ve afectado porque no tiene un trabajo significativo en el Reducer. El tercero, que aún tiene menos trabajo para el Reducer, debería tener un tiempo igual a las demás ejecuciones pero no es así; tiene un tiempo inferior -pasa de los prácticamente 80 segundos a los 50-. La explicación para este caso es que la entrada de este MapReduce es la salida del anterior, que ejecutaba tres Reducers y por lo tanto generaba tres ficheros. De esta manera lo que en verdad se está reduciendo es el tiempo de ejecución de la etapa Map del trabajo, ya que se consigue paralelizar esta parte -el fichero generado del segundo MapReduce no debería sobrepasar el tamaño de bloque y por eso lo realizaba en un solo Mapper en lugar de los tres de ahora-.

Pruebas

Si pasamos a comparar la ejecución con un solo nodo de un fichero de 5 GB con una ejecución con la misma entrada pero tres nodos -y tres Reducers-, se puede observar la escalabilidad y el rendimiento que se gana en términos de speed-up:

$$\text{speedup} = \frac{4.179}{1.474} = 2,83$$

9.4.3. MR3 - TOLERANCIA A FALLOS

Se han realizado tres tipos de ejecuciones de un proceso wordcount sobre un fichero de 5 GB -f5120-:

- Ejecución normal: para poder obtener una base con la que comparar, tanto en resultados como en tiempos de ejecución.
- ejecución con caída de un nodo durante la etapa Mapper: se busca comprobar que el trabajo continúa de manera normal sin el nodo restante.
- Ejecución con caída de un nodo durante el la etapa Reducer.

Ejecución	Definición	Tiempo (s)	Tiempo medio (s)	Resultado
wc de f5120	Ejecución correcta con 3 nodos	307,304	302,82	21810769
		296,488		21810769
		303,396		21810769
		304,113		21810769
wc de f5120	Se tira un nodo a mitad de un proceso map	872,993	929,15	21810769
		966,251		21810769
		932,153		21810769
		945,214		21810769
wc de f5120	Se tira un nodo a mitad de un proceso reduce	989,139	981,32	21810769
		986,587		ERROR/KILLED
		965,556		ERROR
		984		21810769

Tabla 32: Resultados y tiempos de ejecución de la prueba MR3.

De la *Tabla 32* se pueden sacar dos conclusiones:

- MRv1 es totalmente tolerante a caídas de un nodo durante la ejecución de un Mapper, pues el resultado obtenido es correcto.
- Sin embargo, no es tolerante a fallos durante un Reducer. La diferencia entre las ejecuciones que terminan de manera correcta y las que no en este caso, se debe a dos razones:

Pruebas

- Si el nodo que se cae no es el que está haciendo el Reducer, simplemente se vuelven a ejecutar todos los Mappers de ese nodo y después se pasan los resultados al Reducer.
- Una de las variables de configuración de MRv1 -**mapred.task.timeout**- configura un temporizador que al saltar borra los ficheros temporales con los resultados de los Mappers y los que guardan el estado del trabajo.

La variable de configuración comentada anteriormente también es la responsable del aumento de tiempo entre las ejecuciones -con un error duran del orden de 600 segundos más-. Cuando un nodo falla -y no se está ejecutando el Reducer-, el sistema tarda aproximadamente 400 segundos en detectarlo y una vez descubierto el fallo, relanza todos los Mappers hechos por ese nodo para asegurarse de la consistencia de datos.

Adicionalmente, se ha intentado solucionar la poca tolerancia a errores en casos de caídas del Reducer aumentando el número de Reducers por trabajos. Esta solución no ha dado resultado ya que el fichero que guarda el estado del trabajo MapReduce -y por lo tanto el que usa el TaskTracker para gestionar los workers- tiene un factor de replicación 1. Es decir, que al tirar el Reducer puede coincidir con que almacene este fichero y por lo tanto el TaskTracker no pueda realizar el seguimiento del trabajo. La solución a este problema pasa por cambiar una variable de configuración: **mapred.submit.replication**, que precisamente se encarga de configurar el factor de replicación de los ficheros temporales del trabajo MapReduce.

9.5. YARN - PROCESOS

Las pruebas realizadas sobre MRv2 van encaradas sobre todo al rendimiento, puesto que gracias a YARN el sistema ahora utiliza los recursos disponibles de manera más inteligente. Al final se hace una comparación entre MRv1 y MRv2 en el apartado de rendimiento.

Nombre	Código	Descripción breve
Uso de los recursos	Y1	Evaluando la gestión de recursos que realiza YARN.
Escalabilidad	Y2	Evaluando la escalabilidad de YARN.
Rendimiento	Y3	Modificar varios parámetros de configuración de YARN para intentar afinar y optimizar el rendimiento del proceso map-reduce.
Comparación con MRv1	Y4	Comparar el rendimiento de las dos herramientas de Cloudera CDH4 que implementan el paradigma map-reduce: MapReduce y YARN.

Tabla 33: Listado de las pruebas realizadas en el apartado de procesamiento de datos con YARN o MRv2.

9.5.1. Y1 - USO DE LOS RECURSOS

La principal novedad de MRv2 es YARN, un sistema de gestión de recursos para aplicaciones distribuidas; por lo que un punto interesante a comentar es las observaciones que se han realizado durante la ejecución de los casos de uso y de las pruebas comentadas en adelante.

Al trabajar con Containers, YARN no hace distinciones de recursos entre los distintos nodos que componen el clúster. De esta manera, es probable que si uno de los nodos tiene menos recursos que los demás, intente asignarle una parte del trabajo que no pueda realizar -en memoria, por ejemplo- y termine siendo improductivo -en el caso de la memoria, realizando swapping o incluso parando el proceso Java por falta de memoria-. Esto sería consecuencia de configurar un Container con demasiados recursos, de manera que algunos nodos -o todos- no puedan cumplir con los requisitos.

Además, la configuración del Container también servirá para determinar cuántos Mappers y Reducers se lanzan en este. A diferencia de con MRv1 donde prácticamente se configuraba mediante una variable, ahora YARN lo calcula mediante los recursos del sistema disponibles y la configuración de Container que el usuario haya realizado.

Por estas razones es aconsejable tener un clúster homogéneo -al menos en los nodos que vayan a realizar tareas MapReduce- y, sobretodo, teniendo en cuenta los servicios que alberga cada nodo - servicios de Hive, Impala, HDFS, etc.-.

9.5.2. Y2 - ESCALABILIDAD

Las pruebas realizadas en este punto siguen la misma estructura que las de MR1, es decir que son de escalabilidad del Mapper. Se ha añadido la ejecución con un fichero de 25 GB ya que a la hora de realizar esta prueba se tenían los datos suficientes como para generarlo.

Pruebas

Tamaño del fichero (MB)	Número de nodos	Parallel Counting	PFP Growth	Aggregator	Tiempo total (s)
128	3	-	-	-	-
	2	-	-	-	-
	1	57	90	25	207,105
256	3	-	-	-	-
	2	65	132	29	289,465
	1	65	131	29	258,438
384	3	92	272	47	449,102
	2	108	285	42	446,32
	1	90	262	39	425,859
1024	3	120	440	55	656,052
	2	123	496	53	676,627
	1	186	497	47	761,580
5.120	3	384	1.652	73	2.152,355
	2	631	1.781	75	2.456,491
	1	972	1.935	68	2.997,01
25.600	3	1.724	2.165	64	4.047,204
	2	2.489	3.447	63	5.444,74
	1	4.915	4.895	61	9.944,93

Tabla 34: Resultados de la prueba Y2.

Los resultados mostrados en la *Tabla 34* son bastante auto explicativos. YARN parece que no escala con ficheros pequeños -menores de 1 GB-, donde hasta a veces con un solo nodo se ejecuta más rápido - caso del fichero de 384 MB-. Esto puede ser debido a que ahora YARN usa contenedores para gestionar los nodos que realizan trabajos MapReduce y el coste de crear y mantener estas estructuras no debe compensar con ficheros pequeños. Sin embargo, con ficheros grandes, se nota de manera notable la mejora de rendimiento al tener más nodos:

$$\text{speedup } 1 \text{ GB} = \frac{761}{656} = 1,16$$

$$\text{speedup } 5 \text{ GB} = \frac{2.997}{2.152} = 1,39$$

$$\text{speedup } 25 \text{ GB} = \frac{9.944}{4.047} = 2,45$$

En el *Gráfico 4* se ve patente este hecho, donde para los ficheros pequeños los tiempos están igualados y, por otro lado, para los ficheros grandes las diferencias entre las ejecuciones con distintos números de nodos se van agrandando a medida que la entrada es mayor.

Pruebas

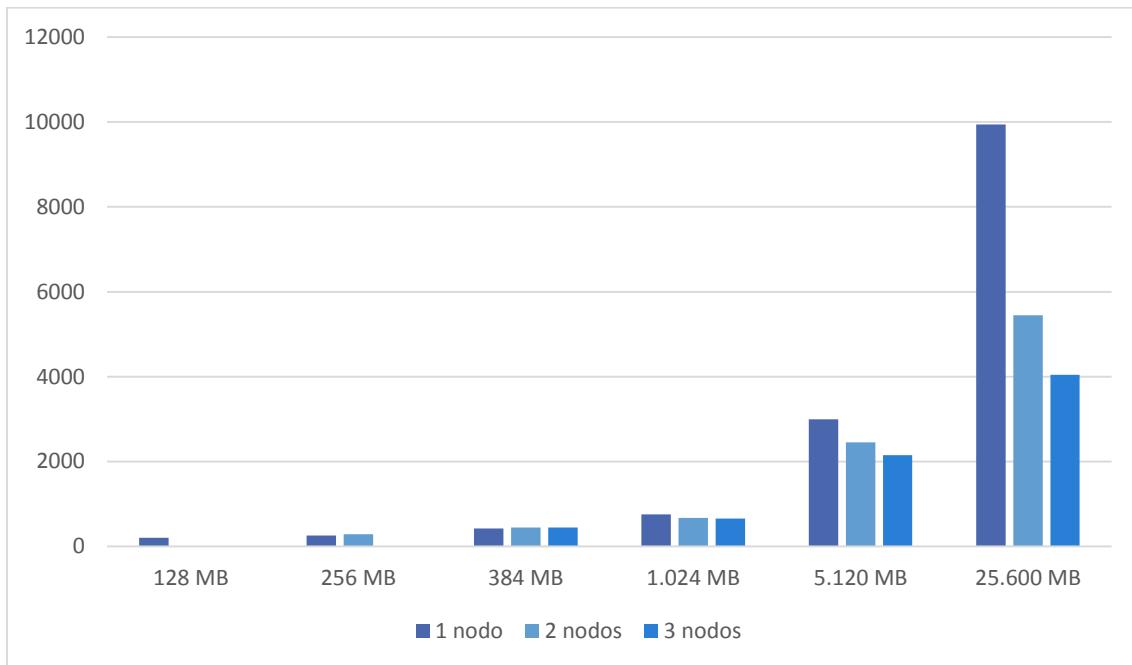


Gráfico 4: Comparación de la mejora en rendimiento al añadir más nodos en un clúster para cada tamaño de entrada.

9.5.3. Y3 - RENDIMIENTO

En este punto de los pruebas se intenta hacer un poco de tuning -es decir, cambiar la configuración de YARN a gusto- para obtener un mayor rendimiento. Una de las ventajas de YARN es que permite realizar múltiples cambios en la configuración de los contenedores y de los recursos usados por cada trabajo, Mapper y Reducer. En la *Tabla 35* se muestran los parámetros modificados.

Nombre	Propiedad	Valor por defecto	Descripción
Número de reducers	mapreduce.job.reduces	1	Número de Reducers que se ejecutarán en una tarea MapReduce.
Memoria dedicada a contenedores	yarn.nodemanager.resource.memory-mb	4 GB	Memoria dedicada a los contenedores.
Memoria para tareas Map/Reduce	mapreduce.map/reduce.memory.mb	1 GB	La cantidad máxima que un Mapper y un Reducer pueden obtener.

Tabla 35: Parámetros modificados en la prueba Y3.

Las ejecuciones se han realizado con el algoritmo FPM de Mahout sobre el fichero de 5 GB con la misma configuración -salvo los parámetros modificados- que en la prueba Y2 con tres nodos. De manera que los tiempos tomados como referencia son los obtenidos en esa prueba.

En la *Tabla 36* están los resultados de diversas ejecuciones -solo se muestra la media- de modificar el número de Reducers entre los valores 1, 5, 10 y 15. También se muestra el tiempo total de ejecutar Reducers en el segundo trabajo del algoritmo ya que es el que tiene una carga de trabajo elevado y donde mejor se puede apreciar la diferencia.

Pruebas

# Reducers	Total	Parallel Counting	PFP Growth	Reducer Growth	Aggregator
Tiempos					
1	2.063	381	1.552	1.311	76
5	1.312	376	793	558	67
10	1.188	397	658	409	67
15	1.194	407	653	408	83

Tabla 36: Resultados de modificar el número de Reducers.

El tiempo de ejecución mejora drásticamente al añadir Reducers, sobre todo por el lado de la etapa reduce del segundo trabajo MapReduce -PFP Growth-. Añadir más Reducers no implica seguir mejorando y, de hecho, la mejora se estanca alrededor de los diez Reducers, por lo que ese es el tiempo que servirá como base. Este hecho se aprecia en el *Gráfico 5*.

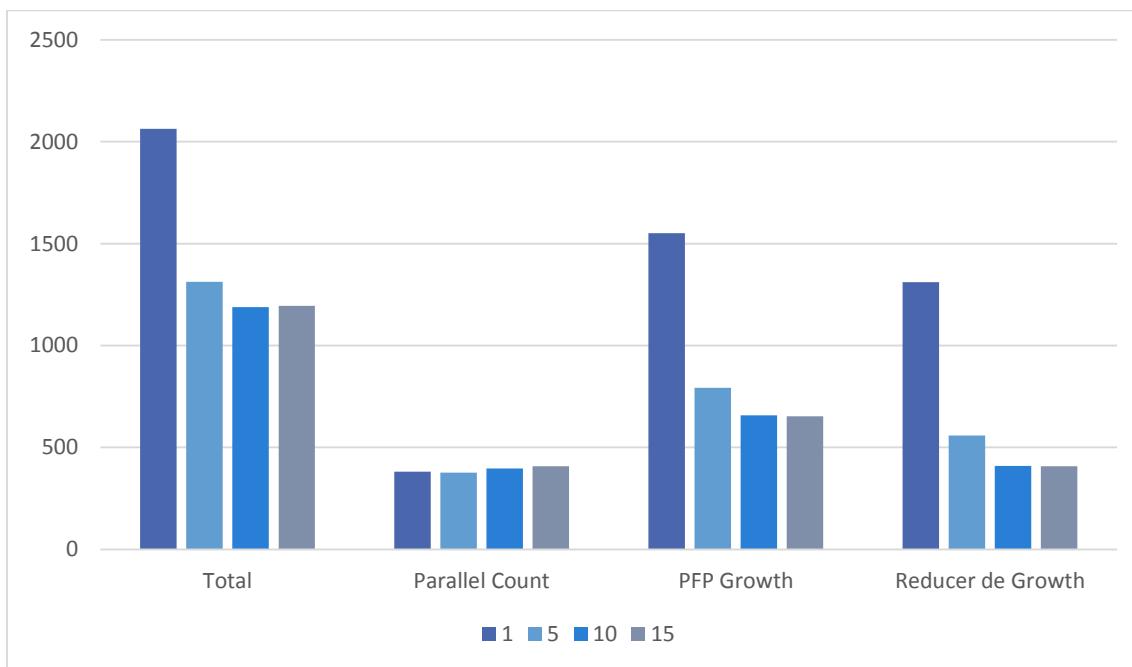


Gráfico 5: Comparativa de tiempos de ejecución entre los diferentes pasos del algoritmo.

La siguiente configuración cambiada es la de memoria dedicada a los contenedores Java para ejecutar tareas MapReduce. En este caso se ha ido disminuyendo -puesto que los nodos tienen 4 GB de memoria, el valor por defecto-. La disminución de memoria también conlleva una disminución en el número de Reducers y, además, por lo comentado en Y1, puede darse el caso de que por intentar hacer demasiados Reducers un nodo sobrepase sus límites. En la *Tabla 37* se observan los tiempos y como el caso ideal es el caso por defecto.

Memoria contenedores	Total	Parallel Counting	FPF Growth	Reducer Growth	Aggregator
Tiempos					
4 GB (10r)	1.188	397	658	409	67
3 GB (10r)	1.263	452	706	434	68
3 GB (7r)	1.201	424	670	397	57
2 GB (7r)	1.578	568	889	540	70
2 GB (4r)	1.493	565	830	494	51

Tabla 37: Resultados de modificar la memoria del contenedor.

Finalmente el último punto de la configuración a modificar es la memoria dedicada a cada Mapper y Reducer donde, como se ve en la *Tabla 38*, no afecta demasiado a los resultados en cuanto a tiempo de ejecución.

Memoria Mapper/Reducer	Total	Parallel Counting	FPF Growth	Reducer Growth	Aggregator
Tiempos					
2 GB (5 slots)	1.216	439	683	419	47
1.5 GB (5 slots)	1.181	403	645	405	68
1 GB (10 slots)	1.188	397	658	409	67

Tabla 38: Resultados de modificar la memoria de los Mapper y Reducers.

De todas maneras esta última prueba sirve para confirmar como se gestionan los Mappers y Reducers a través de los contenedores. En el caso de los Reducers, donde se ve más claro, se tiene la siguiente disposición:

Se cuenta con tres nodos, cada uno con 4 GB de memoria, y cada uno con los siguientes procesos ocupando memoria dedicada a los contenedores -un total 4 GB:-

- uno de los nodos cuenta con el ApplicationMaster, que ocupa 2 GB. Esto significa que a este nodo solamente le quedan disponibles otros 2 GB. En el caso de tener Reducers de 1 GB podrá ejecutar dos de manera paralela pero, si son de 1,5 o 2 GB, solamente podrá con uno a la vez (en el caso de 1,5 GB, este nodo además tiene en desuso 0,5).
- Los otros dos nodos cuentan cada uno con los 4 GB de memoria libres por lo que en el caso de tener Reducers de 1 GB podrán ejecutar cuatro cada uno y, en el caso de ser de 1,5 o 2 GB, podrán con dos cada nodo (esta vez, en el caso de 1,5 GB, se dejará de aprovechar 1 GB).

De esta manera, con Reducers de 1 GB podrán ejecutarse un total de 10 de forma paralela (2 en el nodo master y 4 en los otros) y en los otros dos casos solamente 5 (1 en el master y 2 en los otros). Esta explicación sirve también para los Mappers y es por esta razón que en la Tabla 38 se indica el número de slots en cada caso (es decir, el número de Mappers y Reducers que se pueden ejecutar en paralelo).

Pruebas

Para terminar con esta prueba, a continuación se muestran los speed-ups de la mejor configuración encontrada (contenedores de 4 GB, 1.5 GB para cada Mapper y Reducer y un máximo de 5 Reducers):

$$\text{speedup respecto a un nodo} = \frac{2.997}{1.181} = 2,53$$

$$\text{speedup respecto a tres nodos} = \frac{2.152}{1.181} = 1,82$$

9.5.4. Y4 - COMPARACIÓN CON MRV1

La última prueba es una comparativa de rendimiento entre MRv1 y MRv2:

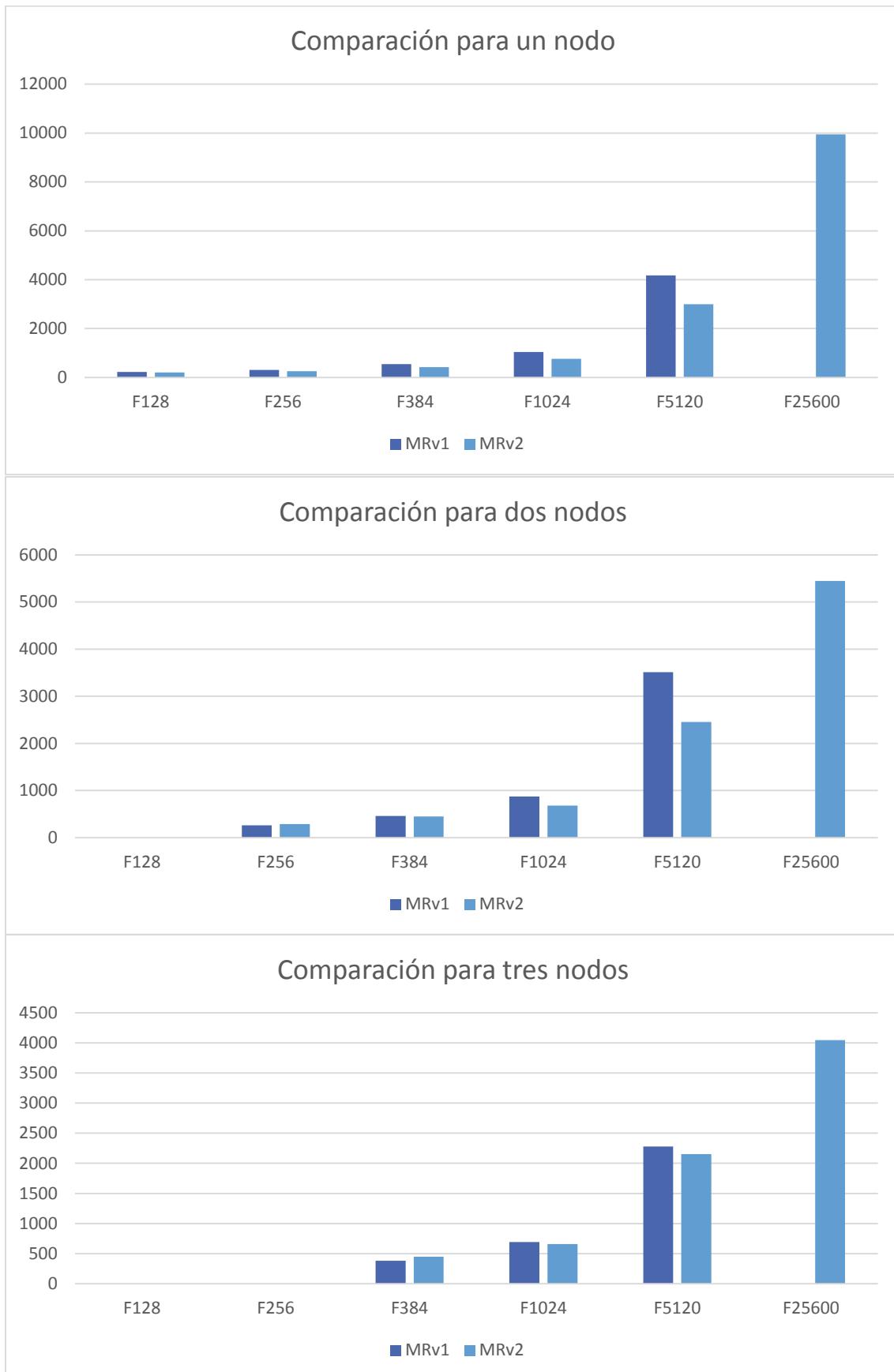
Nodos	Fichero	Tiempo MapReduce (s)	Tiempo YARN (S)
3	F128	-	-
	F256	-	-
	F384	378,737	449,102
	F1024	692,407	656,052
	F5120	2.280,606	2.152,355
	F25600	-	4.047,204
2	F128	-	-
	F256	260,487	289,465
	F384	461,111	446,320
	F1024	873,968	676,627
	F5120	3.509,214	2.456,491
	F25600	-	5.444,740
1	F128	229,075	207,105
	F256	310,266	258,438
	F384	549,229	425,859
	F1024	1.040,983	761,580
	F5120	4.179,817	2.997,012
	F25600	-	9.944,930

Tabla 39: Comparación entre MapReduce v1 y MapReduce v2 o YARN.

Los resultados mostrados en la *Tabla 39* son los tiempos de ejecución sin ningún tipo de tunning (ya que en MRv1 no se ha realizado, sería una comparación injusta). Lo que se observa es lo mismo que en la prueba Y1, YARN no trabaja tan bien como MRv1 con ficheros más pequeños de 1 GB pero a partir de este tamaño, toma la delantera. Por otro lado, MRv1 obtiene un speed-up mayor al añadir nodos pero esto es porque gracias a la gestión de recurso, YARN funciona mejor con clústeres pequeños y por lo tanto el rango de mejora es menor.

En conclusión, además de ofrecer una gestión de recursos más segura a nivel de sistema, YARN también ofrece un rendimiento superior con ficheros de mayor tamaño -que es el objetivo que se busca con Big Data- y una mayor flexibilidad para configurar los nodos y tunear el sistema.

Pruebas



10. CONCLUSIONES

Parece claro que Big Data es el futuro de las tecnologías de la información y la comunicación. No solamente porque se adapta mejor a los cambios y evolución que está sufriendo la sociedad -tanto en tecnología como en necesidades- sino también por la gran aceptación que tiene entre las empresas. Esta aceptación es una realidad a dos niveles: a nivel de desarrollo, donde cada vez más compañías desarrollan nuevas tecnologías y apuestan más por las existentes -como Hadoop-; y a nivel corporativo, donde las empresas empiezan a ver con más interés las nuevas posibilidades que aportan las soluciones Big Data, con el resultado del incremento de proyectos Big Data.

10.1. HADOOP

Hadoop es la solución que más éxito y repercusión está teniendo. No sólo por el hecho de que todas las compañías que ofrecen distribuciones Big data lo hacen con Hadoop sino porque cuenta con un ecosistema muy completo y que no para de crecer. Además de ser un proyecto open source, que también es hacia donde se dirige la comunidad desarrolladora, y de ser una solución hecha desde cero y pensada exclusivamente para Big Data.

En este sentido aún queda mucho camino que recorrer ya que hay un baile constante de herramienta y versiones que hace que el sistema sea algo inestable por dos motivos principalmente:

- La aparición de nuevas versiones muchas veces crea incompatibilidades con las versiones anteriores puesto que muchos desarrollos están en una fase temprana para llegar a su objetivo.
- Tanto la aparición de nuevas herramientas como de nuevas versiones es un quebradero de cabeza para los desarrolladores ya que en ambos casos tienen que realizar un estudio técnico, debido a los altos cambios.

A pesar de eso, las versiones estables de Hadoop están totalmente preparadas para estar en aplicaciones de producción -y los cientos de aplicaciones en funcionamiento así lo atestiguan-. Además, a pesar del impacto inicial y a su curva de aprendizaje elevada, una vez se dominan los paradigmas y las herramientas, el desarrollador puede tener una productividad tan elevada como en soluciones tradicionales pero con las posibilidades que aporta Hadoop.

Finalmente, en el desarrollo de los casos de uso y en los resultados de las pruebas, Hadoop ha resultado ser una solución que se adapta y encaja muy bien con Big Data, mostrando ser un sistema escalable, con tolerancia a errores y a diferentes cambios en la arquitectura -tanto a nivel de diseño o funcionalidades como a nivel de infraestructura-, con un buen rendimiento en el trato de datos no estructurados y con una alta flexibilidad para añadir nuevas herramientas a una solución o incluso acoplarse con ya existentes para tener un efecto menor sobre las infraestructuras ya montadas.

10.2. EVERIS

Trabajar en Everis ha sido una experiencia muy satisfactoria tanto a nivel personal como en el académico y laboral. En todo momento se ha tenido la ayuda necesaria -no solamente por parte de los directores- para llevar a cabo el proyecto y para aprender conocimientos de trabajo en una gran empresa. Everis se preocupa de formar profesionales, encarar su futuro hacia un trabajo donde se sientan cómodos y valorados y de integrar a sus trabajadores -incluso a los becarios- desde el primer momento. Todos esos valores han sido reflejados durante la realización de este proyecto de manera muy satisfactoria.

ANEXOS

A. GLOSARIO

A

Alta disponibilidad - es una característica del sistema que asegura una continuidad operacional durante un periodo de tiempo determinado.

API - significa interfaz de programación de aplicaciones. Es una interfaz que define la manera de trabajar con distintos componentes de programación.

Arquitectura maestro-esclavo - arquitectura de red de servidores con una jerarquía centralizada donde el nodo maestro lleva el peso del control mientras que los esclavos realizan el trabajo de procesamiento.

B

Base de datos - es un conjunto de datos que pertenecen a un mismo contexto y están almacenados en un mismo sistema con la intención de ser usados repetidas veces.

Base de datos relacional - es una base de datos que cumple con el modelo relacional.

Batch - es una ejecución de una serie de trabajos o procesos informáticos que no requieren de la intervención manual para iniciarse.

Bit - unidad básica de información en la informática y las comunicaciones digitales.

Byte - unidad de información digital formada por una secuencia de ocho bits. A continuación se muestra la escala de unidades en binario:

- **kilobyte** - 1024 bytes
- **megabyte** - 1024² bytes
- **gigabyte** - 1024³ bytes
- **terabyte** - 1024⁴ bytes
- **petabyte** - 1024⁵ bytes
- **exabyte** - 1024⁶ bytes
- **zettabyte** - 1024⁷ bytes
- **yottabyte** - 1024⁸ bytes

C

Cartera de inversiones - conjunto de activos financieros en los cuales se invierte.

CERN - proviene de Conseil Européen pour la Recherche Nucléaire. Es un centro de investigación científico internacional especializado en física de partículas.

Clúster - conjunto de ordenadores interconectados que actúan como si fueran uno solo.

Consola de comandos - también conocida como línea de comandos, es una herramienta que permite a los usuarios de una aplicación darle instrucciones a través de comandos de texto sencillos.

D

Daemon - también demonio, es un proceso que se ejecuta en segundo plano.

Anexos

Dataflow - o flujo de información. El diseño de una ejecución de una serie de procesos o autómatas que se comunican enviándose información a través de distintos canales.

Data warehouse - es una colección de datos orientada a un ámbito empresarial u organizativo y almacenada en un sistema no volátil e integrado que ayuda a la toma de decisiones.

Dashboard - o tablero de instrumentos es una interfaz desde la que el usuario puede controlar y administrar una aplicación.

Distribuido - la computación distribuida es un modelo de computación en la que se usan una serie de ordenadores organizados en clústeres.

Double - tipo básico de número racional en coma flotante de doble precisión.

E

Escalabilidad - es una propiedad de un sistema que indica su capacidad de reacción y de adaptación a los cambios de envergadura, ya sean al crecer o disminuir.

Espacio de nombres - contenedor abstracto en el que se agrupan varios identificadores únicos - organizados de varias maneras, en el caso de un sistema de ficheros convencional de manera jerárquica- que coexisten en un mismo espacio de nombres.

ETL - proceso de transformación de datos realizado para extraer datos de una fuente y almacenarlos en una base de datos o data warehouse.

F

Framework - conjunto de conceptos y tecnologías que sirven de base para el desarrollo de aplicaciones. Acostumbra a incluir bibliotecas de software, lenguajes, soportes a aplicaciones de desarrollo...

H

Hash - función que realiza una transformación -normalmente de reducción- a su entrada, dejando una salida -valor hash- de tamaño finito.

HashMap - es una colección de objetos sin orden que asocia valores a una clave, formando parejas de tipo <clave - lista de valores>, donde cada valor de la lista es procesado por una función hash y da como resultado la clave.

Handshaking - es un protocolo de comunicación que sirve para definir un proceso de comunicación e identificación entre dos entidades a través de un canal.

HTTP - de Hypertext Transfer Protocol. Es un protocolo de comunicación sin estado -no guarda información sobre las conexiones realizadas con anterioridad- para el intercambio de peticiones y documentos. Es el protocolo usado en la World Wide Web.

IDE - significa Integrated Development Environment y consiste en una aplicación que agrupa un conjunto de herramientas -como editor de texto, gestor de proyectos, compilador o debugger- para facilitar el desarrollo de aplicaciones.

Log - es un fichero que archiva un conjunto de entradas que informan de los distintos eventos -cambios en los estados de los procesos, comunicaciones, errores...- que se producen en un ordenador.

M

Modelo relacional - es un modelo de datos basado en la lógica de predicados y en la teoría de conjuntos. Está compuesto fundamentalmente por tablas y se basa en el uso de las relaciones entre estas.

N

NoSQL - amplia clase de sistemas de gestión de bases de datos que ofrecen una alternativa al modelo relacional tradicional en aspectos muy importantes, destacando la no utilización de SQL.

O

Open-source - hace referencia al código distribuido y desarrollado libremente, de manera abierta a todo el mundo, dando acceso al código fuente del proyecto.

Outsourcing - es el proceso empresarial de destinar recursos a contratar una empresa externa para realizar ciertas tareas determinadas.

P

Phishing - ataque informático que consiste en suplantar la identidad de una persona o una entidad para engañar al atacado.

Repositorio - servidor centralizado donde se almacena y mantiene información digital -como bases de datos, aplicaciones o código- para poder ser accedidos remotamente.

R

RFID - sistema de almacenamiento y recuperación de pequeñas cantidades de datos que se utiliza mediante etiquetas RFID y que generalmente se usan para la identificación.

S

Script - archivo de texto plano que contiene una serie simples de comandos -o códigos sencillos en lenguajes interpretados- cuyo objetivo es generalmente el de realizar tareas de orquestación de procesos o monitorización.

Sensor - dispositivo capaz de detectar eventos físicos -cambios de luz, movimiento, temperatura- y convertirlo en información analógica o digital.

Serialización/deserialización de datos - codificación o decodificación de un objeto en un medio con el fin de transmitirlo en forma de series de bytes.

Sistema gestor de bases de datos (SGBD) - conjunto de programas que permiten el almacenamiento, modificación y extracción de la información en una base de datos. También incluyen métodos de administración y monitorización.

Socket - concepto abstracto mediante el que dos aplicaciones pueden conectarse e intercambiar datos. Generalmente se usa una API para protocolos TCP/IP para realizar la comunicación mediante sockets.

Speed-up - en computación en paralelo hace referencia a cuantas veces un algoritmo en paralelo es más rápido que uno en secuencial.

SQL - lenguaje de consultas estructuradas para bases de datos relacionales.

Anexos

Streaming - distribución de datos de manera constante en forma de flujo continuo -sin interrupción-, usada por ejemplo en la transmisión de contenido multimedia a través de internet.

String - tipo básico de lenguajes de programación que representa una cadena de caracteres.

Subconsulta SQL - instrucción de selección anidada dentro de otra consulta SQL.

Swapping - Mecanismo del sistema operativo para relacionar la memoria principal del ordenador con la memoria del disco, de manera que se pueda usar la segunda para ampliar la principal realizando intercambios de información.

T

Tail - proceso Unix que escribe por pantalla el final de un fichero. También permite escribir un fichero en tiempo real a medida que va creciendo -ideal para la visualización de ficheros de logs-.

TCP/IP - conjunto de protocolos de red definidos con el fin de hacer una definición de la transferencia de datos entre computadoras.

TI - tecnologías de la información.

Tiempo real - un sistema en tiempo real es aquel que interactúa de manera dinámica con un entorno generador de datos con el fin de capturar u ofrecer información a medida que se va generando.

Timestamp - secuencia de caracteres que denotan una fecha y hora.

Tupla - estructura de datos que contiene diversos objetos.

U

Unix - sistema operativo desarrollado en 1969 y que forma parte del núcleo de sistemas actuales como Linux o Mac OS X.

URL - es un sistema de localización de recursos uniforme que usa una secuencia de caracteres con un formato modélico y estándar para Internet.

V

Virtualización - emular un entorno físico mediante software dentro de otro entorno físico real.

W

Wordcount - programa que cuenta el número de palabras de texto determinado.

Workflow - flujo de trabajo que define como tienen que ejecutarse y comunicarse entre ellas diversas aplicaciones o ejecuciones.

B. PLANIFICACIÓN

En este apartado se muestran los diagramas de Gantt más detallados.

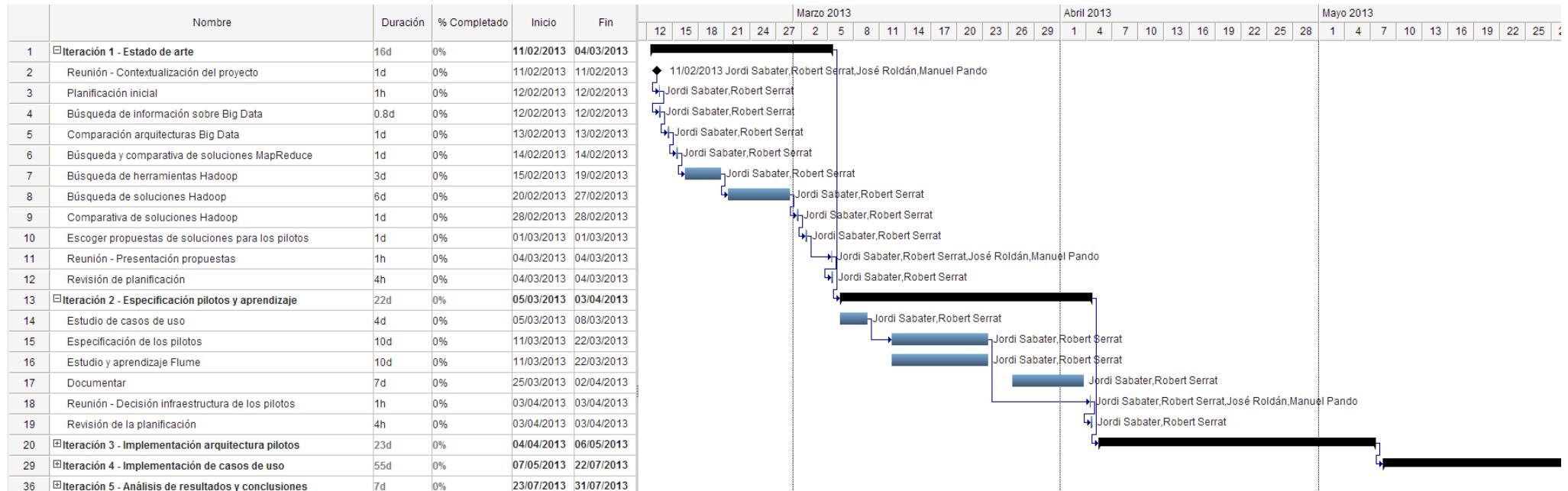


Figura 52: Diagrama de Gantt de las iteraciones 1 y 2 de la planificación inicial.

Anexos

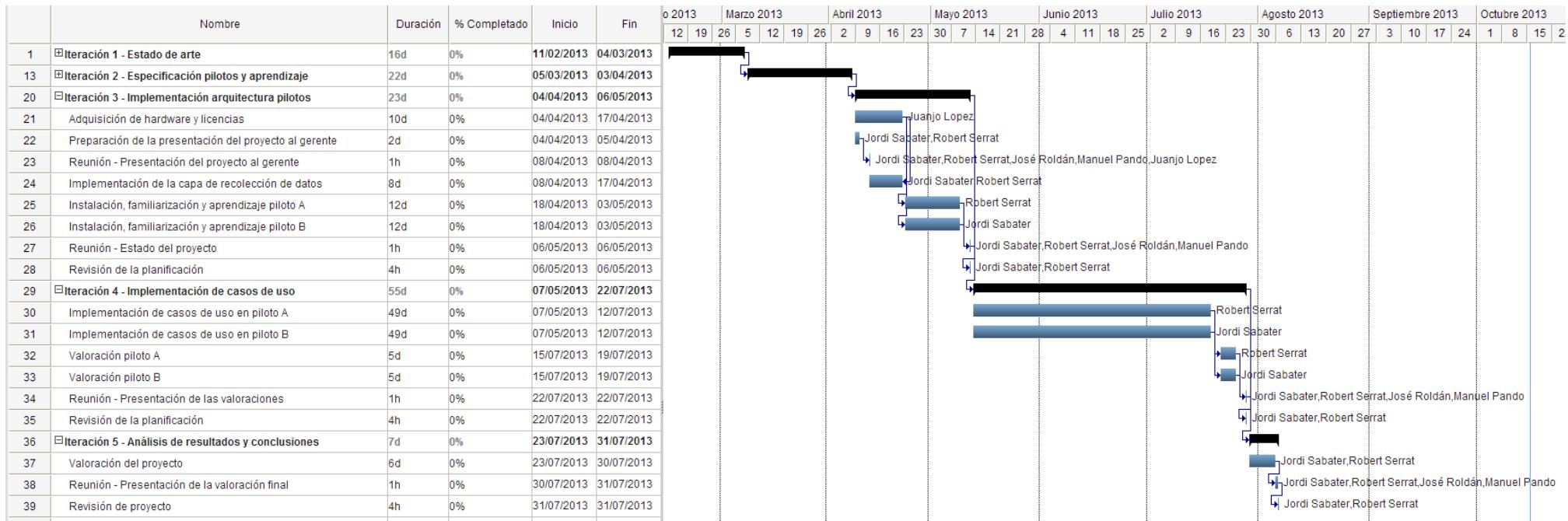


Figura 53: Diagrama de Gantt de las iteraciones 3, 4 y 5 de la planificación inicial.

Anexos

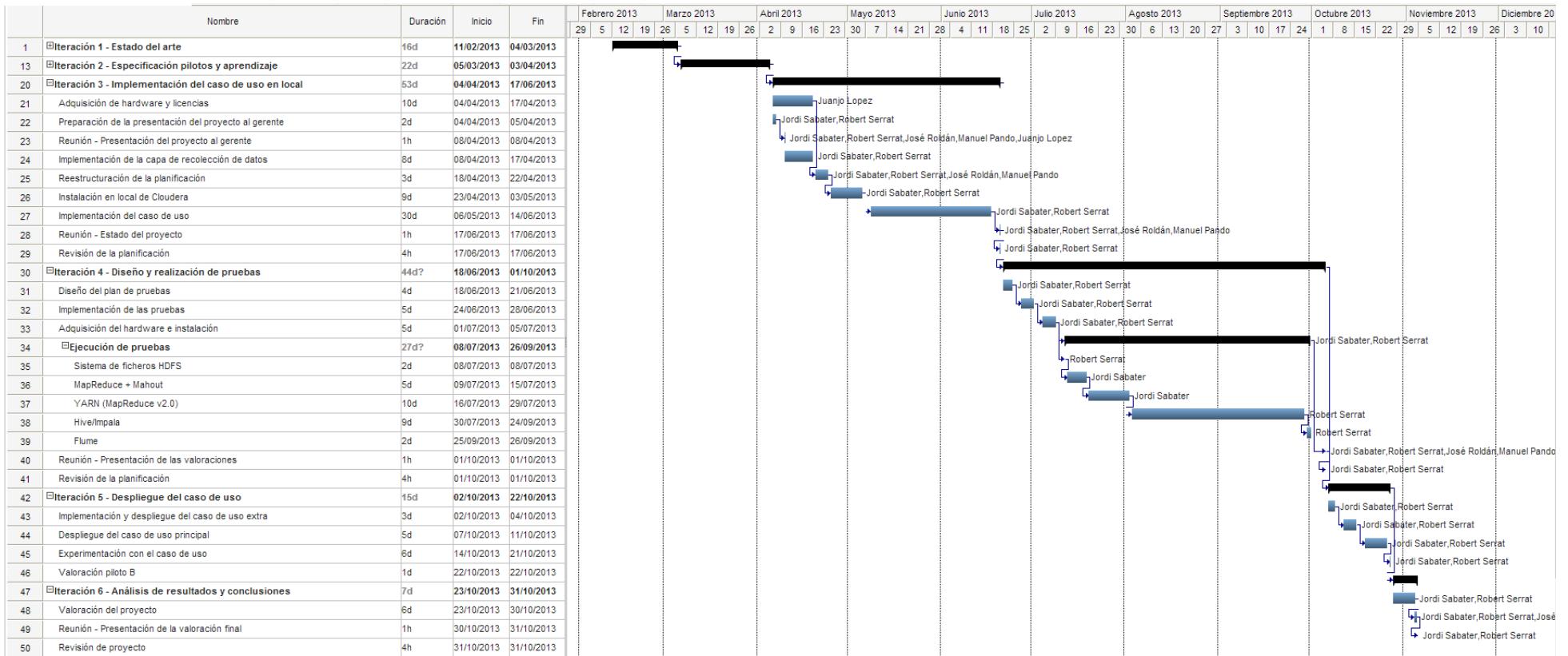


Figura 54: Diagrama de Gantt de las iteraciones 3, 4, 5 y 6 de la planificación final.

C. ARQUITECTURA

En este apartado se describe la arquitectura que se ha usado durante la parte técnica del proyecto, tanto para la realización de las pruebas como para los casos de uso. La infraestructura era básicamente un servidor con la siguiente configuración:

- **Modelo:** HP ProLiant DL 380 G6
- **Procesador:** Intel Xeon E5520 @ 2.27 GHz
- **Núcleos del procesador:** 8 núcleos con 2 hilos cada uno (16 hilos de ejecución en total)
- **Memoria RAM:** 18 GB (2 GB están dedicados al firmware del servidor)
- **Capacidad del disco 1:** 1.82 TB
- **Capacidad del disco 2:** 926,50 GB
- **Licencia de software:** ESXi 4 Single Server

Dentro de este servidor se han instalado cuatro máquinas virtuales, todas con las mismas especificaciones:

- **Hilos de ejecución:** 2 hilos de ejecución
- **Memoria RAM:** 4 GB
- **Espacio de disco:** 129 GB del disco 2
- **Sistema Operativo:** CentOS 6.4 64-bit
- **Distribución Hadoop instalada:** Cloudera CDH 4

Adicionalmente también se ha usado los portátiles prestados por Everis para externalizar algunos de los procesos a modo de apoyo periférico (y realizar la implementación del código). Las especificaciones de los portátiles son las siguientes:

- **Modelo:** Lenovo ThinkPad L510
- **Procesador:** Intel Core2 Duo P8800 @ 2.66 GHz
- **Núcleos del procesador:** 2 núcleos
- **Memoria RAM:** 4 GB
- **Capacidad del disco:** 300 GB
- **Sistema operativo:** Windows 7 Enterprise

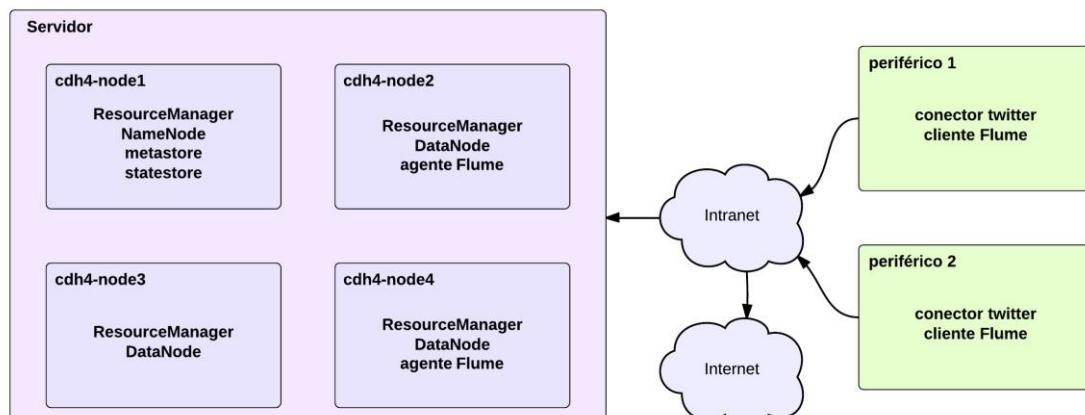


Figura 55: Esquema de la arquitectura empleada en el proyecto.

En la Figura 55 se puede ver un esquema de la arquitectura usada en el proyecto. Los ordenadores periféricos se conectaban a las máquinas virtuales del servidor a través de la intranet de la empresa y tenían acceso a internet -el servidor no tenía acceso a fuera-. Para poder trabajar más cómodamente y

Anexos

de manera remota se ha usado el software vSphere Client de VMWare que permite trabajar en una máquina externa -conectada a través de una red- y visualizar la pantalla en una ventana.

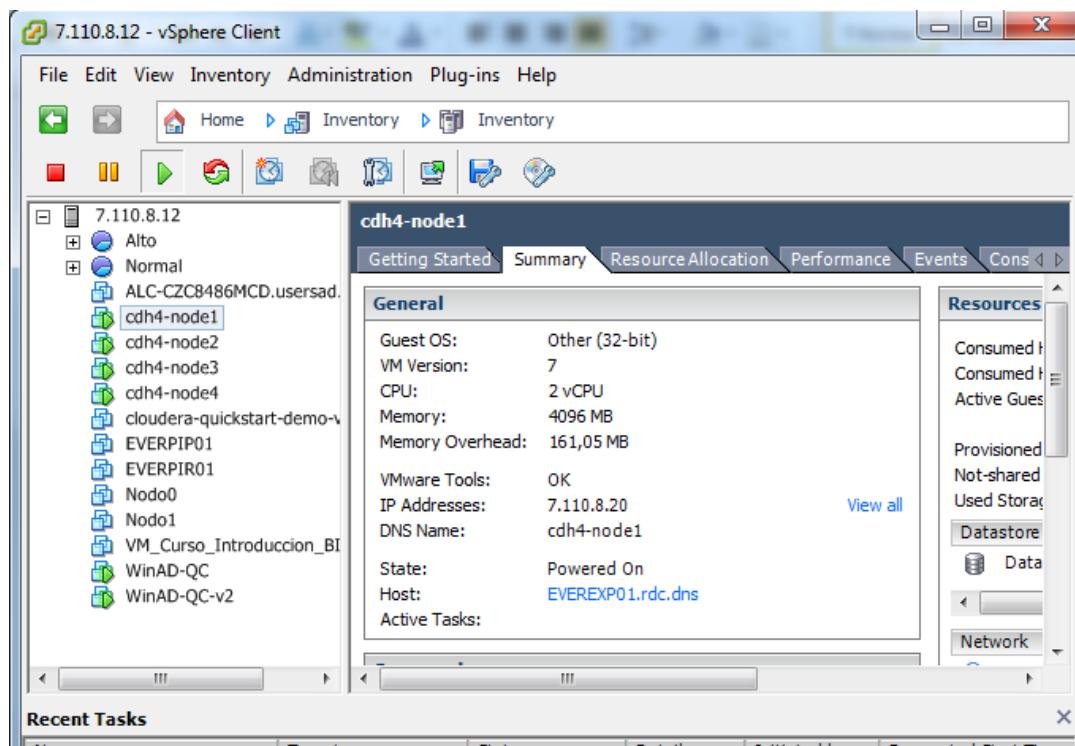


Ilustración 17: vSphere conectado al servidor.

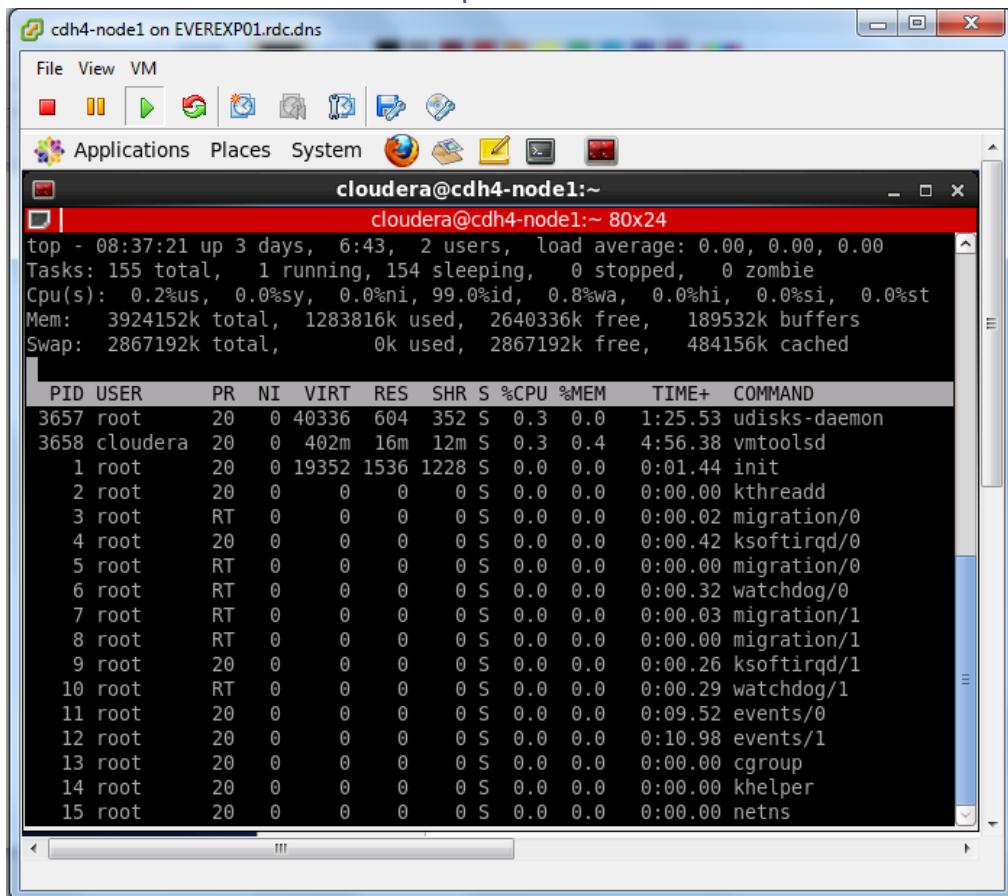


Ilustración 18: Trabajando remotamente en un nodo a través de vSphere.

D. PROCESOS DE INSTALACIÓN

En este apartado se detalla la instalación de una distribución Cloudera CDH4 sobre un clúster como el especificado en el apartado C. Arquitectura. Primero de todo se descarga el instalador de Cloudera Manager, que será la herramienta que se usará para instalar la distribución, y se instala en uno de los nodos.

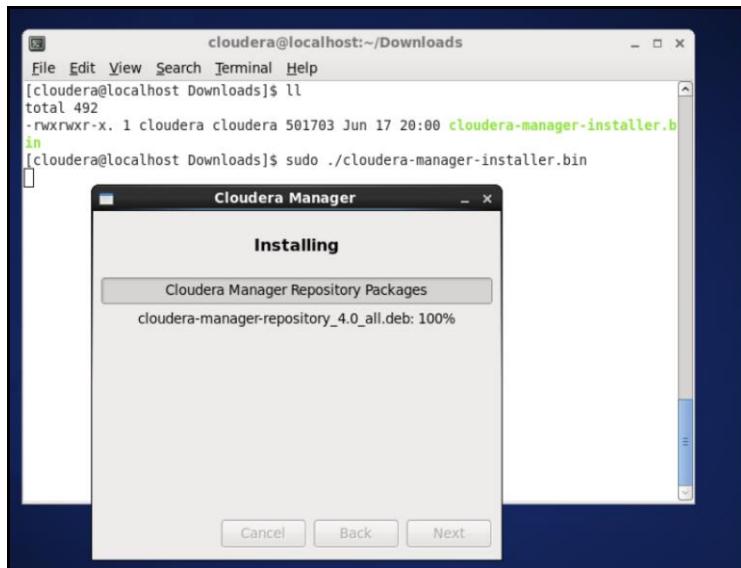


Ilustración 19: Instalando Cloudera Manager.

Una vez Cloudera Manager está instalado se ejecuta y a través de su interfaz web se van siguiendo los pasos. Al principio se buscan y seleccionan los nodos que formarán el clúster de la distribución; la búsqueda se realiza a través de la red usando sus hostnames.

Specify hosts for your CDH cluster installation.				
Cloudera recommends including Cloudera Manager server's host because it is often used for the Cloudera Management Service, and because this will enable health monitoring for that host. Hint: Search for hostnames and/or IP addresses using patterns .				
4 hosts scanned, 4 running SSH. New Search				
Expanded Query	Hostname (FQDN)	IP Address	Currently Managed	Result
<input checked="" type="checkbox"/>	bbd-node1	192.168.216.137	No	✓ Host ready: 2 ms response time.
<input checked="" type="checkbox"/>	bbd-node2	192.168.216.134	No	✓ Host ready: 4 ms response time.
<input checked="" type="checkbox"/>	bbd-node3	192.168.216.135	No	✓ Host ready: 6 ms response time.
<input checked="" type="checkbox"/>	bbd-node4	192.168.216.136	No	✓ Host ready: 1 ms response time.

Ilustración 20: Seleccionando los nodos del clúster.

El siguiente paso es escoger que tipo de instalación se quiere: por parcelas o por paquetes. En el caso de escoger una instalación por paquetes el usuario debe seleccionar los paquetes y un repositorio donde descargarlo. Esta manera es adecuada en entornos que se desea instalar versiones determinadas de las herramientas o en servidores sin acceso a internet. En el caso específico de este proyecto se siguió una instalación por paquetes ya que no se tenía acceso a los repositorios oficiales y se tuvo que configurar uno propio con todos los paquetes necesarios. Las ilustraciones mostradas en este apartado son de una instalación en local sobre cuatro máquinas virtuales idénticas a las usadas en el proyecto.

Anexos

Choose Method:

- Use Packages (1)
- Use Parcels (Recommended) (1)
- [More Options](#)

CDH	<input checked="" type="radio"/> CDH-4.3.0-1.cdh4.3.0.p0.22
SOLR	<input checked="" type="radio"/> SOLR-0.9.0-1.cdh4.3.0.p0.211
	<input type="radio"/> None

Note: Solr is supported only on CDH 4.3 or later deployments.

IMPALA	<input checked="" type="radio"/> IMPALA-1.0.1-1.p0.431
	<input type="radio"/> None

Note: Impala is supported only on CDH 4.1 or later deployments.

Select the specific release of the Cloudera Manager Agent you want to install on your hosts.

- Matched release for this Cloudera Manager server
- Custom Repository

Ilustración 21: Configurando una instalación por parcelas.

Para permitir la instalación en todos los nodos se tienen que configurar las credenciales SSH para que el Cloudera Manager tenga acceso a los demás nodos.

You may connect via password or public-key authentication for the user selected above.

Authentication Method: All hosts accept same password
 All hosts accept same private key

Enter Password:

Confirm Password:

SSH Port:

Number of simultaneous installations: (Running a large number of installations at once can cause performance issues.)

Ilustración 22: Configurando el acceso SSH de Cloudera Manager a los demás nodos.

A partir de aquí Cloudera Manager empieza la instalación de los paquetes o parcelas seleccionadas en los distintos nodos del clúster. Una vez pasado este punto los nodos tendrán instalado Hadoop pero faltará realizar la configuración e instalación de las herramientas.

Cluster Installation				
Installation in progress.				
1 of 4 host(s) completed successfully. Abort Installation				
Hostname	IP Address	Progress	Status	
bbd-node1.localdomain	192.168.216.137	<div style="width: 100%; background-color: #2e6b2e; height: 10px;"></div>	Installation completed successfully.	Details <small>(1)</small>
bbd-node2.localdomain	192.168.216.134	<div style="width: 0%; background-color: #007bff; height: 10px;"></div>	Waiting for newly installed agent to heartbeat...	Details <small>(1)</small>
bbd-node3.localdomain	192.168.216.135	<div style="width: 0%; background-color: #007bff; height: 10px;"></div>	Waiting for newly installed agent to heartbeat...	Details <small>(1)</small>
bbd-node4.localdomain	192.168.216.136	<div style="width: 0%; background-color: #007bff; height: 10px;"></div>	Waiting for newly installed agent to heartbeat...	Details <small>(1)</small>

Ilustración 23: Proceso de instalación de las parcelas o paquetes en los nodos.

Anexos

El primer paso de la configuración es elegir las herramientas a instalar, Cloudera Manager da bastantes opciones a elegir según las necesidades del proyecto.

Choose a combination of services to install.

- Core Hadoop**
HDFS, MapReduce, ZooKeeper, Oozie, Hive, and Hue
- Core with Real-Time Delivery**
HDFS, MapReduce, ZooKeeper, HBase, Oozie, Hive, and Hue
- Core with Real-Time Query**
HDFS, MapReduce, ZooKeeper, Impala, Oozie, Hive, and Hue
- All Services**
HDFS, MapReduce, ZooKeeper, HBase, Impala, Oozie, Hive and Sqoop.
- Custom Services**
Choose your own services. Services required by chosen services must also be selected. Note that Flume and Solr services can be added after your initial cluster has been set up.

Service Type	Description
<input checked="" type="checkbox"/>  HDFS	Apache Hadoop Distributed File System (HDFS) is the primary storage system used by Hadoop applications. HDFS creates multiple replicas of data blocks and distributes them on compute hosts throughout a cluster to enable reliable, extremely rapid computations.
<input checked="" type="checkbox"/>  MapReduce	Apache Hadoop MapReduce supports distributed computing on large data sets across your cluster (requires HDFS).
<input checked="" type="checkbox"/>  YARN	Apache Hadoop MapReduce 2.0 (MRv2), or YARN, is a data computation framework that supports MapReduce applications (requires HDFS). <i>The current upstream MRv2 release is not yet considered stable and should not be considered production-ready at this time.</i>
<input checked="" type="checkbox"/>  ZooKeeper	Apache ZooKeeper is a centralized service for maintaining and synchronizing configuration data.
<input type="checkbox"/>  HBase	Apache HBase provides random, real-time, read/write access to large data sets (requires HDFS and ZooKeeper).

Ilustración 24: Escogiendo las herramientas a instalar.

Una vez escogidas las herramientas Cloudera Manager pide por la distribución de los procesos de las herramientas seleccionadas a través de los nodos.

	ZooKeeper		HDFS		HBase				MapReduce		YARN									
	Server	All None	DataNode	All None	NameNode	SecondaryNameNode	HttpFS	All None	Master	All None	RegionServer	All None	HBase REST Server	All None	HBase Thrift Server	All None	TaskTracker	All None	JobTracker	All None
bbd-node1.localdomain	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>			<input type="checkbox"/>		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>			<input type="checkbox"/>		<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
bbd-node2.localdomain	<input type="checkbox"/>		<input checked="" type="checkbox"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="checkbox"/>		<input type="checkbox"/>		<input checked="" type="checkbox"/>		<input type="checkbox"/>		<input type="checkbox"/>		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
bbd-node3.localdomain	<input type="checkbox"/>		<input checked="" type="checkbox"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="checkbox"/>		<input type="checkbox"/>		<input checked="" type="checkbox"/>		<input type="checkbox"/>		<input type="checkbox"/>		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
bbd-node4.localdomain	<input type="checkbox"/>		<input checked="" type="checkbox"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="checkbox"/>		<input type="checkbox"/>		<input checked="" type="checkbox"/>		<input type="checkbox"/>		<input type="checkbox"/>		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	

Ilustración 25: En este punto se escoge la asignación de roles y servicios a los nodos para cada herramienta.

Si se ha escogido instalar alguna herramienta que necesite de una base de datos para funcionar, Cloudera Manager pedirá configurar la conexión a estas mediante un puerto, base de datos y usuarios por defecto.

Service Monitor

Currently assigned to run on bbd-node4.localdomain.

Database Host Name:	localhost.localdomain:7432	Database Type:	PostgreSQL	Database Name :	smon	Username:	smon	Password:	8XC6q9XaZR	Successful
---------------------	----------------------------	----------------	------------	-----------------	------	-----------	------	-----------	------------	------------

Activity Monitor

Currently assigned to run on bbd-node4.localdomain.

Database Host Name:	localhost.localdomain:7432	Database Type:	PostgreSQL	Database Name :	amon	Username:	amon	Password:	80ZYo3iPBI	Successful
---------------------	----------------------------	----------------	------------	-----------------	------	-----------	------	-----------	------------	------------

Host Monitor

Currently assigned to run on bbd-node4.localdomain.

Database Host Name:	localhost.localdomain:7432	Database Type:	PostgreSQL	Database Name :	hmon	Username:	hmon	Password:	F1gpWLyDsH	Successful
---------------------	----------------------------	----------------	------------	-----------------	------	-----------	------	-----------	------------	------------

Ilustración 26: Configurando las herramientas que requieran de bases de datos.

Anexos

El último paso de configuración permite modificar diversos parámetros tanto de las herramientas como de Hadoop; como puertos o los directorios donde se almacenan los metadatos.

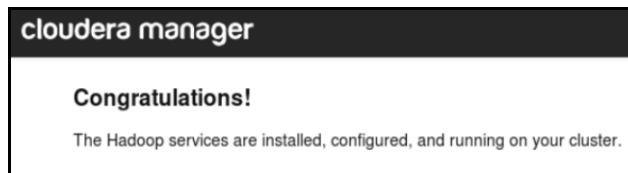
Service hive1		
Service-Wide	Hive Warehouse Directory hive.metastore.warehouse.dir	/user/hive/warehouse default value
Hive Metastore Server (Default) Show Members	Hive Metastore Server Port hive.metastore.port	9083 default value
Service mapreduce1		
JobTracker (Default) Show Members	JobTracker Local Data Directory* mapred.local.dir	/mapred/jt Reset to empty default value
TaskTracker (Default) Show Members	TaskTracker Local Data Directory List* mapred.local.dir	/mapred/local Reset to empty default value
TaskTracker (1) Show Members	TaskTracker Local Data Directory List* mapred.local.dir	/mapred/local Reset to empty default value

Ilustración 27: Configuración final de las herramientas y de Hadoop.

Una vez todo está instalado y configurado solamente queda lanzar los servicios para iniciar el clúster con la distribución de Hadoop.

Starting your cluster services.	
Completed 22 of 22 steps.	
✓ Waiting for ZooKeeper Service to initialize Finished waiting	
✓ Starting ZooKeeper Service Service started successfully.	
✓ Checking if the name directories of the NameNode are empty. Formatting HDFS only if empty. Successfully formatted NameNode.	
✓ Starting HDFS Service Service started successfully.	
✓ Creating HDFS /tmp directory Successfully created HDFS directory /tmp.	
✓ Creating HBase root directory Successfully created HBase root directory.	
✓ Starting HBase Service Service started successfully.	
✓ Starting MapReduce Service Service started successfully.	

Ilustración 28: Lanzamiento de los servicios instalados.



E. PROCESOS DE CONFIGURACIÓN

Cloudera Manager también permite añadir nuevos servicios y configurar ya existentes. En este apartado se muestran dos ejemplos de esto: instalación de Impala sobre un clúster con una distribución Hadoop ya instalada y configuración de un agente de Flume.

Select the type of service you want to add.

Service Type	Description
<input type="radio"/> HDFS	Apache Hadoop Distributed File System (HDFS) is the primary storage system used by Hadoop applications. HDFS creates multiple replicas of data blocks and distributes them on compute hosts throughout a cluster to enable reliable, extremely rapid computations.
<input type="radio"/> MapReduce	Apache Hadoop MapReduce supports distributed computing on large data sets across your cluster (requires HDFS).
<input type="radio"/> YARN	Apache Hadoop MapReduce 2.0 (MRv2), or YARN, is a data computation framework that supports MapReduce applications (requires HDFS). The current upstream MRv2 release is not yet considered stable and should not be considered production-ready at this time.
<input type="radio"/> ZooKeeper	Apache ZooKeeper is a centralized service for maintaining and synchronizing configuration data.
<input type="radio"/> HBase	Apache HBase provides random, real-time, read/write access to large data sets (requires HDFS and ZooKeeper).
<input type="radio"/> Hive	Hive is a data warehouse system that offers a SQL-like language called HiveQL.
<input type="radio"/> Oozie	Oozie is a workflow coordination service to manage data processing jobs on your cluster.
<input type="radio"/> Hue	Hue is a graphical user interface to work with Cloudera's Distribution Including Apache Hadoop (requires HDFS, MapReduce, and Hive).
<input type="radio"/> Flume	Flume collects and aggregates data from almost any source into a persistent store such as HDFS.

Ilustración 29: Pantalla de selección del servicio a añadir.

Select the set of dependencies for your new service

<input type="checkbox"/> ZOOKEEPER zookeeper1	<input type="checkbox"/> HDFS hdfs1	<input type="checkbox"/> MAPREDUCE mapreduce1	<input type="checkbox"/> HIVE hive1
--	--	--	--

Ilustración 30: Selección de las dependencias del servicio.

Hostname	Rack	Health	Roles Currently Assigned to Host	Impala Daemon <input checked="" type="radio"/> All <input type="radio"/> None	Impala StateStore Daemon <input checked="" type="radio"/> Select One
bbd-node1.localdomain	/default	Good	3 Dependencies DataNode for HDFS hdfs1 Gateway for Hive hive1 TaskTracker for MapReduce mapreduce1	<input checked="" type="checkbox"/>	<input type="radio"/>
bbd-node2.localdomain	/default	Good	3 Dependencies DataNode for HDFS hdfs1 Gateway for Hive hive1 TaskTracker for MapReduce mapreduce1	<input checked="" type="checkbox"/>	<input type="radio"/>
bbd-node3.localdomain	/default	Good	3 Dependencies DataNode for HDFS hdfs1 Gateway for Hive hive1 TaskTracker for MapReduce mapreduce1	<input checked="" type="checkbox"/>	<input type="radio"/>
bbd-node4.localdomain	/default	Good	8 Dependencies DataNode for HDFS hdfs1 Gateway for Hive hive1 Hive Metastore Server for Hive hive1 JobTracker for MapReduce mapreduce1 NameNode for HDFS hdfs1 SecondaryNameNode for HDFS hdfs1 Server for ZooKeeper zookeeper1 TaskTracker for MapReduce mapreduce1	<input checked="" type="checkbox"/>	<input checked="" type="radio"/>

Ilustración 31: Selección de la distribución de roles entre los nodos.

Group	Parameter	Current Value <input type="button" value="Select All"/>	New Value <input type="button" value="Select All"/>	Description
Service hdfs1				
Service-Wide	DataNode Local Path Access Users dfs.block.local-path-access.user	impala	<input checked="" type="checkbox"/>	Comma separated list of users allowed to do short circuit read. A short circuit read allows a client co-located with the data to read HDFS file blocks directly from HDFS. If empty, will default to the DataNode process' user.
DataNode (Default) <input type="button" value="Show Members"/>	DataNode Data Directory Permissions dfs.datanode.data.dir.perm	700	<input checked="" type="checkbox"/>	Permissions for the directories on the local file system where the DataNode stores its blocks. The permissions must be octal. 755 and 700 are typical values.
DataNode (1) <input type="button" value="Show Members"/>	DataNode Data Directory Permissions dfs.datanode.data.dir.perm	700	<input checked="" type="checkbox"/>	Permissions for the directories on the local file system where the DataNode stores its blocks. The permissions must be octal. 755 and 700 are typical values.

Ilustración 32: Pantalla con los cambios realizados en el clúster una vez desplegado el nuevo servicio.

Anexos

Name	Status	Role Counts	Actions
flume1	None	None	Actions ▾
hdfs1	Good Health	1 SecondaryNameNode, 1 NameNode, 4 DataNodes	Actions ▾
hive1	Good Health	1 Hive Metastore Server, 4 Gateways	Actions ▾
hue1	Good Health	1 Beeswax Server, 1 Hue Server	Actions ▾
impala1	Good Health	4 Impala Daemons, 1 Impala StateStore Daemon	Actions ▾
mapreduce1	Good Health	1 JobTracker, 4 TaskTrackers	Actions ▾
oozie1	Good Health	1 Oozie Server	Actions ▾
zookeeper1	Good Health	1 Server	Actions ▾

Ilustración 33: Lista de los servicios desplegados en el clúster y su estado.

Add Role Instances to flume1				
Hostname	Rack	Health	Roles Currently Assigned to Host	Agent All None
bbd-node1.localdomain	/default	✓ Good	1 Dependencies DataNode for HDFS hdfs1 4 Other Roles	<input type="checkbox"/>
bbd-node2.localdomain	/default	✓ Good	1 Dependencies DataNode for HDFS hdfs1 4 Other Roles	<input type="checkbox"/>
bbd-node3.localdomain	/default	✓ Good	1 Dependencies DataNode for HDFS hdfs1 4 Other Roles	<input type="checkbox"/>
bbd-node4.localdomain	/default	✓ Good	4 Dependencies DataNode for HDFS hdfs1 NameNode for HDFS hdfs1 SecondaryNameNode for HDFS hdfs1 Server for ZooKeeper zookeeper1 11 Other Roles	<input checked="" type="checkbox"/>

Ilustración 34: Pantalla de selección de roles para añadir un agente de Flume.

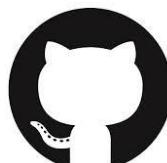
Default ► Service-Wide ▼ Agent (Default) Resource Management Ports and Addresses Logs Advanced Flume-NG Solr Sink Monitoring	Agent Name tier1 <small>default value</small> Configuration File <pre># Please paste flume.conf here. Example: # Sources, channels, and sinks are defined per # agent name, in this case 'tier1'. tier1.sources = source1 tier1.channels = channel1 tier1.sinks = sink1 # For each source, channel, and sink, set # standard properties. tier1.sources.source1.type = netcat tier1.sources.source1.bind = 127.0.0.1 tier1.sources.source1.port = 9999 tier1.sources.source1.channels = channel1 tier1.channels.channel1.type = memory tier1.sinks.sink1.type = logger tier1.sinks.sink1.channel = channel1 # Other properties are specific to each type of # source, channel, or sink. In this case, we # specify the capacity of the memory channel. tier1.channels.channel1.capacity = 100 default value</pre>
---	---

Ilustración 35: Configuración de un agente de Flume a través de Cloudera Manager.

F. CÓDIGO

En este apartado se explica la estructura del código implementado tanto para los casos de uso como para las pruebas.

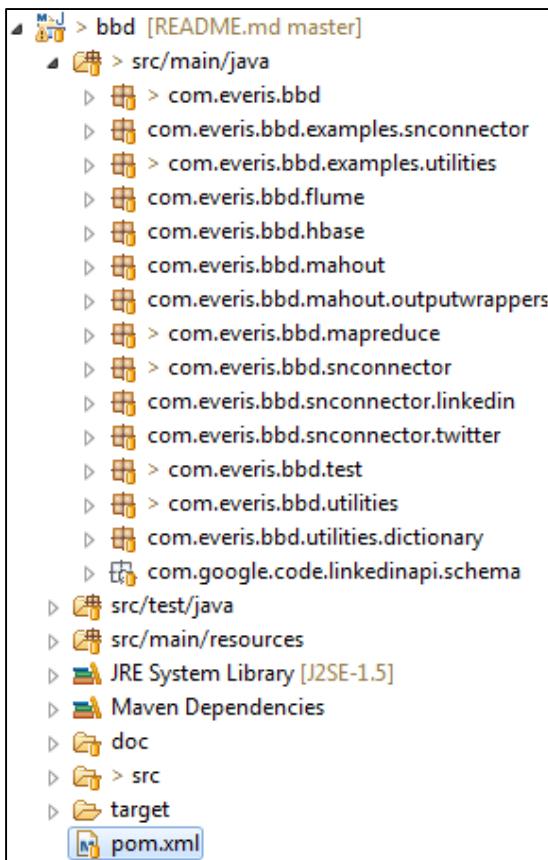
El código del proyecto se puede encontrar en un repositorio de GitHub. GitHub es un servicio web gratuito que ofrece repositorios Git para albergar proyectos. De esta manera se podía trabajar en paralelo cada uno implementando su parte sin interferir al otro. La dirección del repositorio del proyecto en GitHub es:



<https://github.com/LinJuuichi/BecaBigData>

Para la gestión del proyecto y mantener organizado el código y sus dependencias se ha usado Maven, una herramienta que está desarrollada para estas labores. El código está implementado la mayor parte en Java y en menor medida en scripts de bash, con la función de realizar flujos de trabajo o lanzar aplicaciones con diversas opciones. Además se ha usado Eclipse como IDE para el desarrollo, con la siguiente estructura:

- **src/main/java**: código Java del proyecto estructurado en paquetes con la raíz com.everis:



- **bbd**: código de las aplicaciones.
- **examples**: código con ejemplos.
- **flume**: código referente a Flume.
- **hbase**: código para HBase.
- **mahout**: código para Mahout.
- **mapreduce**: implementación de diferentes MapReduces.
- **snconnector**: código de los conectores para redes sociales.
- **test**: código para las pruebas.
- **utilities**: clases de utilidades usadas para diferentes propósitos.

También se cuenta con un directorio para la documentación, además de para los ficheros de configuración y scripts bash.

Ilustración 36: Estructura del código en eclipse.

BIBLIOGRAFÍA

1. **Everis.** Informe anual 2011-2012. *everis*. [En línea] everis group, 2012. [Consultado el: 05 de Mayo de 2013.] <http://www.everis.com/spain/es-ES/sobre-everis/compania/PublishingImages/HTML/index.html>.
2. **Zikopoulos, Paul, y otros.** *Harness the Power of Big Data*. [ed.] Roman Melnyk. s.l. : The McGraw-Hill Companies, 2012. pág. 281. 978-0-07180818-7.
3. **Rogers, Shawn.** Big Data is Scaling BI and Analytics. *Information Management*. [En línea] Source Media. [Consultado el: 8 de Marzo de 2013.] http://www.information-management.com/issues/21_5/big-data-is-scaling-bi-and-analytics-10021093-1.html?zkPrintable=1&nopagination=1.
4. **Versace, Michael.** The Case for Big Data in the Financial Services Industry. *Slide Share*. [En línea] Septiembre de 2012. [Consultado el: 7 de Marzo de 2013.] <http://www.slideshare.net/HPenterprise/the-case-for-big-data-in-the-financial-services-industry>.
5. **Cloud Partners Inc.** Use Cases for Hadoop. *Cloud Partners*. [En línea] Septiembre de 2012. [Consultado el: 7 de Marzo de 2013.] <http://www.cloudpartnerstm.com/wp-content/uploads/2012/09/Use-Cases-for-Hadoop.pdf>.
6. **Harris, Derrick.** How Hadoop can help keep your money in the bank. *GigaOm*. [En línea] GigaOm, 8 de Marzo de 2012. [Consultado el: 6 de Marzo de 2013.] <http://gigaom.com/2012/03/08/how-hadoop-can-help-keep-your-money-in-the-bank/>.
7. **Jose, Derick.** 3 Game Changing Big Data Use Cases in Telecom. *Futura Solutions*. [En línea] 17 de Julio de 2012. [Consultado el: 7 de Marzo de 2013.] <http://blog.fluturasolutions.com/2012/07/3-game-changing-big-data-use-cases-in.html>.
8. **Stiglich, Pete y Rajagopal, Hari.** Using Big Data for Improved Healthcare Operations and Analytics. *Slide Share*. [En línea] 30 de Mayo de 2012. [Consultado el: 7 de Marzo de 2013.] <http://www.slideshare.net/perficientinc/using-big-data-for-improved-healthcare-operations-and-analytics>.
9. **Gawande, Atul.** The Hot Spotters. *The New Yorker*. [En línea] Condé Nast, 24 de Enero de 2011. [Consultado el: 7 de Marzo de 2013.] http://www.newyorker.com/reporting/2011/01/24/110124fa_fact_gawande?currentPage=all&mobify=0.
10. **Greenplum.** Predicting the Next Pandemic. *Big Data Use Case*. [En línea] EMC. [Consultado el: 7 de Marzo de 2013.] <http://www.greenplum.com/industry-buzz/big-data-use-cases/predicting-the-next-pandemic>.
11. **Pentaho.** Pentaho Big Data Analytics. *Pentaho Big Data Analytics Center*. [En línea] Pentaho, 2013. [Consultado el: 14 de Octubre de 2013.] <http://www.pentahobigdata.com/ecosystem/capabilities/analytics>.
12. **Wikipedia.** MapReduce. *Wikipedia - La enciclopedia libre*. [En línea] Fundación Wikipedia Inc., 11 de Marzo de 2013. [Consultado el: 15 de Mayo de 2013.] <http://es.wikipedia.org/wiki/MapReduce>.
13. **Mythics.** Mythics & Big Data. *Mythics*. [En línea] Mythics. [Consultado el: 20 de Mayo de 2013.] <http://www.mythics.com/solutions/mythics-big-data/>.

Anexos

14. **Landa, Ibon.** Big Data, Hadoop y Windows Azure. *estoy en la nube.* [En línea] Plain Concepts Corp, 10 de Abril de 2012. [Consultado el: 20 de Mayo de 2013.] <http://www.estoyenlanube.com/big-data-hadoop-y-windows-azure-ii/>.
15. **Apache.** Hadoop - Who we are. *Apache Hadoop.* [En linea] The Apache Software Foundation, 30 de Agosto de 2013. <http://hadoop.apache.org/who.html>.
16. **DeBie, Wouter.** PoweredBy. *Hadoop Wiki.* [En línea] The Apache Foundation, 19 de Junio de 2013. <http://wiki.apache.org/hadoop/PoweredBy>.
17. **The Apache Software Foundation.** Overview. *Hadoop 1.2.1 Documentation.* [En línea] The Apache Software Foundation., 8 de Agosto de 2013. <http://hadoop.apache.org/docs/r1.2.1/>.
18. **Borthakur, Dhruba Borthakur.** HDFS Architecture Guide. *Apache Hadoop.* [En línea] Apache, 2 de Febrero de 2013. [Consultado el: 2 de Marzo de 2013.]
http://hadoop.apache.org/docs/r1.0.4/hdfs_design.html.
19. **The Apache Software Foundation.** Apache Hadoop 2.2.0 documentation. *Apache Hadoop 2.2.0.* [En línea] The Apache Software Foundation, 2013. <http://hadoop.apache.org/docs/r2.2.0/>.
20. —. Apache Avro™ 1.7.5 Specification. *Apache Avro.* [En línea] The Apache Software Foundation, 19 de Agosto de 2013. <http://avro.apache.org/docs/current/spec.html>.
21. —. Apache Solr Reference Guide. *Apache Solr.* [En línea] [Consultado el: 23 de Abril de 2013.]
<http://archive.apache.org/dist/lucene/solr/ref-guide/apache-solr-ref-guide-4.4.pdf>.
22. —. Chukwa: Architecture and Design. *Apache Chukwa.* [En línea] The Apache Software Foundation, 4 de Abril de 2010. [Consultado el: 15 de Marzo de 2013.]
<http://incubator.apache.org/chukwa/docs/r0.4.0/design.html>.
23. —. Chukwa Agent Setup Guide. *Chukwa 0.4 Documentation.* [En línea] The Apache Software Foundation, 26 de Abril de 2010. [Consultado el: 15 de Marzo de 2013.]
<http://incubator.apache.org/chukwa/docs/r0.4.0/agent.html>.
24. —. Flume User Guide. *Apache Flume.* [En línea] The Apache Software Foundation, 2012. [Consultado el: 12 de Marzo de 2013.] <http://flume.apache.org/FlumeUserGuide.html>.
25. —. Apache Hive. *Apache Hive.* [En línea] The Apache Software Foundation. [Consultado el: 5 de Abril de 2013.] <https://cwiki.apache.org/confluence/display/Hive/Home>.
26. **Chen, Charles.** Diseño. *Apache Hive.* [En línea] The Apache Software Foundation, 21 de Julio de 2011. [Consultado el: 5 de Abril de 2013.] <https://cwiki.apache.org/confluence/display/Hive/Design>.
27. **The Apache Software Foundation.** Scalable and vibrant. *Apache Mahout.* [En línea] The Apache Software Foundation, 2011. [Consultado el: 13 de Abril de 2013.] <http://mahout.apache.org/>.
28. —. Overview of mahout. *Apache Mahout cwiki.* [En línea] The Apache Software Foundation, 2011. <https://cwiki.apache.org/confluence/display/MAHOUT/Overview>.
29. —. Oozie, Workflow Engine for Apache Hadoop. *Apache Oozie Documentation.* [En línea] The Apache Software Foundation, 2013. [Consultado el: 13 de Abril de 2013.]
<http://oozie.apache.org/docs/4.0.0/index.html>.

Anexos

30. —. Oozie Workflow Overview. *Oozie Documentation*. [En línea] The Apache Software Foundation, 2013. http://oozie.apache.org/docs/4.0.0/DG_Overview.html.
31. —. Welcome to Apache Pig! *Apache Pig*. [En línea] The Apache Software Foundation, 22 de Octubre de 2013. [Consultado el: 16 de Abril de 2013.] <http://pig.apache.org/>.
32. —. Getting Started. *Apache Pig*. [En línea] The Apache Software Foundation, 14 de Octubre de 2013. [Consultado el: 13 de Abril de 2013.] <http://pig.apache.org/docs/r0.12.0/start.html#pl-statements>.
33. **Cloudera.** Cloudera Standard. *Cloudera - Ask Bigger Questions*. [En línea] Cloudera. [Consultado el: 5 de Abril de 2013.] <http://www.cloudera.com/content/cloudera/en/products/cloudera-standard.html>.
34. —. Cloudera Impala Frequently Asked Questions. *Cloudera Documentation*. [En línea] Cloudera. [Consultado el: 21 de Junio de 2013.] <http://www.cloudera.com/content/cloudera-content/cloudera-docs/Impala/latest/Cloudera-Impala-Frequently-Asked-Questions/Cloudera-Impala-Frequently-Asked-Questions.html>.
35. —. Impala Concepts and Architecture. *Cloudera Documentation*. [En línea] Cloudera. [Consultado el: 19 de Abril de 2013.] http://www.cloudera.com/content/cloudera-content/cloudera-docs/Impala/latest/Installing-and-Using-Impala/ciuu_concepts.html.
36. —. Cloudera Search. *Cloudera - Ask Bigger Questions*. [En línea] 2013. [Consultado el: 5 de Abril de 2013.]
http://www.cloudera.com/content/dam/cloudera/Resources/PDF/Cloudera_Datasheet_Cloudera_Search.pdf.
37. —. Understading Cloudera Search. *Cloudera Documentation*. [En línea] 2013. [Consultado el: 5 de Abril de 2013.] http://www.cloudera.com/content/cloudera-content/cloudera-docs/Search/latest/Cloudera-Search-User-Guide/csug_understanding.html.
38. —. Cloudera Navigator. *Cloudera - Ask Bigger Questions*. [En línea] Cloudera, Abril de 2013. [Consultado el: 19 de Abril de 2013.]
<http://www.cloudera.com/content/cloudera/en/products/cloudera-navigator.html>.
39. **Luciani, Jake.** Cassandra File System Design. *DataStax Developer Blog*. [En línea] DataStax, 11 de Febrero de 2012. [Consultado el: 24 de Abril de 2013.] <http://www.datastax.com/dev/blog/cassandra-file-system-design>.
40. **DataStax.** Comparing the HDFS with CFS. *DataStax Enterprise*. [En línea] Agosto de 2013. [Consultado el: 20 de Octubre de 2013.] <http://www.datastax.com/wp-content/uploads/2012/09/WP-DataStax-HDFSvsCFS.pdf>.
41. —. When to Choose DataStax Enterprise Edition Versus DataStax Community Edition. *DataStax*. [En línea] DataStax, 2013. [Consultado el: 20 de Octubre de 2013.]
<http://www.datastax.com/download/dse-vs-dsc>.
42. **Pivotal Inc.** Pivotal HD Enterprise. *GoPivotal*. [En línea] Pivotal Inc., Abril de 2013. [Consultado el: 20 de Abril de 2013.] <http://www.gopivotal.com/pivotal-products/data/pivotal-hd>.
43. —. MADlib. *GoPivotal*. [En línea] Pivotal Inc., Abril de 2013. [Consultado el: 20 de Abril de 2013.]
<http://www.gopivotal.com/pivotal-products/analytics/madlib>.
44. —. Pivotal GemFire. *Pivotal*. [En línea] Pivotal Inc., Septiembre de 2013. [Consultado el: 27 de Septiembre de 2013.] <http://www.gopivotal.com/pivotal-products/data/pivotal-gemfire>.

Anexos

45. —. Analytics Workbench. *GoPivotal*. [En línea] Pivotal Inc., Abril de 2013. [Consultado el: 20 de Abril de 2013.] <http://www.gopivotal.com/pivotal-services-and-solutions/pivotal-partner-solutions/analytics-workbench>.
46. **Pivotal**. Pivotal DCA. *GoPivotal*. [En línea] Pivotal, Mayo de 2013. <http://www.gopivotal.com/pivotal-products/data/pivotal-dca>.
47. —. Start using Pivotal HD Today! *Pivotal HD / GoPivotal*. [En línea] [Consultado el: 15 de Abril de 2013.] <http://gopivotal.com/pivotal-products/data/pivotal-hd>, Agosto de 2009. [Consultado el: 20 de Octubre de 2013.] <http://infolab.stanford.edu/~echang/recsys08-69.pdf>.
49. **The Apache Software Foundation**. MapReduce Tutorial. *MapReduce*. [En línea] The Apache Software Foundation, 2013 de Abril de 8. [Consultado el: 2013 de Mayo de 15.] http://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html#Overview.

