Universidad de Alcalá
Escuela Politécnica Superior

GRADO EN SISTEMAS DE INFORMACIÓN

**Trabajo Fin de Grado**

# Sistema de visión 360º con una plataforma Big Data Hadoop

**Autor:** Luis Fernando Nueda Garcia

**Tutor/es:** Dr. D. Iván González Diego



2017

# UNIVERSIDAD DE ALCALÁ

# Escuela Politécnica Superior

## GRADO EN SISTEMAS DE INFORMACIÓN

Trabajo Fin de Grado

Sistema de visión 360º con una plataforma Big Data Hadoop

**Autor:** Luis Fernando Nueda Garcia

**Tutor/es:** Dr. D. Iván González Diego

**TRIBUNAL:**

**Presidente:**

**Vocal 1º:**

**Vocal 2º:**

**FECHA:**

# 1. ÍNDICE

# 2. RESUMEN

Big Data es un conjunto de tecnologías que permiten a las organizaciones almacenar y analizar ingentes volúmenes de información con bajos tiempos de respuesta.Big data es volumen, velocidad y variabilidad. Se planteará un caso de uso para demostrar el potencial de esta tecnología, se irá por Velocidad y Variabilidad, que consistirá en cruzar información dispar, ¿Qué información? Toda la que se pueda extraer desde la API de Twitter sobre un evento deportivo, se cruzará toda la información extraída y se procesará con tecnología Big Data a la par que se hará un recorrido sobre las tecnologías más punteras del ecosistema: Hadoop.

# 3. RESUMEN EN INGLÉS

Big data is considered a set of technologies, which allows organisations to save and analyse huge amounts of information very efficiently and very fast. Big data is volume, velocity and variability. A business case will be performed in order to demonstrate the potential of the technology going through velocity and variability from different sources of information. What kind of information? All possible data extraction from a twiter API about an sport event will be processed with big data technology following the top technologies from the Hadoop environment.

# 4. MEMORIA

## 4.1. Planteamiento del trabajo y objetivos.

Cuando llegó el momento de plantear un tema para el TFG, surgieron una multitud de temas, ya que en nuestra titulación se ha abarcado una gran variedad de tecnologías, pero hay una en especial que actualmente está muy de moda y muy mal tratado por la industria de la Tecnología de la Información, por el desconocimiento real de las tecnologías que hay detrás de este concepto. Después de un largo estudio se va a intentar hacer una aproximación que sea didáctica a los presentes. Para ello se va a hacer un breve recorrido histórico del problema clásico de cómo organizar la información para un consumo eficiente de la misma. Fue uno de los primeros problemas que se resolvieron con tecnología que se puede denominar Big Data, para luego pasar a revisar las características que debe tener una arquitectura para denominarla Big Data y finalizar con casos de uso y simulaciones.

Ya en la antigüedad, el problema de recopilar información relevante de forma eficaz se remonta a la época de Tolomeo II (hijo de Tolomeo I) quien funda la biblioteca de Alejandría allá por el siglo III a. C.. Tolomeo II tenía como objetivo recopilar todo el conocimiento de la época (obras de teatro, poemas épicos, tratados de filosofía, medicina, matemáticas, retórica y cualquier rama del saber de la época) para construir la mayor y mejor biblioteca del mundo, pero llegó un momento que se vio desbordado por la enorme cantidad de rollos de papiro que tenía la biblioteca (se estima que llegó a albergar 900.000 manuscritos). Entonces Tolomeo II contactó con Zenodoto que, después de visitar la biblioteca, comprendió que ordenar todo aquello era clave, pues la biblioteca no valía nada por el mero hecho de acumular centenares de rollos si nadie era capaz de encontrar uno cuando necesitara consultarlo. Después de noches de insomnio pensando cómo clasificar aquellas montañas de cestos de manuscritos, recordó el glosario de palabras antiguas de Homero que él había ordenado por grupos: los que empezaban por A todos juntos, luego los que empezaban por B y así sucesivamente. Al principio le pareció demasiado simple, pero pronto se dio cuenta de que aquello funcionaba muy bien para localizar una palabra sobre la que hubiera trabajado. Así decidió ordenar todos los rollos por orden alfabético según su autor. La tarea llevó meses, años, pero Zenodoto tuvo

tiempo de ver en vida aquella inmensa biblioteca con todos los centenares de miles de rollos archivados y localizables.

Tuvieron que pasar unos veintitrés siglos para encontrarse de nuevo con el problema de recuperar información relevante de forma eficaz. Nos encontramos en plena burbuja de Internet donde el número de páginas Web crece exponencialmente, al igual que los usuarios inexpertos que empiezan a acceder a Internet. Estos nuevos usuarios para poder encontrar información relevante podrían consultar páginas de índices como Yahoo (mantenidas de forma manual) y motores de búsquedas basados en palabras clave como AltaVista. En muchas ocasiones los resultados de las búsquedas realizadas por los usuarios no resultaban relevantes y además los sistemas de búsquedas empezaban a tener problemas de rendimiento.

En 1996 Sergey Brin y Lawrence Page comienzan a colaborar en la creación de un motor de búsqueda llamado BackRub en la Universidad de Stanford y escriben el artículo “The Anatomy of a Large-Scale Hypertextual Web Search Engine” donde presentan un prototipo de un motor de búsqueda a gran escala, llamado Google, que hace un uso intensivo de la estructura presente en hipertextos (documentos que hacen referencias a otros documentos mediante híper-enlaces). En dicho artículo los creadores de Google muestran su preocupación por la escalabilidad del motor de búsqueda y las limitaciones que tienen las arquitecturas centralizadas de índices para dar unos tiempos de respuesta aceptables para el usuario. Como respuesta a este problema de escalabilidad y rendimiento, los investigadores de Google desarrollan el sistema de ficheros distribuido Google File System (2003) y el framework de procesamiento de datos distribuido MapReduce (2004), que permiten el análisis de enormes volúmenes de conjuntos de datos. Se entra en la Era del Big Data.

Posteriormente, en 2005, Yahoo crea un proyecto open source llamado Apache Hadoop, originariamente desarrollado para soportar el motor de búsqueda Nutch pero que luego se convirtió en un estándar de facto para lo que se denomina actualmente arquitecturas Big Data.

A partir de aquí todos los grandes fabricantes de software (IBM, Oracle, Microsoft, etc.) se han sumado a esta tendencia tecnológica incorporando a sus productos diversos componentes de Apache Hadoop para poder vender sus productos como Big Data. Además, han aparecido un conjunto de fabricantes especializados en Apache Hadoop que han creado sus propias

distribuciones (conjunto de componentes testados e integrados entre sí, así como herramientas de administración y mantenimiento) que ofrecen servicios tanto de consultoría para la implantación como soporte y mantenimiento. Cabe destacar las distribuciones de Apache Hadoop de Cloudera, MapR Technologies y Hortonworks, que contribuyen en la evolución de Apache Hadoop.

Por otro lado, en muchas ocasiones se identifica Big Data con Apache Hadoop, pero existen otras arquitecturas consideradas Big Data como por ejemplo MongoDB, Google BigQuery y Amazon DynamoDB entre otros, que tienen una aproximación distinta a Apache Hadoop, aunque empiezan a tener funcionalidades cada vez más parecidas.

Con este estudio se quiere demostrar el potencial de Big Data con respecto a las tecnologías ya conocidas utilizando un ecosistema Hadoop, simulando un ecosistema completo con sus respectivas tecnologías y arquitecturas, y aplicarlo a un caso de uso.

## 4.2. Conceptos teóricos utilizados.

### 4.2.1. Introducción

#### 4.2.1.1. El desafío de los datos

En los últimos años, debido al avance de las Tecnologías de la Información, estamos siendo testigos de una verdadera explosión en la cantidad de datos disponibles, listos para ser analizados y así convertirse en información importante para la inteligencia de los negocios. Este nuevo escenario se refiere no sólo al volumen de datos, sino también a la velocidad, complejidad y variedad de los tipos de información disponible, como acontece con los datos de las redes sociales, logs de acceso a Internet o datos generados por máquinas…. Por lo tanto, los modelos tradicionales de Data Warehouse y soluciones analíticas, desarrolladas para ofrecer soporte a ese mundo de informaciones, encuentran un nuevo desafío relacionado, sobre todo, a la manipulación de ese contenido, acompañado de toda esa complejidad y dinamismo: es aquí donde surge el concepto de Big Data.

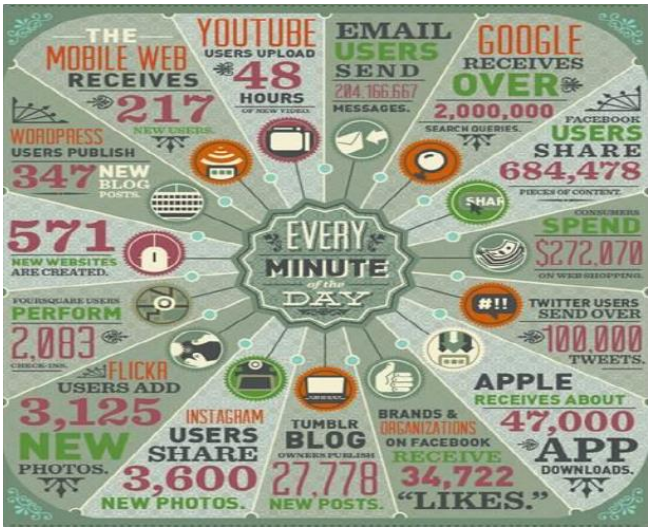



Ilustración 1 – Big Data.

Los Big Data o Grandes Datos son datos ricos y extremadamente útiles para análisis, pero que no se encuentran disponibles, al menos inicialmente de una manera estructurada, ya sea por la alta velocidad con que son producidos o por los mecanismos a través de los cuales son

generados. Siendo así, más allá de la gran cantidad de información disponible hoy, los Big Data se relacionan directamente a la capacidad de manipular y analizar datos multi-estructurados no relacionados, que requieren de una interacción rápida y adaptable.

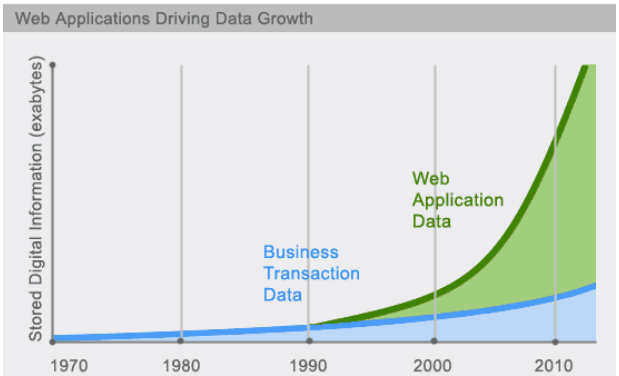



**Ilustración 2 - Web Application Driving Data Growth.**

Como soporte a esa nueva realidad, ya están disponibles nuevas técnicas y tecnologías como MapReduce o Hadoop, que resuelven limitaciones de SQL tradicional, para tratar los variados tipos de datos complejos disponibles en Internet o en otras fuentes. Desde el punto de vista de hardware, los Big Data pueden recurrir a tecnologías existentes, como las arquitecturas de Procesamiento Paralelo Masivo (MPP), que permiten el rápido procesamiento de estos grandes y complejos volúmenes de datos.

Muchas empresas ya están comenzando a desarrollar su propia lógica utilizando ese tipo de infraestructura, posibilitando análisis importantes como el comportamiento de los clientes en casos de optimización del marketing digital, las interacciones entre usuarios en las redes sociales, transacciones e interacciones on-line para detección y prevención del fraude y la eficiencia operacional por datos generados por máquinas, entre otros.

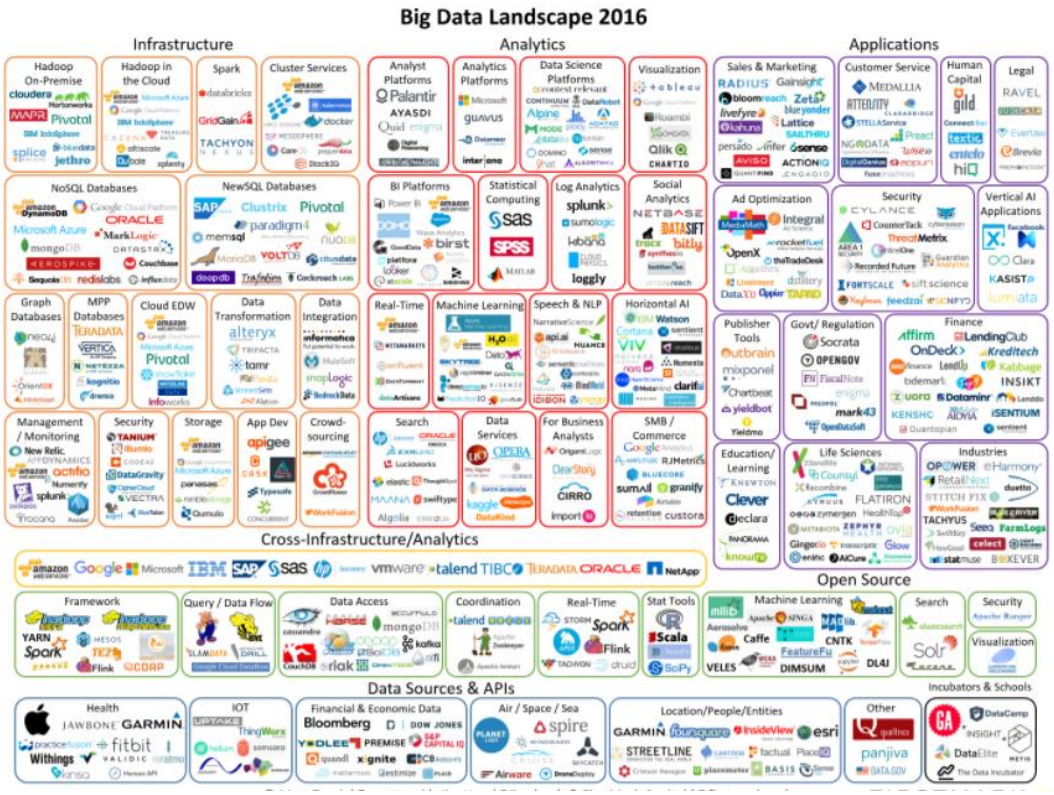



Ilustración 3 - Big Data Landscape 2016.

### 4.2.1.2. Big Data

Big Data es un conjunto de tecnologías que permiten a las organizaciones almacenar y analizar ingentes volúmenes de información con bajos tiempos de respuesta.

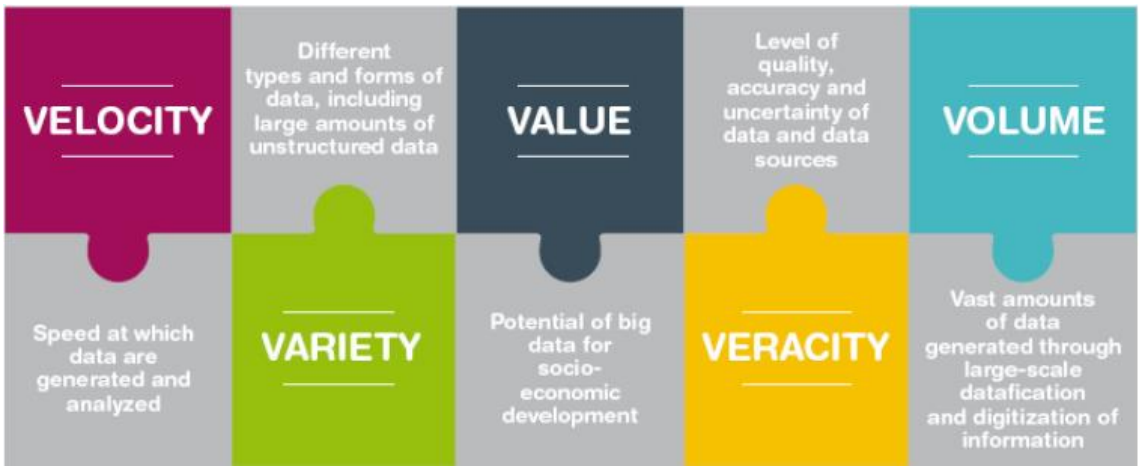



Ilustración 4 - Las cinco 'V'.

## 4.2.2. Introducción a Hadoop

### 4.2.2.1. ¿Por qué se necesita Hadoop?

La escalabilidad horizontal y vertical.

En los inicios, el trabajo era realizado por una única máquina,  poseían poca información, pero compleja de procesar .

Lo que ocurría era que se encontraban con el famoso cuello de botella ( microprocesador, memoria), y las únicas medidas que se podían aplicar ante la insuficiente potencia, eran:

- Más memoria, más microprocesador.
- Era una solución cara.
- El hardware podía no existir aun en el mercado.

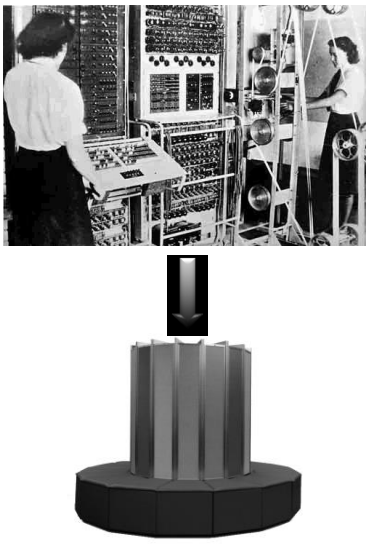


Ilustración 5 - Introducción a Hadoop

Con la escalabilidad horizontal, los sistemas distribuidos, el trabajo es realizado por varias máquinas:

- La información se distribuye para su procesamiento.
- Cada máquina procesa parte de la información.
- Al finalizar todos se unifica el resultado.

- La programación distribuida es compleja:
    - Transmitir la información a procesar.
    - Consolidar los resultados parciales.
    - Sincronizar procesos
    - Gestionar errores
        - Red: la red no es confiable.
        - Hardware

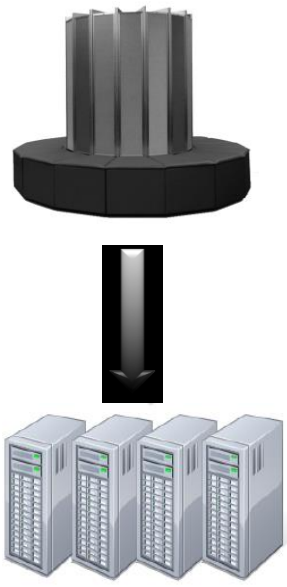


Ilustración 6 - Introducción a Hadoop 2

#### 4.2.2.2. Hadoop

Ilustración 7 - Hadoop

Apache Hadoop es un framework de código abierto que permite el almacenamiento distribuido y el procesamiento de grandes conjuntos de datos en base a un hardware comercial. Está diseñado para escalar desde los servidores individuales a miles de máquinas, cada una ofreciendo computación y almacenamiento local, en lugar de depender de hardware para ofrecer alta disponibilidad. El framework en sí está diseñado para detectar y manejar los fallos en la capa de aplicación, en otras palabras, Hadoop hace posible a las organizaciones obtener conocimiento rápidamente a partir de cantidades masivas de datos, estructurados y no estructurados, posicionándolas al nivel de las exigencias actuales de los mercados en términos de dinamismo y capacidad.

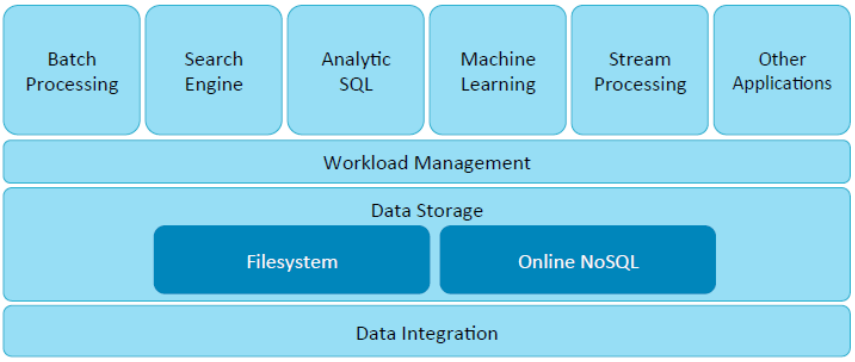



Ilustración 8 - Ecosistema Hadoop.

El ecosistema Hadoop cuenta con soluciones de todo tipo para cubrir cualquier necesidad que pueda presentarse al negocio en materia de:

- Gestión de datos.
- Acceso a los datos.

- Gobernabilidad e integración de datos.
- Seguridad de la información.
- Operaciones.

Son precisamente estas funcionalidades las que mejor definen qué es Apache Hadoop aunque, para conocer realmente las posibilidades de esta herramienta y el secreto de su versatilidad, hay que comprender el origen de los beneficios que aporta; los que impulsan a muchas corporaciones a decantarse por esta alternativa para sus proyectos Big Data. Todas las ventajas de Hadoop se basan en algunas de sus principales cualidades:

- **Escalabilidad**: esta herramienta permite almacenar y distribuir conjuntos de datos inmensos en sus cientos de servidores que operan en paralelo, permitiendo olvidarse de los límites que otras alternativas imponen.

- **Velocidad**: garantiza una eficiencia de procesamiento que nadie puede igualar.

- **Efectividad en costes**: el almacenamiento de datos se convierte en una realidad para las empresas ya que la inversión necesaria pasa de ser decenas de miles de Euros por Terabyte a quedarse reducida a cientos de Euros por Terabyte.

- **Flexibilidad:** Apache Hadoop se adapta a las necesidades del negocio y le acompaña en su expansión, aportando soluciones reales para cualquier iniciativa que surja.

- **Resistencia al fracaso**: su tolerancia a errores es uno de sus atributos mejor valorados por los usuarios ya que toda la información contenida en cada nodo tiene su réplica en otros nodos del clúster. En caso de producirse un fallo siempre existirá una copia lista para ser usada.

Los dos conceptos en los que se apoya Hadoop son, por un lado, la técnica MapReduce y, por otro, el sistema distribuido de archivos HDFS:

- **HDFS:** Hadoop Distributed File System, es un sistema de archivos distribuido, escalable y portátil.
- **MapReduce:** es el modelo de programación utilizado por Google para dar soporte a la computación paralela. Trabaja sobre grandes colecciones de datos en grupos de computadoras y sobre commodity hardware.

Sin Embargo para entender cómo funciona Hadoop se adentrará más adelante en estos conceptos y se profundizará en las implicaciones que cada uno de ellos tiene en la interacción con Big Data.

#### 4.2.2.3. Arquitectura Hadoop y HDFS

¿Qué es un Clúster Hadoop?, es un conjunto de máquinas que permiten almacenar y procesar información. La estructura básica está compuesta por los Maestros (Master Node) y los Esclavos ( Worker Node), los maestros se encargan de coordinar y los esclavos de trabajar.

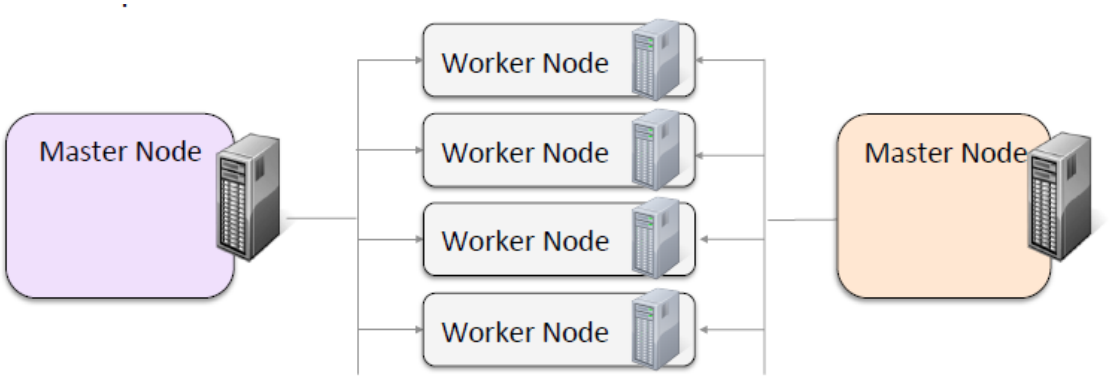



Ilustración 9 - Arquitectura Hadoop y HDFS.

Existen tres estados principales en un clúster:

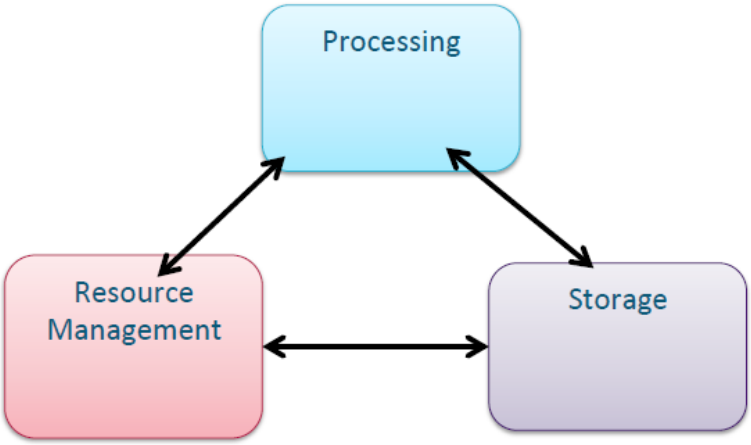



Ilustración 10 - Estados de un clúster.

El clúster:

- Se organiza en racks para disminuir el tráfico de red.
- Está en una red aislada del resto de la red de la compañía.
- La compañía conecta con el clúster a través de los edge nodes.

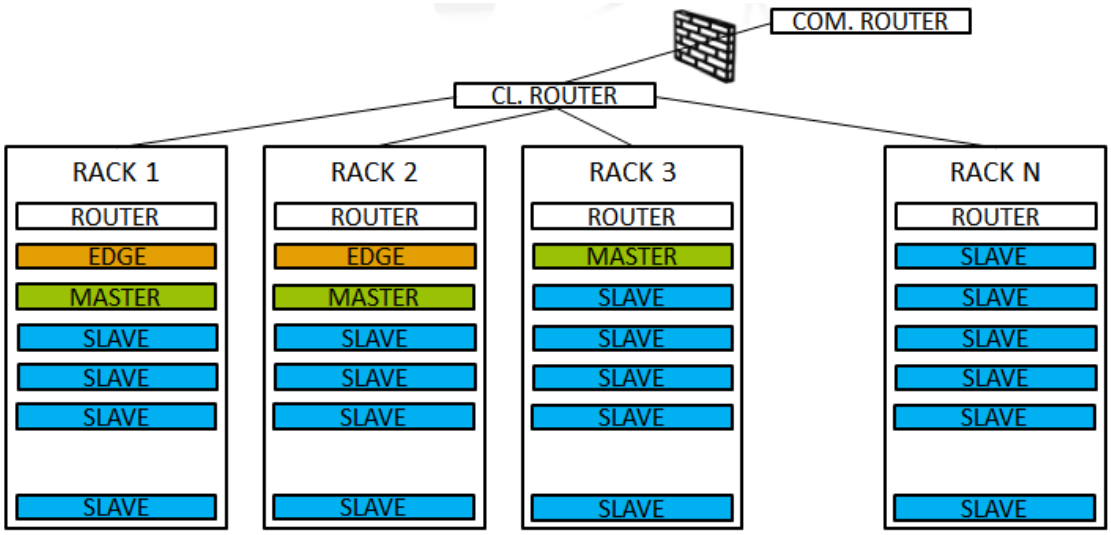



Ilustración 11 – Clúster.

Aspectos técnicos del Hardware:

- **Maestros**
  - Hardware confiable, que permita escalar sin preocupaciones.
  - Tienen que tener dos fuentes de alimentación.
  - Dos tarjetas Ethernet.

- Discos duros en RAID.

- Con una cantidad de RAM que dependa del número de esclavos:
    - Si tiene menos de 20 esclavos =>RAM = 64GB
    - Si tiene menos de 300 esclavos =>RAM = 96GB
    - Si tiene más de 300 esclavos => RAM = 128GB

- **Esclavos**
    - Hardware no especializado.
    - Procesadores:
        - Procesamiento típico 2x6 cores
    - RAM:
        - Entre 64GB y 256GB, dependerá de las tecnologías instaladas
    - Discos:
        - 4 – 18 discos de entre 1-4T.
        - Clúster pequeños : pocos discos grandes.
        - Clúster grandes: muchos discos grandes.
            - Por norma general no superar los 36 TB.
    - Workloads específicos: Coste(procesamiento) >> Coste (acceso a disco)
        - Machine Learning, procesamiento de imágenes, simulaciones complejas.
        - High-end CPU´s y pocos discos pequeños.

Una vez conocido los aspectos técnicos se llega al punto de plantear cuantos esclavos se va a necesitar:

- Lo primero que se necesita saber es una estimación de los datos a almacenar.
- Multiplicar la información por (3+1)/2
    - La información se multiplica por 3 por la replicación
    - Se añade algo de espacio para temporales, multiplicar por 4 en lugar de por 3
    - Los datos suelen estar comprimidos, multiplicar por 2 en lugar de por 4.
- Elegir la configuración de discos.
- Número de esclavos = Size(información) * ((3+1)/2)/ Size(esclavo).

Ya que se sabe calcular el número de esclavos que se va a necesitar, ahora se va a ver con la siguiente tabla el número de maestros que se van a utilizar.

| Cluster size type | # of Worker Nodes | # of Master Nodes | # of Edge Nodes |
|---|---|---|---|
| Mini | 8-16 | 2-3 | 3+ |
| Small | 17-40 | 4-6 | 3+ |
| Medium | 41-120 | 7-9 | 3+ |
| Large | 121-512 | 10-12 | 3+ |
| Jumbo | >512 | Dependes | 3+ |

Ilustración 12 - Tamaño de un Clúster.

Los entornos de desarrollo pueden tener un único maestro.

Por último el número de edge nodes dependerá de la ingesta de datos y del número de usuarios.

**Arquitectura básica de HDFS.**

HDFS es un sistema de ficheros escrito en Java, basado en Google File System(GFS), se asienta sobre un sistema de archivos nativos como puede ser ext3, ext4 o xfs.

Ofrece rendimiento masivo, escalabilidad de rendimiento a medida que se incluyen nuevos discos .

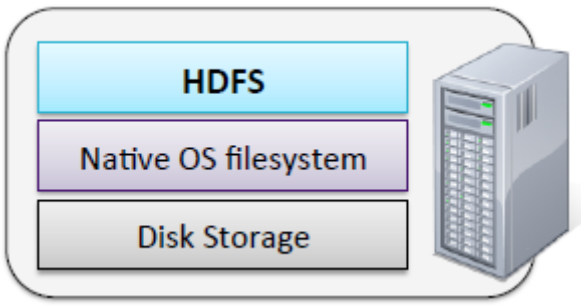



Ilustración 13 - Arquitectura básica de HDFS.

Proporciona almacenamiento redundante para cantidades masivas de datos. Por el contrario HDFS no está preparado para el uso de muchos ficheros pequeños, ni múltiples escrituras simultáneas.

Los archivos se dividen en bloques o trozos contiguos, un bloque es la mínima cantidad de información que puede ser escrita o leída, un bloque en HDFS suele ser de 64 o 128 MB (un fichero que es más pequeño que un bloque no ocupa el tamaño de un bloque entero en el disco). El objetivo de utilizar bloques tan grandes es minimizar el coste de las búsquedas, una vez al principio del bloque el proceso puede leerlo de principio a fin en una sola lectura.

Este sistema de archivos sobre el que se estructura Hadoop cuenta con tres pilares básicos:

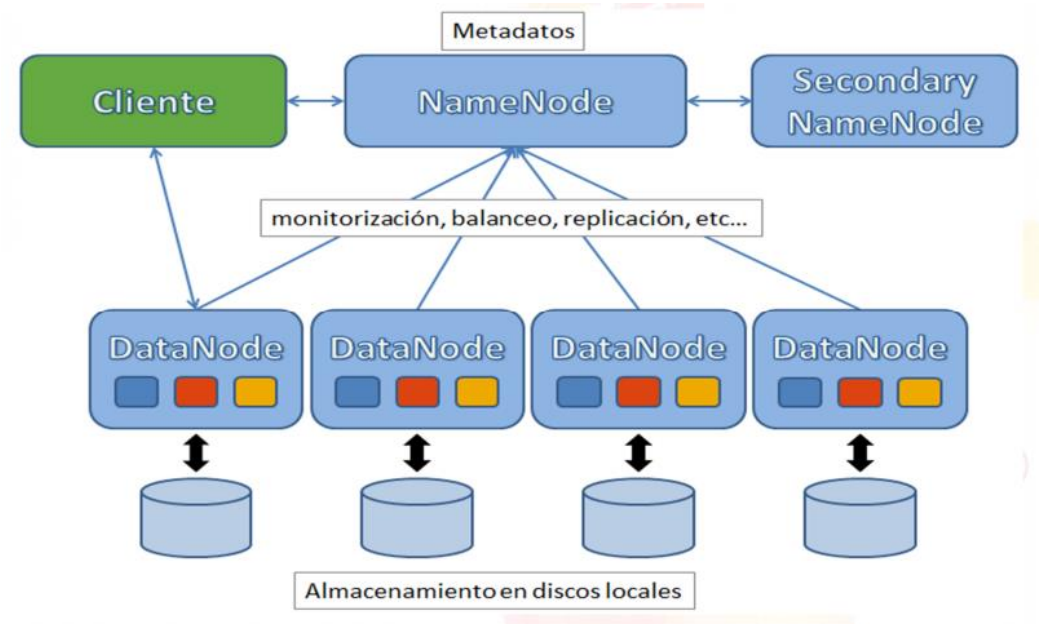



Ilustración 14 - Estructura Hadoop.

- **Namenode:** se ocupa del control de acceso y tiene la información sobre la distribución de datos en el resto de nodos.
    - Gestiona los metadatos de los ficheros.
    - Coordina el acceso desde los clientes a los DataNodes.
    - Monitoriza el estado de los DataNodes.
    - Asegura el nivel de réplica de cada bloque.
    - Mantiene dos estructuras en el disco:
        - Una imagen del estado original del filesystem.
        - Un fichero de log de ediciones.
    - Cada vez que arranca, el NameNode combina ambas estructuras en una nueva imagen del filesystem y la carga en memoria.
        - Gestiona toda esa información en memoria.
    - El NameNode es un posible punto de fallo del sistema.

- Es vital mantener la información de los metadatos segura.
    - El NameNode puede ser configurado para escribir en varios discos el contenido del log de edición.
    - También deben realizarse backups periódicos de los metadatos.

- **SecondaryNameNode:** de forma opcional el sistema puede ejecutar otro demonio llamado SecondaryNameNode, no es un NameNode. Su tarea es realizar de forma periódica la fusión de la imagen del filesystem con el log de edición, de forma que este no crezca demasiado, reduciendo así el tiempo de arranque del NameNode. Al correr en una máquina independiente puede usarse como backup de los metadatos en caso de desastre.

- **Datanode:** son los encargados de ejecutar el cómputo, es decir, las funciones Map y Reduce, sobre los datos almacenados de manera local en cada uno de dichos nodos.
    - Realiza las operaciones de lectura y escritura en comunicación directa con el cliente.

El proceso de lectura de un fichero se realiza mediante una serie de pasos secuenciales en los que intervienen cada uno de los componentes de HDFS:

- El cliente solicita abrir el fichero.
- El NameNode determina si el fichero es accesible y devuelve una lista de los bloques que lo componen.
- El cliente se conecta a un DataNode que contenga el primer bloque y lee.
- El cliente va pasando de DataNode en DataNode a medida que lee los datos.
- El cliente cierra el fichero.

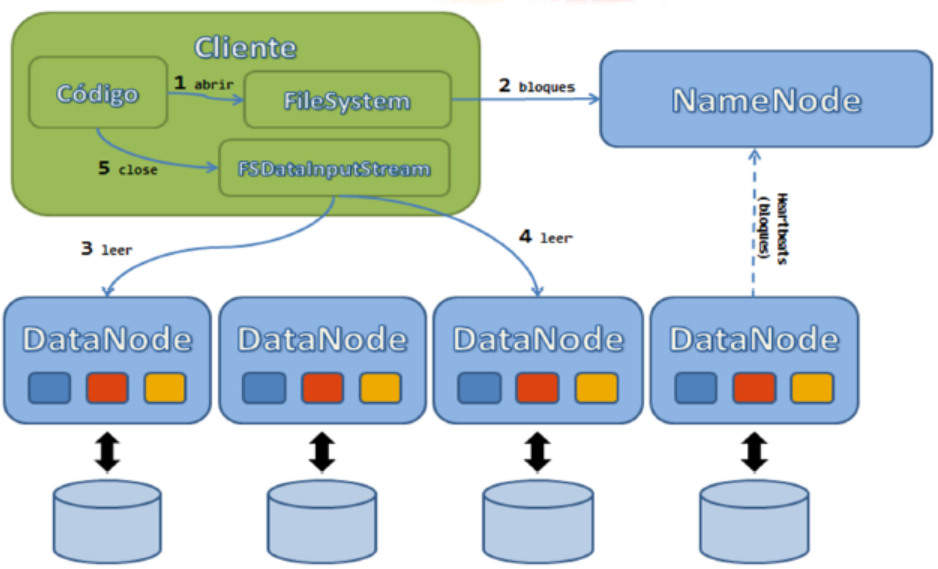



Ilustración 15 - Proceso de lectura.

El proceso de escritura sigue un flujo similar, pero con sus propias necesidades:

- El cliente solicita al NameNode la creación del fichero.
- El NameNode chequea si es posible crear el fichero. Crea la entrada para el fichero.
- El Cliente escribe los datos enviando un paquete de datos al DataNode.
- El DataNode escribe los datos en el disco y le pasa los datos a otros DataNodes para su replicación.
- Cuando todos los DataNodes responden sobre la réplica el primero devuelve una confirmación al cliente.
- El cliente cierra el fichero.

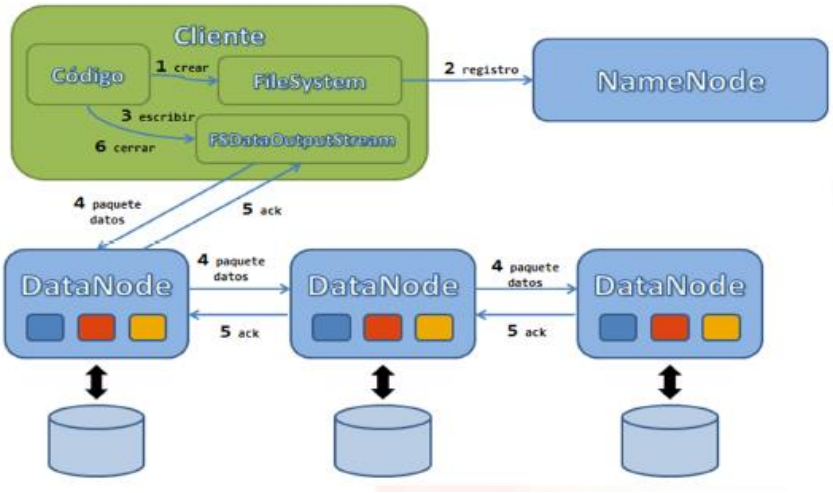



Ilustración 16 - Proceso de escritura.

A continuación se muestra de manera  gráfica como sería el proceso de almacenar y recuperar un fichero:

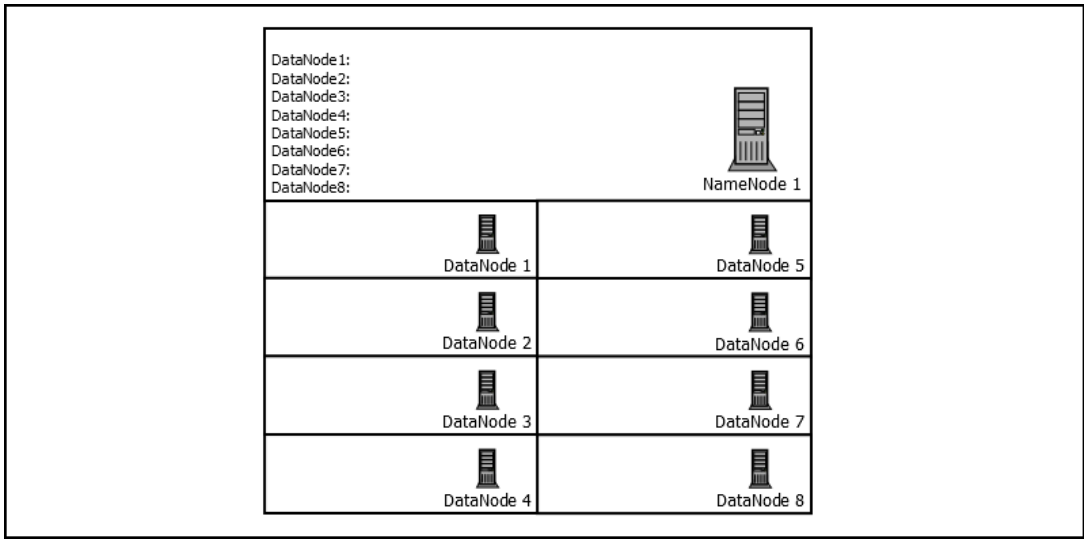



Ilustración 17 - Proceso de almacenar y recuperar un fichero.

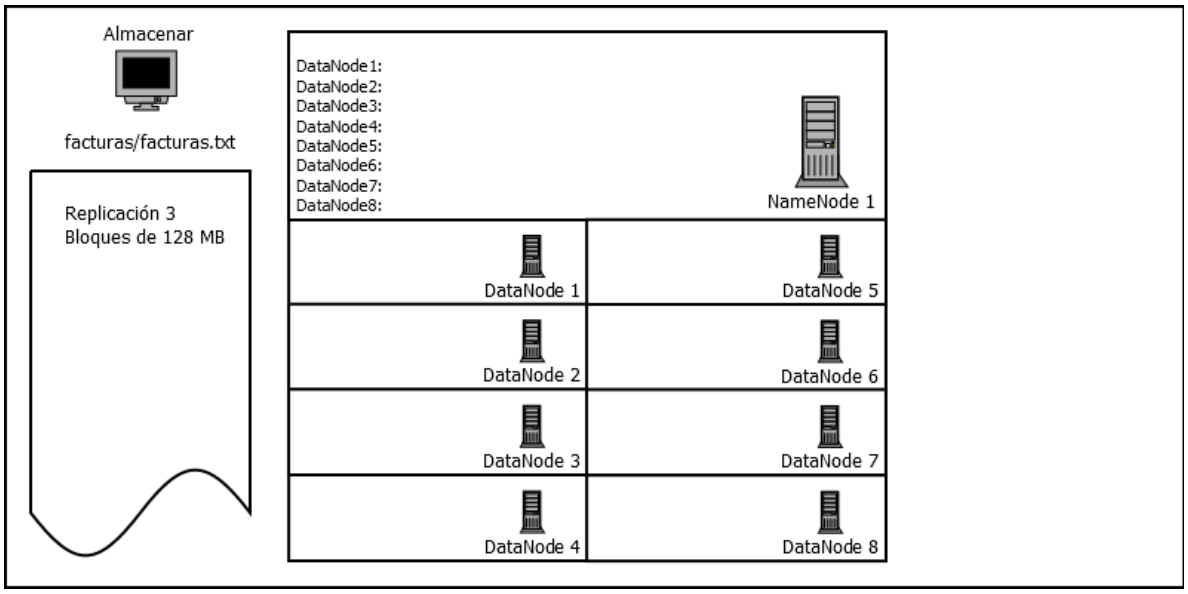



Ilustración 18 - Proceso de almacenar y recuperar un fichero.

Como se puede ver en la imagen se va a almacenar en HDFS, un fichero llamado facturas.txt, con replicación 3, y en bloques de 128 MB.

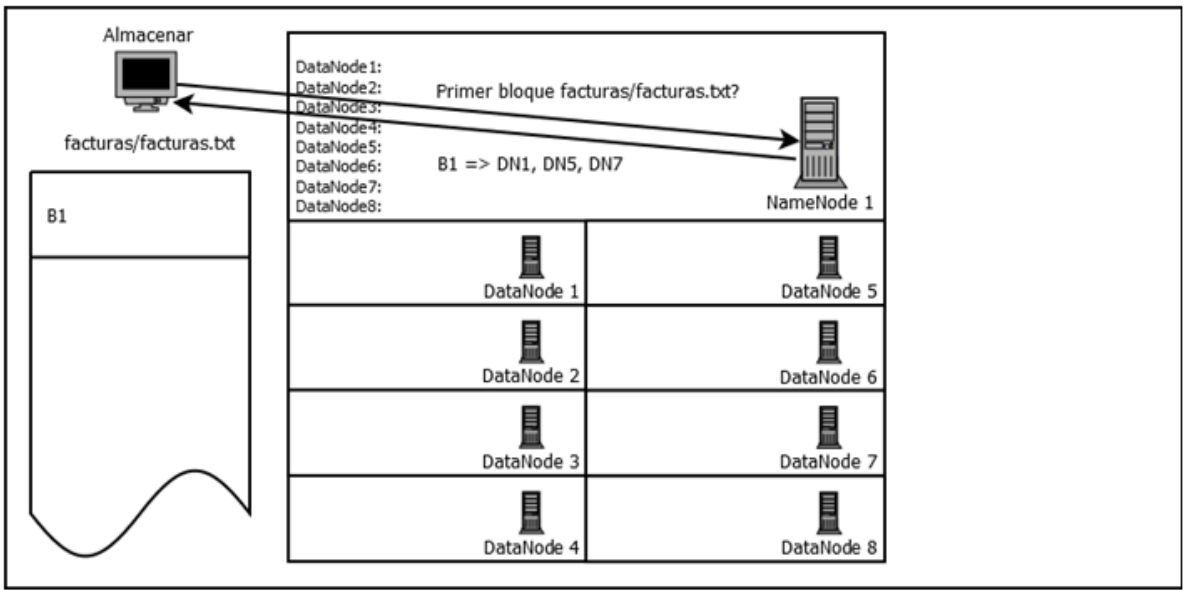



Ilustración 19 - Proceso de almacenar y recuperar un fichero.

Como se puede observar, el cliente se conecta con el NameNode, y le dice que va a introducir un fichero, el NameNode le va a responder que ese fichero al ser menor o igual de 128MB lo va a particionar solo en un bloque( en este caso B1 ), y como se quería replicar 3 veces, el NameNode le va a responder B1 => DN1,DN5,DN7 lo cual significa que el bloque 1, se va a encontrar en el DataNode1, DataNode5 y DataNode7.

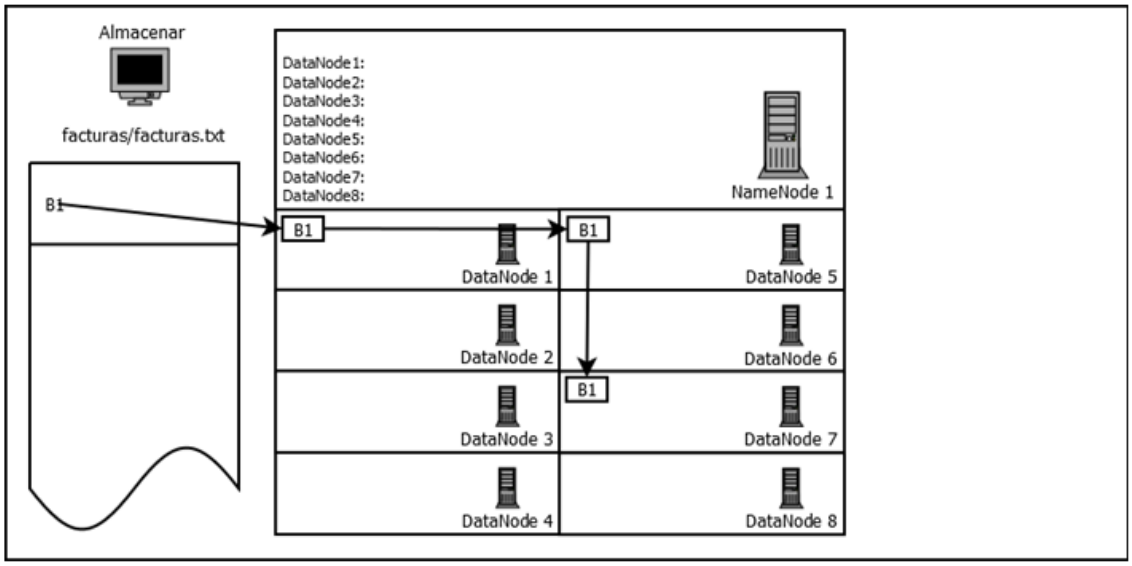



Ilustración 20 - Proceso de almacenar y recuperar un fichero.

El bloque 1 es replicado en los DataNodes.

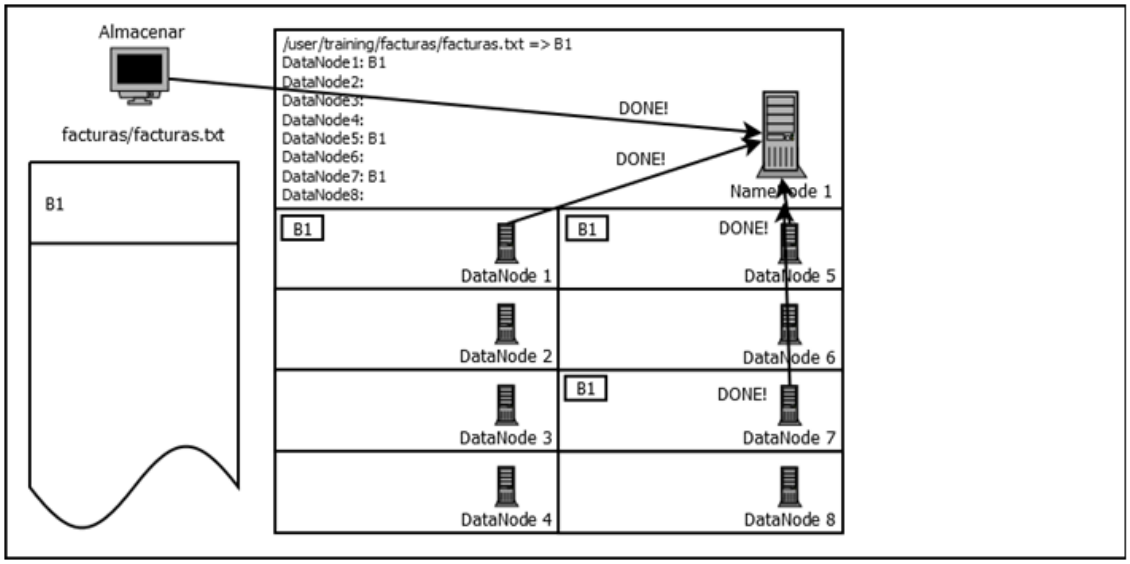



Ilustración 21 - Proceso de almacenar y recuperar un fichero.

El bloque 1 queda replicado y el NameNode recibe la confirmación de en qué DataNodes se encuentra el B1( facturas.txt).

Ahora se va a recuperar facturas.txt que en este caso se va a encontrar en cuatro bloques, con replicación 3, quedando así :

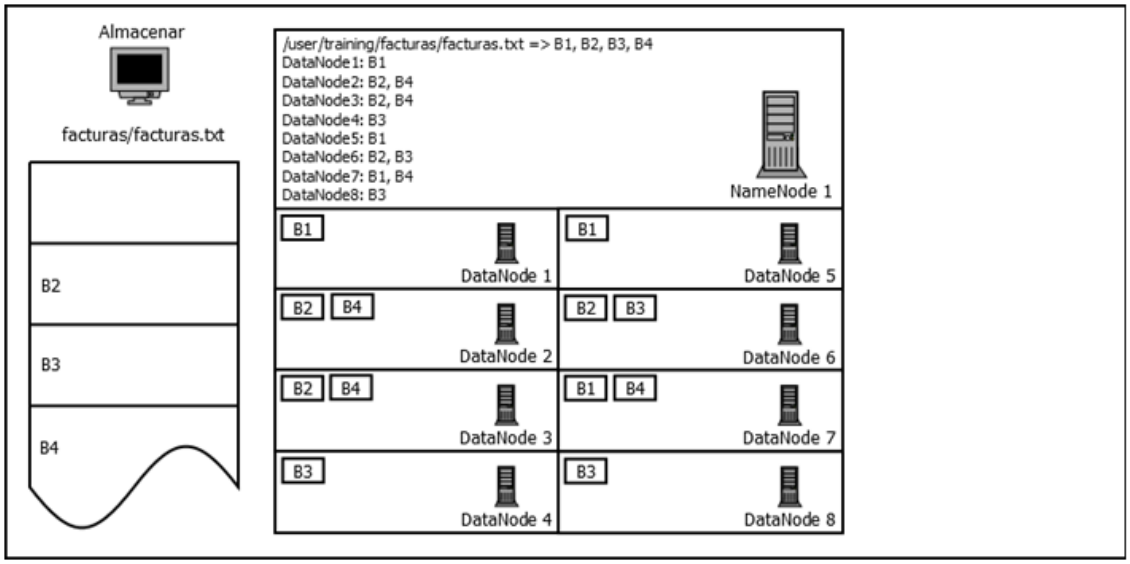



Ilustración 22 - Proceso de almacenar y recuperar un fichero.

El cliente quiere recuperar facturas.txt

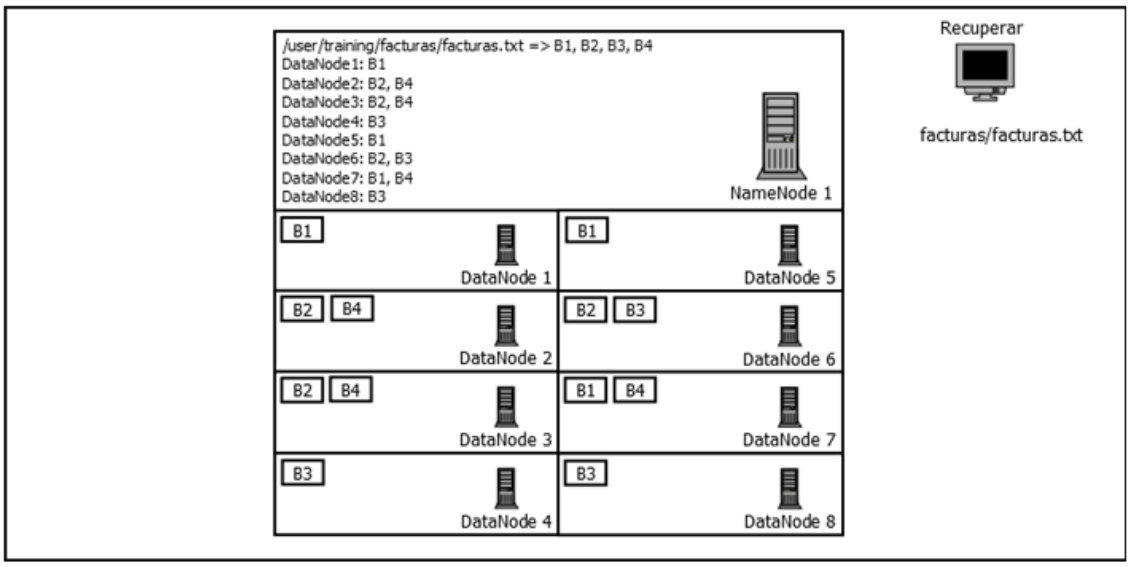



Ilustración 23 - Proceso de almacenar y recuperar un fichero.

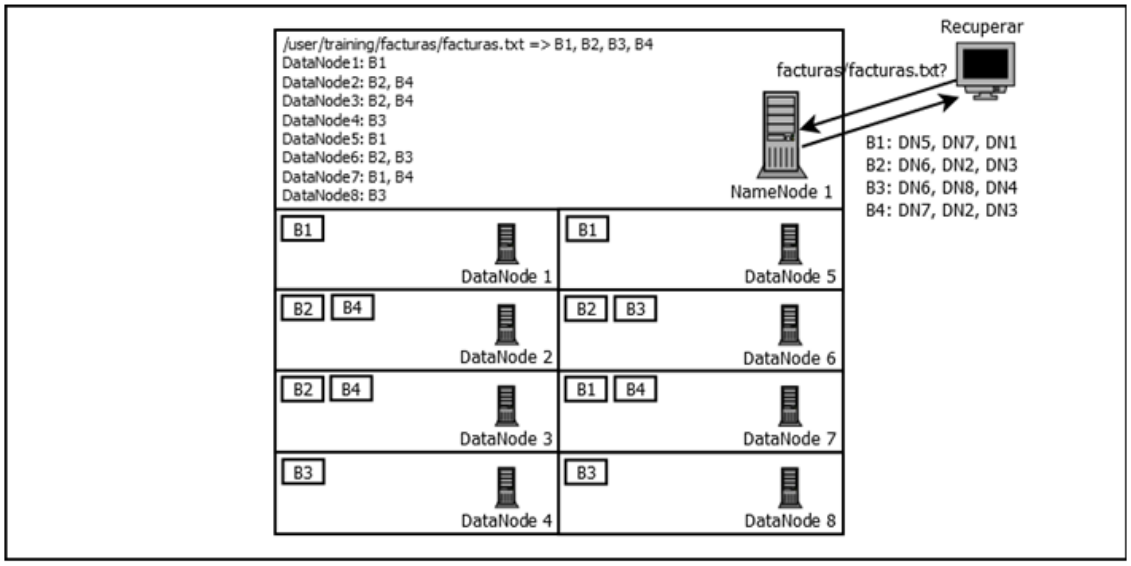



Ilustración 24 - Proceso de almacenar y recuperar un fichero.

El cliente consulta al NameNode donde se encuentra facturas.txt, el NameNode le responde que esta particionado en 4 bloques y en sus respectivos DataNodes.

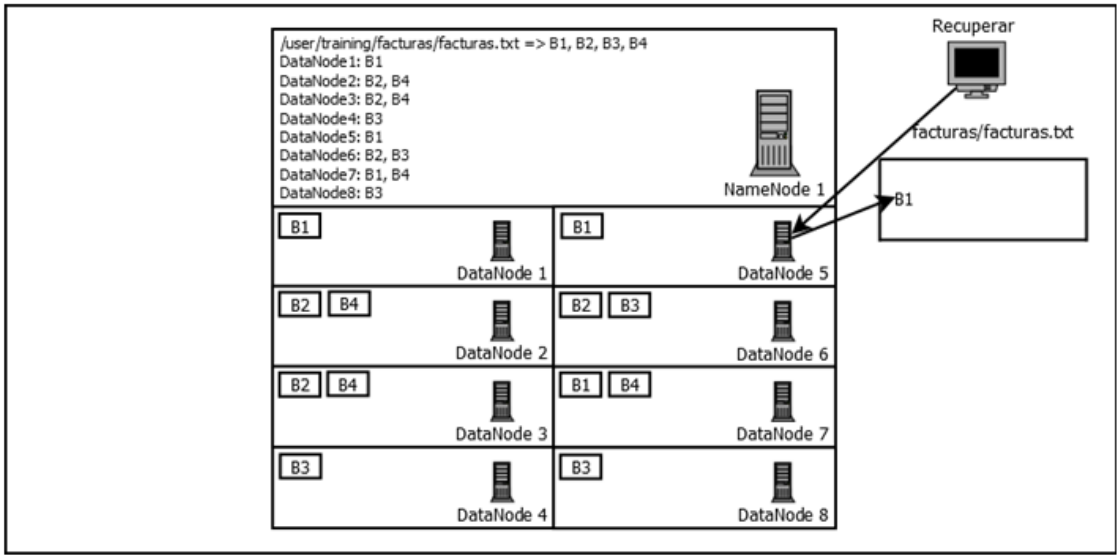



Ilustración 25 - Proceso de almacenar y recuperar un fichero.

El NameNode devuelve al cliente facturas.txt accediendo al DataNode y Bloque que tenga el acceso más rápido, suele valorarlo en función de la latencia. En este caso devuelve el b1 del DataNode5.

### 4.2.3. Arquitectura de procesamiento distribuido

Existen dos modelos de programación:

- Memoria compartida: cada procesador tiene acceso a un conjunto de memoria compartida.
- Memoria distribuida( paso por mensaje ): procesos colocados en diferentes máquinas, se comunican con otros procesos enviando y recibiendo mensajes.

Estos modelos permiten el desarrollo de paralelismo de tareas y datos.

La mayoría de los esfuerzos en paralelización están dentro de las siguientes categorías:

- Paralelizados usando directivas de compilación tal como OpenMP.
- Paralelizados usando librerías de paso por mensaje tal como MPI.
- Paralelizados usando nuevos lenguajes paralelos, como CUDA.

**OpenMP** (memoria compartida)

Es un estándar para programación en memoria compartida, provee un conjunto estándar de directivas, rutinas de librerías run-time, no necesita modificar el código secuencial, solamente indicarle mediante pragmas la realización de operaciones paralelas.

**MPI**( Paso de Mensaje)

Es el estándar para paso por mensaje, no es un nuevo lenguaje, sino una biblioteca que añade lenguajes ya existentes (C, C++, Fortran, Python).

**MPI vs. OpenMP**

Cada una tiene sus ventajas y desventajas.

- OpenMP:
  - Conceptualmente sencillo
  - Memoria compartida.
- MPI:
  - General y Universal.
  - Útil en sistemas distribuidos, mayor escalabilidad.
  - Control muy fino sobre las comunicaciones.

### 4.2.3.1. Modelo de Programación MapReduce

MapReduce es un framework que proporciona un sistema de procesamiento de datos paralelo y distribuido. Su nombre se debe a las funciones principales que son Map y Reduce.

MapReduce surge en un escenario de paralelismo de datos anterior al Big Data y hasta cloud, se obtuvieron los primeros resultados en 2003, MPI ya provee funciones Reduce, viene de la idea divide y vencerás.

MapReduce está pensado para la solución práctica de algunos problemas que pueden ser paralelizados, pero se ha de tener en cuenta que no todos los problemas pueden resolverse eficientemente con MapReduce. MapReduce está orientado a resolver problemas con

conjuntos de datos de gran tamaño, por lo que utiliza el sistema de archivos distribuido HDFS ya mencionado antes.

MapReduce ofrece un servicio eficiente y escalable para la gestión de grandes volúmenes de datos.

El Framework MapReduce tiene una arquitectura maestro / esclavo. Cuenta con un servidor maestro o JobTracker y varios servidores esclavos o TaskTrackers, uno por cada nodo del clúster, con lo que puede funcionar en cualquier entorno ( en comparación con el procesamiento GPGPU con CUDA que requiere una arquitectura específica).

El objetivo final de MapReduce es hacer posible el procesado de grandes volúmenes de datos.

El JobTracker es el punto de interacción entre los usuarios y el framework MapReduce. Los usuarios envían trabajos MapReduce al JobTracker, que los pone en una cola de trabajos pendientes y los ejecuta en el orden de llegada. El JobTracker gestiona la asignación de tareas y delega las tareas a los TaskTrackers. Los TaskTrackers ejecutan tareas bajo la orden del JobTracker y también manejan el movimiento de datos entre la fase Map y Reduce.

Para ver las diferencias entre JobTracker y TaskTracker se van a ver las características de cada uno.

- JobTracker
  - Tiene la capacidad de manejar metadatos de trabajos.
  - Estado de la petición del trabajo.
  - Estado de las tareas que se ejecutan en TaskTracker.
  - Hay un JobTracker por clúster.
  - Recibe peticiones de tareas enviadas por el cliente.
  - Decide sobre la programación.
  - Programa y monitoriza los trabajos MapReduce con TaskTrackers.

- TaskTracker
  - Ejecuta las solicitudes de trabajo de JobTrackers.
  - Obtiene el código que se ejecutará.
  - Aplica la configuración específica del trabajo.
  - Comunicación con el JobTracker.

- Finaliza las tareas, envíos de las salidas, actualización de tareas.

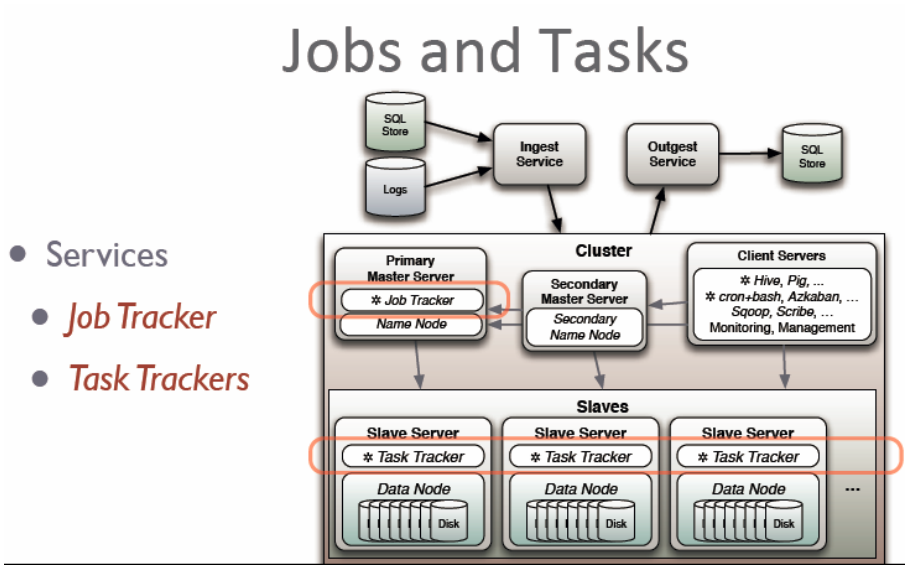



Ilustración 26 – MapReduce.

Aquí se puede observar donde se encuentran el JobTracker y el TaskTracker en un clúster.

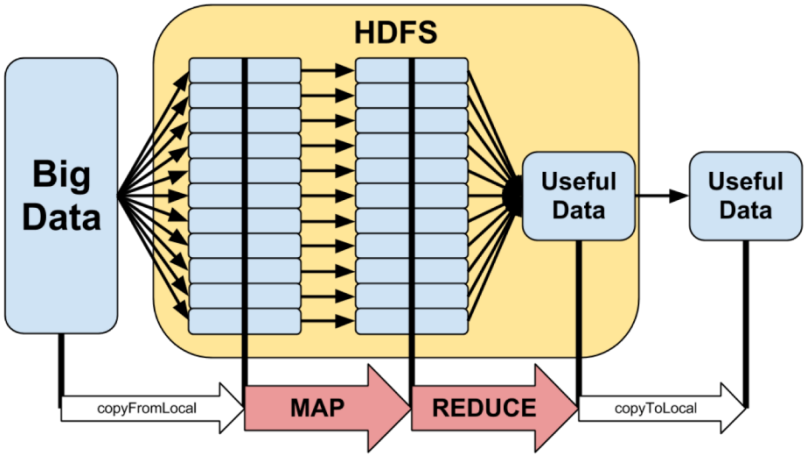



Ilustración 27 – MapReduce.

Y aquí se ve donde se encuentras las funciones Map y Reduce.

Visto la arquitectura de MapReduce, se va a entrar en detalle acerca de cada una las funciones Map y Reduce. Se utilizara MapReduce para abordar problemas que pueden ser resueltos utilizando las operaciones de Map y Reduce, estas funciones están definidas con respecto a datos estructurados en tuplas del tipo (clave, valor).

- **MAP**

La función Map recibe como parámetros un par de (clave, valor) y devuelve una lista de pares. Esta función se encarga del mapeo y se aplica a cada elemento de la entrada de datos, por lo que se obtiene una lista de pares por cada llamada a la función Map. Después se agrupan todos los pares con la misma clave de todas las listas, creando un grupo por cada una de las diferentes claves generadas. No hay requisito de que el tipo de datos para la entrada coincida con la salida y no es necesario que las claves de salida sean únicas.

Map (clave1, valor1) –> lista (clave2, valor2)

La operación de Map se paraleliza, el conjunto de archivos de entrada se divide en varias tareas, el tamaño típico de bloque es de 128MB. Las tareas se distribuyen a los nodos TaskTrackers, y estos a su vez pueden realizar la misma tarea si hiciera falta.


Ilustración 28 – MapReduce.

Los mapper son claramente paralelizables, se ejecutan sobre cada elemento de forma individual, en lugar de procesar un conjunto de elementos atómicos digamos que los mappers procesan pares (k1,v1) y generan otros pares (k2,v2).

- **Reduce**

La función Reduce se aplica en paralelo para cada grupo creado por la función Map().La función Reduce se llama una vez para cada clave única de la salida de la función Map. Junto con esta clave, se pasa una lista de todos los valores asociados con la clave para que pueda realizar alguna fusión para producir un conjunto más pequeño de los valores.

Reduce (clave2, lista(valor2)) –> lista(valor2)

Cuando se inicia la tarea Reduce, la entrada se encuentra dispersa en varios archivos a través de los nodos en las tareas de Map. Los datos obtenidos de la fase Map se

ordenan para que los pares clave-valor sean contiguos, esto hace que la operación Reduce se simplifique ya que el archivo se lee secuencialmente.

Los reducers son difícilmente paralelizables, digamos que los reducers procesan pares de la forma (k2, [v2,v2,v2..]) y generan pares (k3,v3).


Ilustración 29 - MapReduce.

Así se consiguen varios Reducers, uno por cada clave k1, lo cual es paralelizable, como los mappers generan varios valores k, se pueden organizar de la siguiente forma:


Ilustración 30 - MapReduce.

Aquí se puede ver el proceso completo de las funciones Map y Reduce.

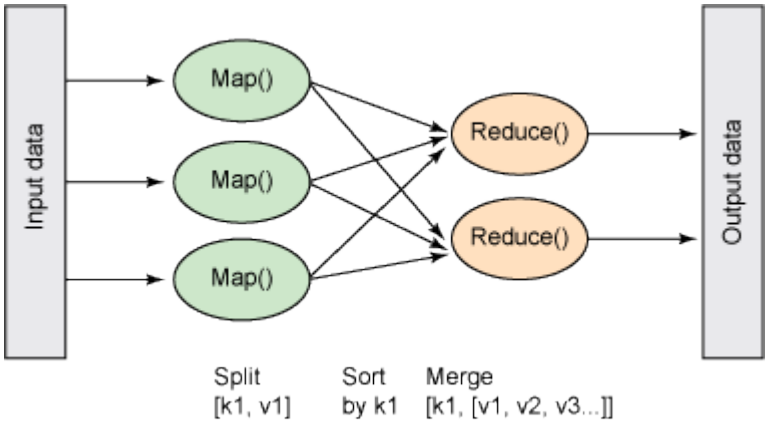



Ilustración 31 - MapReduce.

Pero para entenderlo mejor se va a explicar con un ejemplo.

Se quiere sacar el importe de la facturación diaria de una empresa. Los campos son: código_factura, día_factura, cliente_factura, importe_factura.

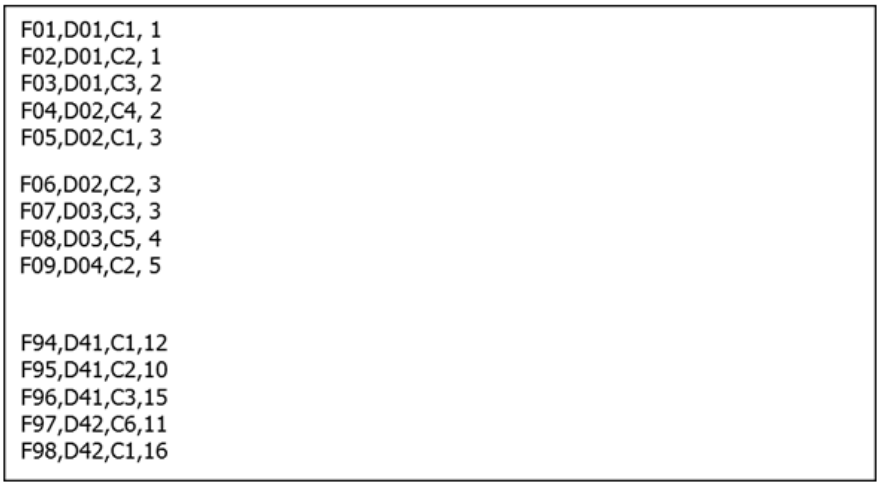



Ilustración 32 - MapReduce.

Como se quiere sacar el importe de la facturación diaria, los campos que van a interesar son día_factura e importe_factura, por lo que se va a utilizar la función Map sobre esos campos utilizando día_factura como clave, y el importe_factura como valor. Ya que esta función opera sobre parte de los datos,  permite filtrar, parsear o transformar la información.

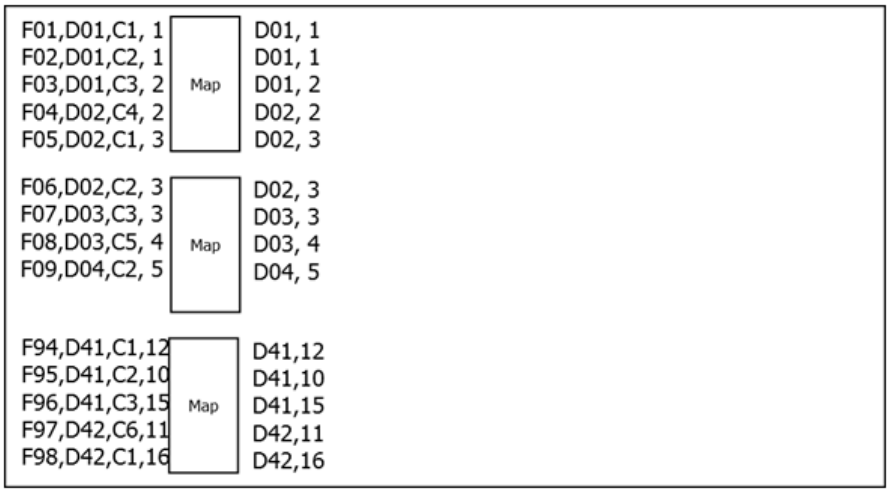



Ilustración 33 - MapReduce.

El resultado del mapeo quedaría así.

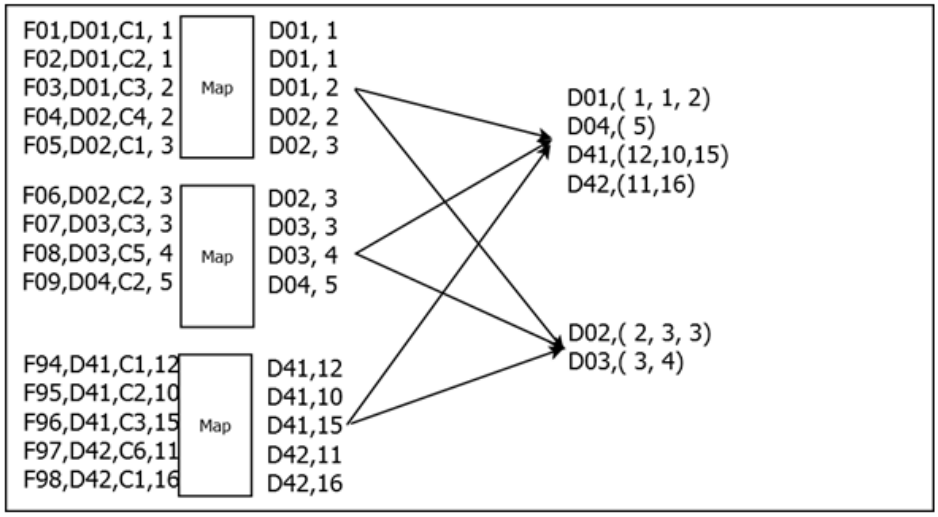



Ilustración 34 - MapReduce.

Como se ha visto anteriormente al utilizar la función Map (clave1, valor1), ha devuelto una lista.

- Map (clave1, valor1) –> lista (clave2, valor2)
- Map(día_factura, importe_factura) –> lista(día_factura, importe_factura)

Lo que se tiene que hacer ahora es un Reduce para que unifique los resultados.

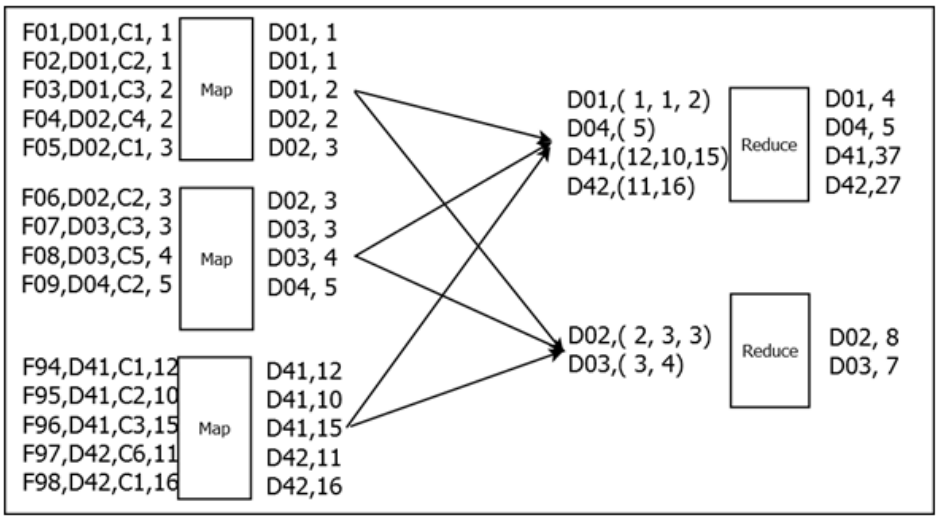



Ilustración 35 - MapReduce.

Como se puede ver con el Reduce, ha unificado los resultados, ha agregado la información.

- Reduce(clave2, lista(valor2)) –> lista(valor2)
- Reduce(día_factura, lista(importe_factura)) –> lista(importe_factura)

A nivel arquitectura, la ejecución en un clúster quedaría así:

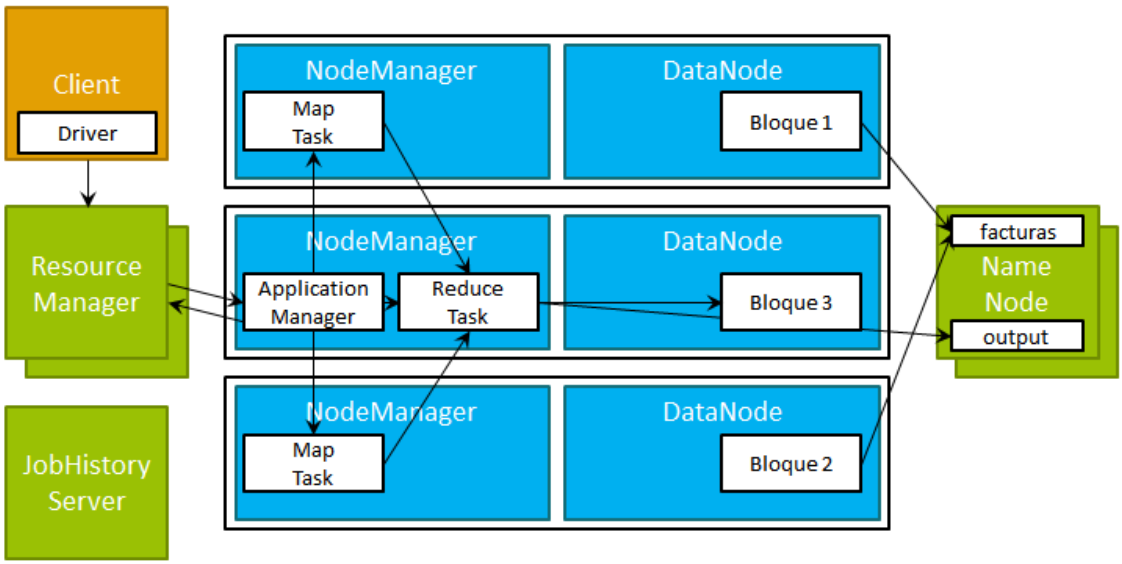



Ilustración 36 – Arquitectura MapReduce.

La información tiene que moverse a través de la red entre los nodos. Leer los datos a través de la red es lento, por lo que en la medida de los posible hay que procurar ejecutar las funciones Map y Reduce en la misma máquina donde se encuentran los datos o al menos lo más cerca posible.

En cualquier caso, se puede producir un envío masivo de datos de mappers a reducers, aunque puede generar un problema en el rendimiento, la salida del mapper genera una gran cantidad de datos intermedios, estos se tienen que transmitir por la red hacia los Reducers.

- Si la cantidad de datos es excesivamente grande aquí se puede producir un cuello de botella.
- Se puede implementar un Combiner, que se ejecuta a la salida de la fase Map y de forma local a éste antes de enviar los datos a través de la red.
  - Si se añade un Combiner, la operación que se realiza en el Combiner tiene que ser asociativa y conmutativa.
  - Ahora la entrada del Reducer no es la salida del mapper si no la del Combiner.

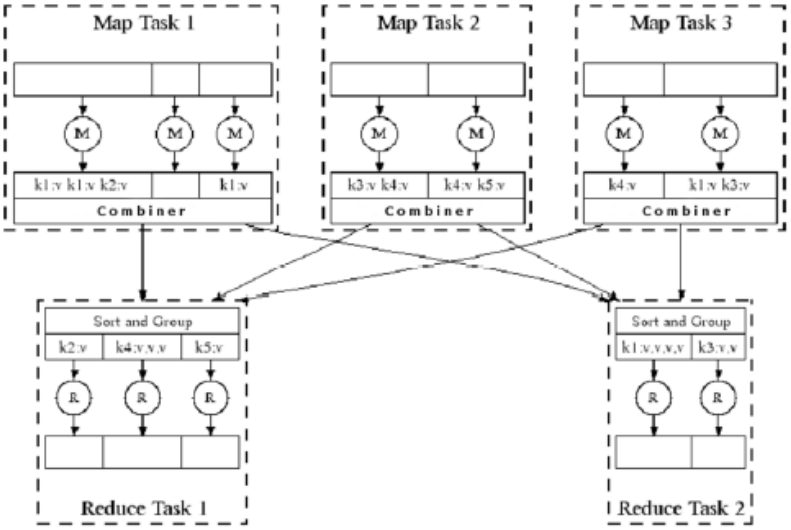



Ilustración 37 - MapReduce.

Para verlo más claro, se utilizará el ejemplo anterior de la facturación diaria utilizando el Combiner.

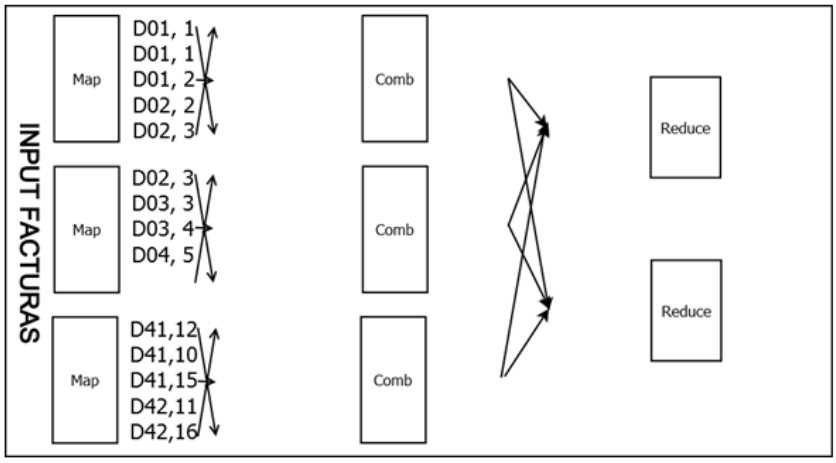



Ilustración 38 - Ejemplo MapReduce.

Primero se realizará un Map sobre los campos día_factura e importe_factura.

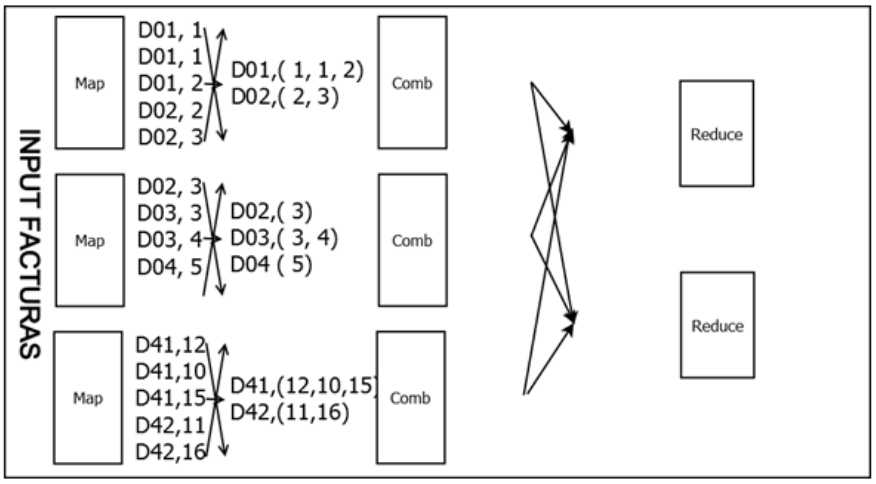



Ilustración 39 - Ejemplo MapReduce.

Ahora se utilizara el Combiner que será la nueva salida para el futuro Reduce.

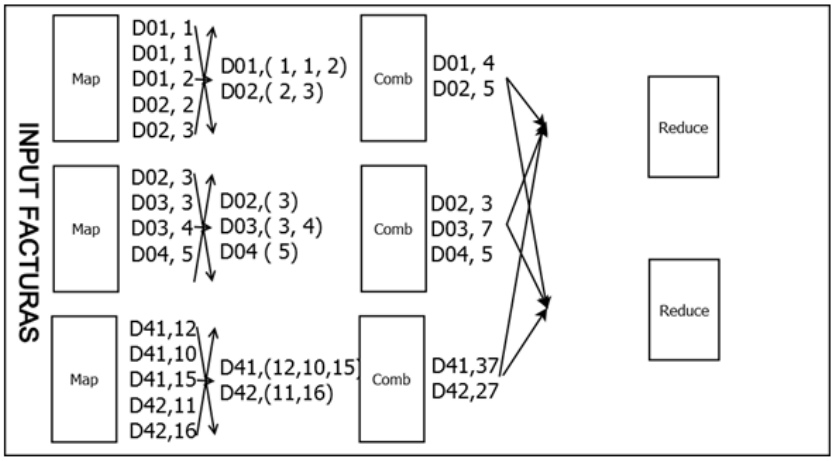



Ilustración 40 - Ejemplo MapReduce.

Se obtiene el resultado del Combiner, se realizará esta operación para mejorar el rendimiento, hay que tener en cuenta que la operación tiene que ser asociativa y conmutativa, se sabe que hay que utilizar esta operación porque la reducción de datos es evidente.

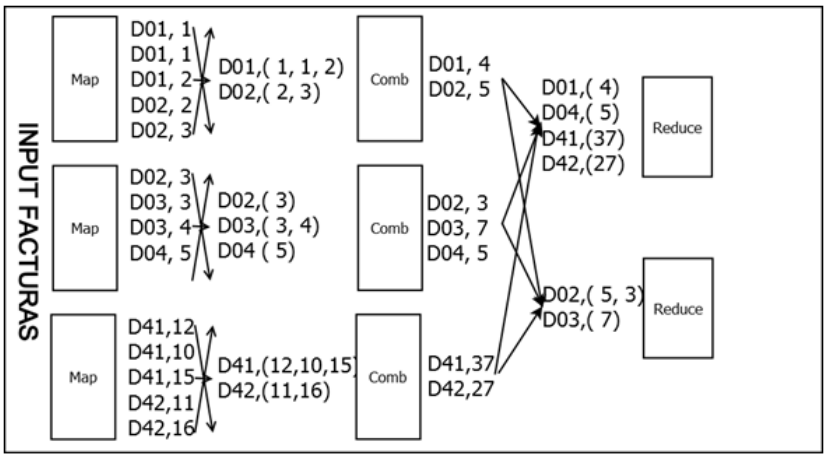



Ilustración 41 - Ejemplo MapReduce.

Y ahora se aplica el Reduce sobre la salida del Combiner .

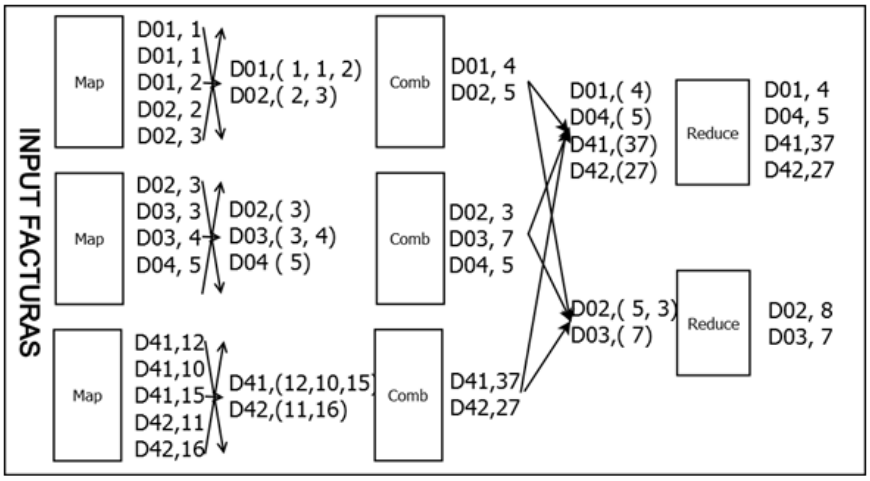



Ilustración 42 - Ejemplo MapReduce.

Para el almacenamiento del resultado se utiliza HDFS, ya que está pensado para la ejecución de aplicaciones MapReduce.

## 4.2.4. Interfaz de Procesamiento Distribuido de Datos

### 4.2.4.1. Apache Hadoop YARN

Hadoop definió una interfaz de facto, inicialmente era un MapReduce sobre HDFS, cuya concepción era batch, hasta que aparece Hadoop 2.0 que ofrece más flexibilidad en HDFS, y diseñan una nueva interfaz a capas superiores :

- YARN : Yet Another Resource Negotiator (2013).

Para entenderlo mejor en la imagen se puede ver cómo funcionaba Hadoop 1.0

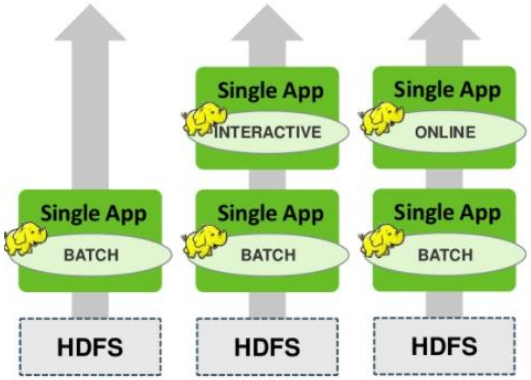



Ilustración 43 - Hadoop 1.0

De tal forma que la arquitectura funcional de MapReduce se mapeaba sobre la arquitectura de HDFS.

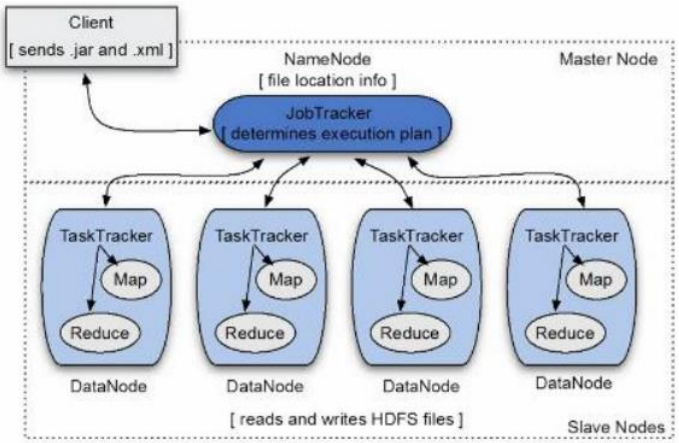



Ilustración 44- Hadoop 1.0

Por lo que Hadoop 1.0 tenía las siguientes limitaciones:

- No había paradigmas alternativos.
  - Todo se esfuerza en parecerse a MapReduce.
  - Las aplicaciones interactivas eran muy lentas .
- Menor escalabilidad:
  - Tiene límites máximos.
- Menor disponibilidad:
  - Un fallo puede matar trabajos en ejecución o en cola.
- Partición de recursos inflexible:
  - No es óptima la utilización de recursos.

Nace la nueva generación Hadoop 2.0, en el que pasa de batch a modelo genérico (batch, online, streaming...)

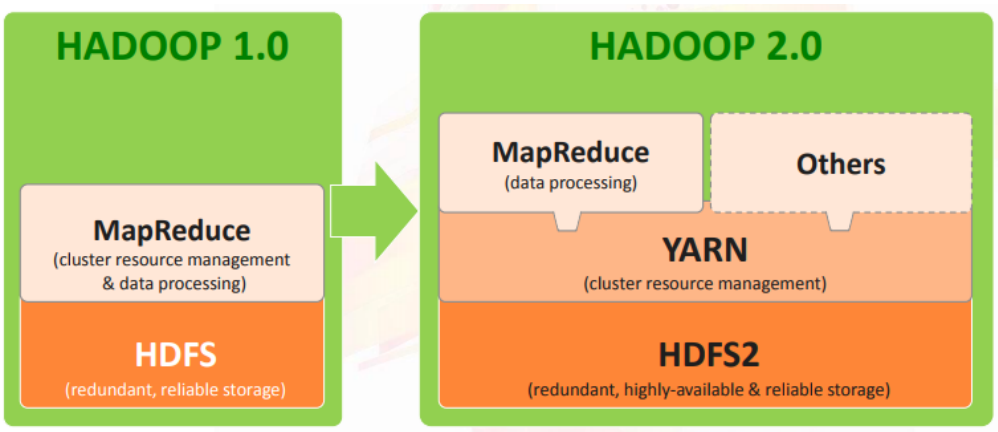



Ilustración 45- Hadoop 1.0 Vs. Hadoop 2.0

Como idea general, Hadoop como nueva plataforma almacena toda la información en un lugar, pero interactúa de múltiples maneras.

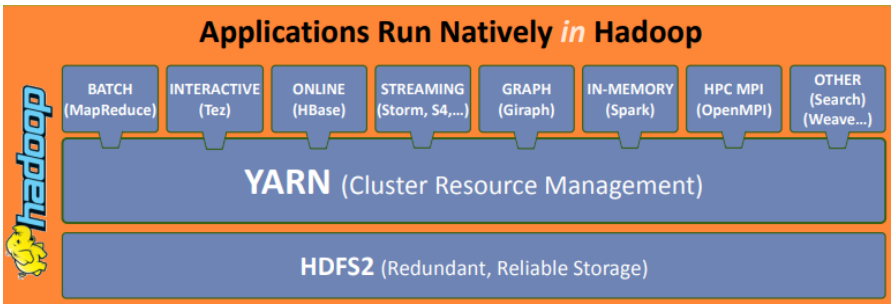



Ilustración 46 - Hadoop 2.0

Elementos de YARN:

- **Resource Manager**
  - Agente central que gestiona recursos del clúster.
- **Node Manager**
  - Agente local (por Nodo) que gestiona la asignación de recursos en el Nodo.
- **Application Master**
  - Gestor por aplicación.
  - Gestiona el ciclo de vida y la planificación de tareas, es decir el Resource Manager más el Application Master asumen por separado las funciones del JobTracker, pero desde una perspectiva más flexible.

Los beneficios de YARN:

- Escalabilidad
  - Diseño con acoplamiento débil.
    - Basado en paso de mensajes y máquinas de estado.
- Agilidad en innovación
  - MapReduce se convierte en una aplicación.
    - Ocupa espacio de memoria de usuario sin privilegios.
  - Múltiples versiones de la misma App pueden coexistir.
    - Lo que permite experimentar con menos riesgo.
  - Nueva versión del marco y la aplicación más sencilla.
- Servicios compartidos
  - Un marco extensible de servicios comunes.
    - Necesarios para sistemas distribuidos genéricos.
    - Las aplicaciones se conectan a ese marco.
  - Sistema de fichero compartido distribuido.
  - Servicio de lectura remota de datos.
  - Servicio de agregación de logs.
- La multi-tenancy viene incorporada de serie
  - Colas, colas jerárquicas, administración.
  - Aislamiento de recursos.

En las imágenes se muestra cómo funciona Yarn para entenderlo de forma más fácil:

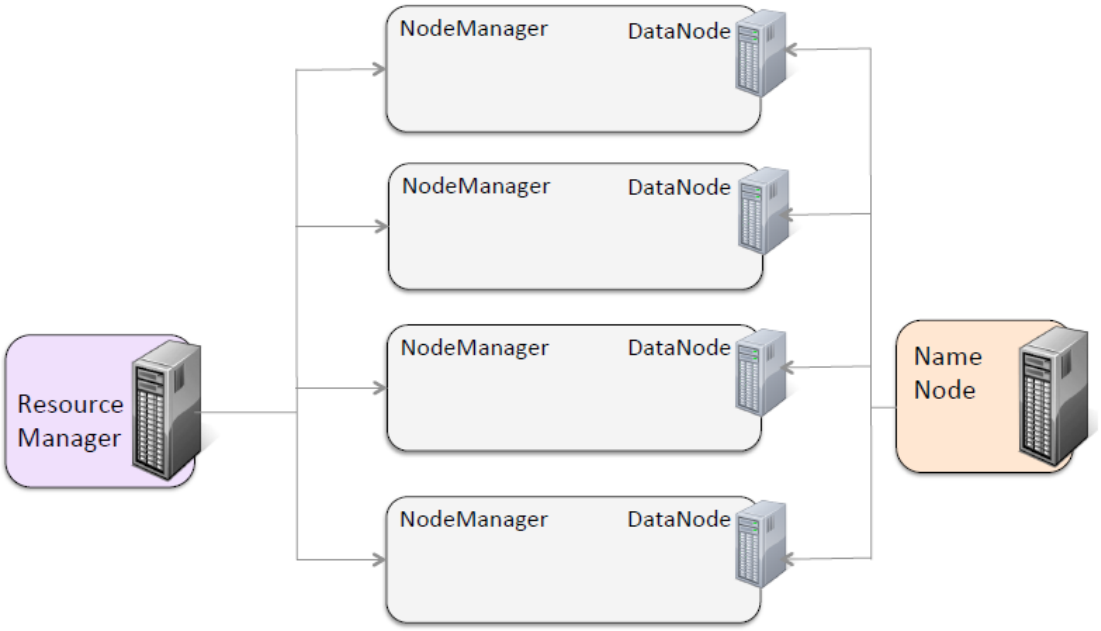



Ilustración 47 - Ejemplo Yarn.

Se sabe que los datos en HDFS se encuentran en el bloque 1 y 2 , de dos DataNodes distintos, como señalan las flechas.

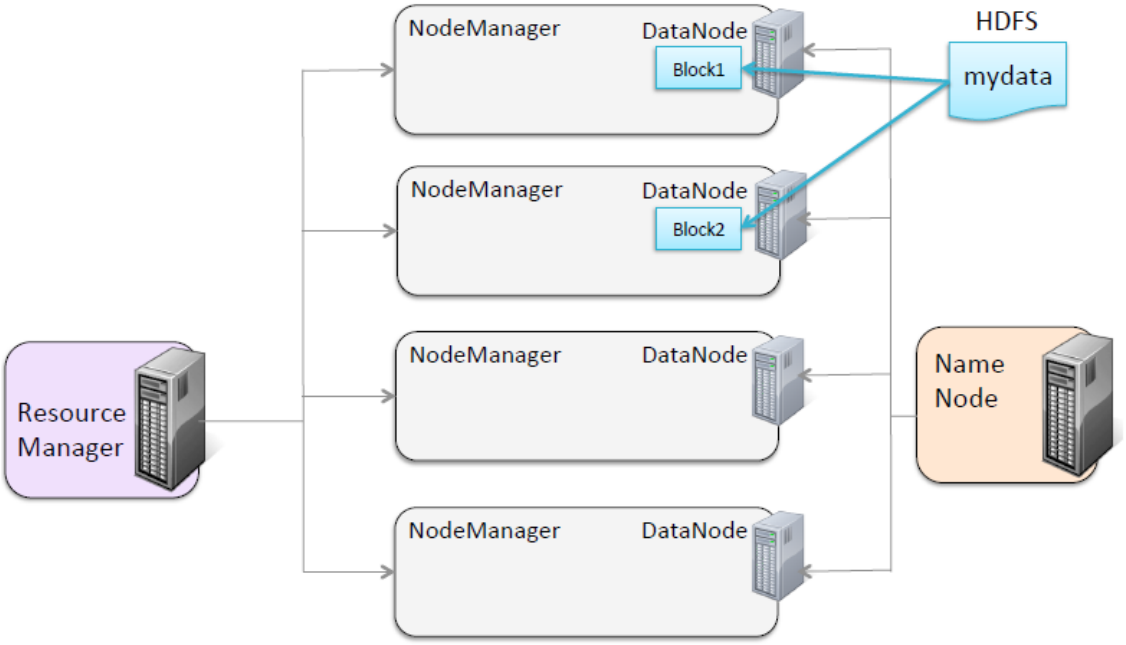



Ilustración 48 - Ejemplo Yarn.

Ahora se quiere saber dónde se encuentran los datos, de tal forma que el cliente establece una conexión con el Resource Manager a su vez activando el Application Master, éste le va a preguntar al Name Node donde se encuentran los datos y éste a su vez le va a responder que están en el bloque 1 y 2 de los dos DataNodes 1 y 2.

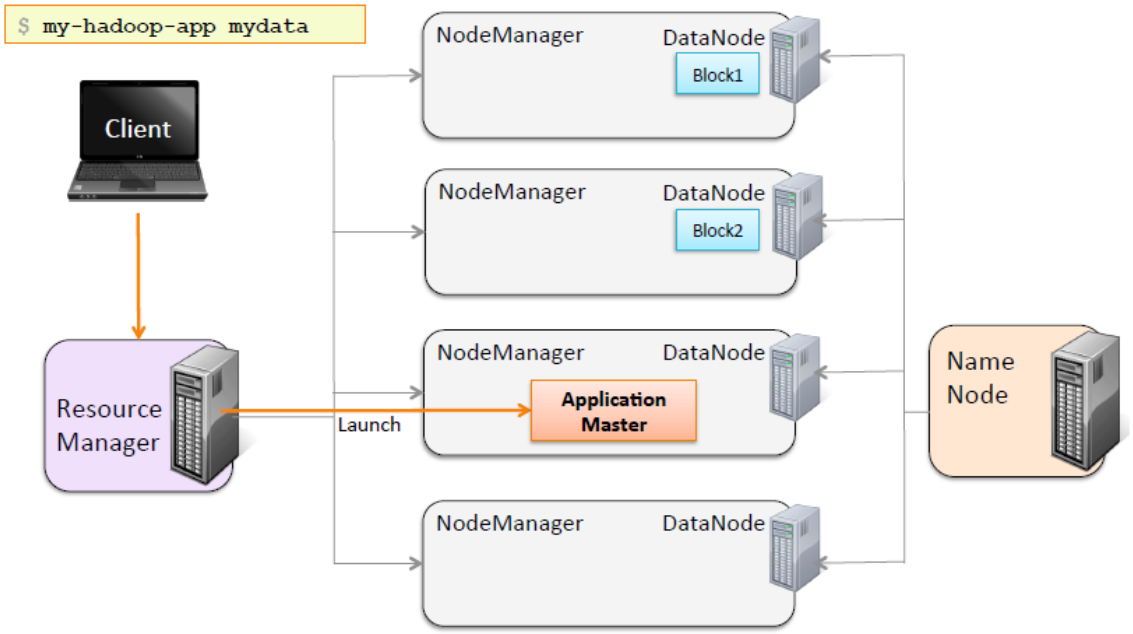



Ilustración 49 - Ejemplo Yarn.

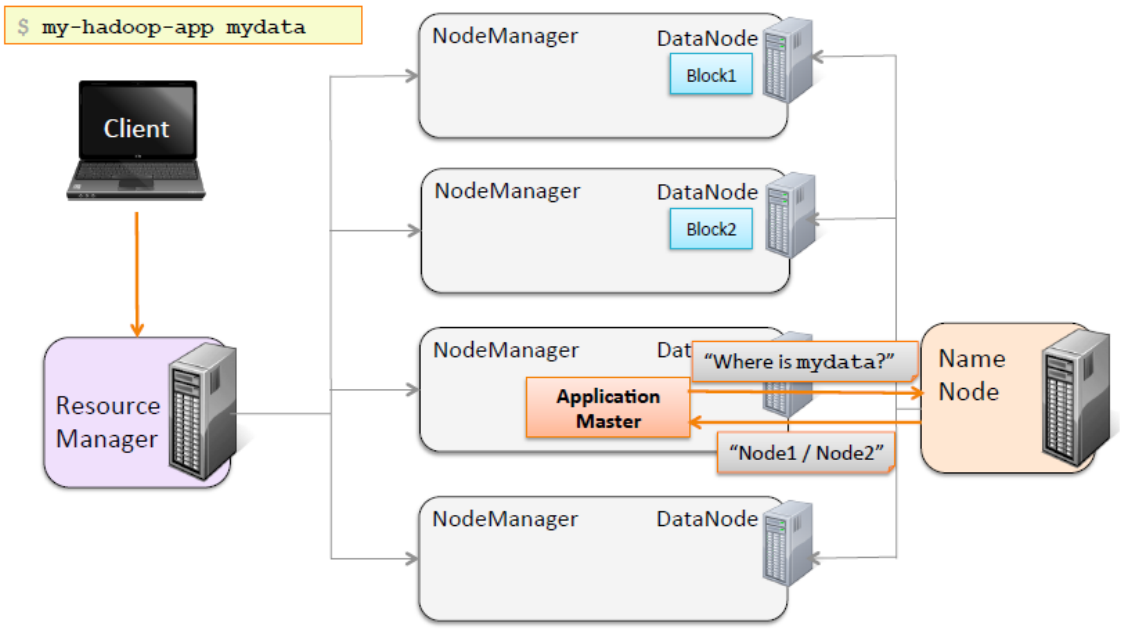



Ilustración 50 - Ejemplo Yarn.

Una vez que se saben dónde están los datos, automáticamente el Application Master le va a decir que necesita reservar una serie de recursos del clúster general al Resource Manager, para repartirlos en los nodos donde se tiene la información a procesar .

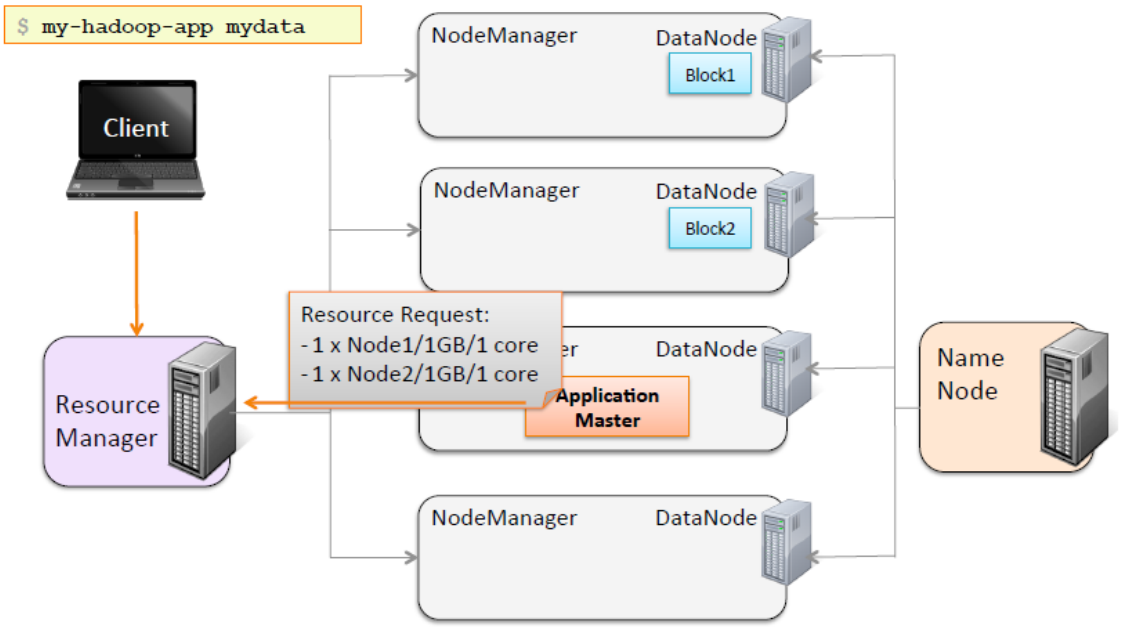



Ilustración 51 - Ejemplo Yarn.

El Resource Manager en función de los recursos que tenga nuestro clúster le va a dar el visto bueno o no, de tal forma que en este caso si le da el visto bueno ( antes de crear un clúster se estudia la memoria de procesamiento y almacenamiento necesaria por lo que siempre el Resource Manager concede la solicitud al Application Master), con lo que el Application Master va a reservar esos recursos en “contenedores” dentro de cada Data Node mientras se realiza el procesamiento de los datos.

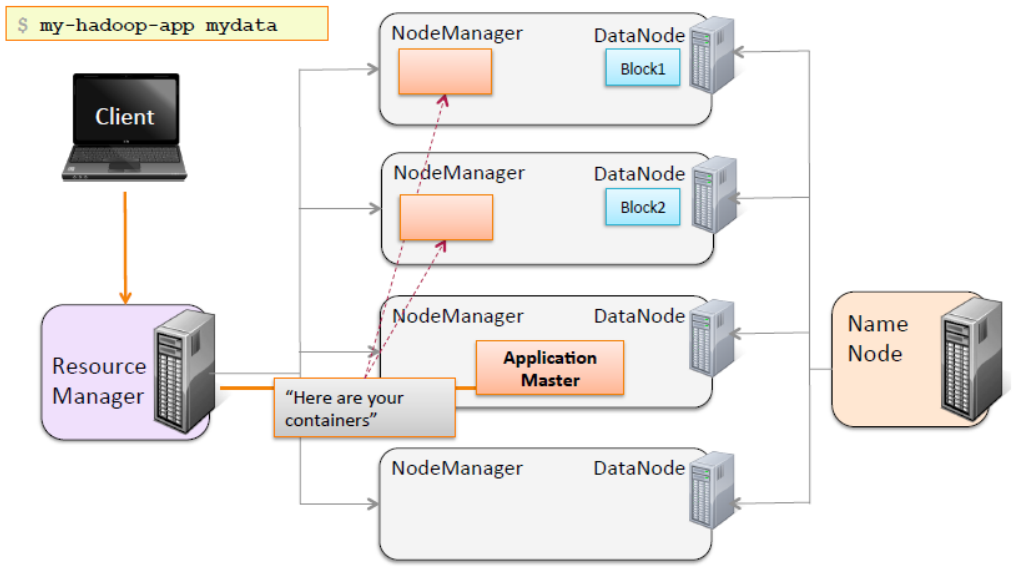



Ilustración 52 - Ejemplo Yarn.

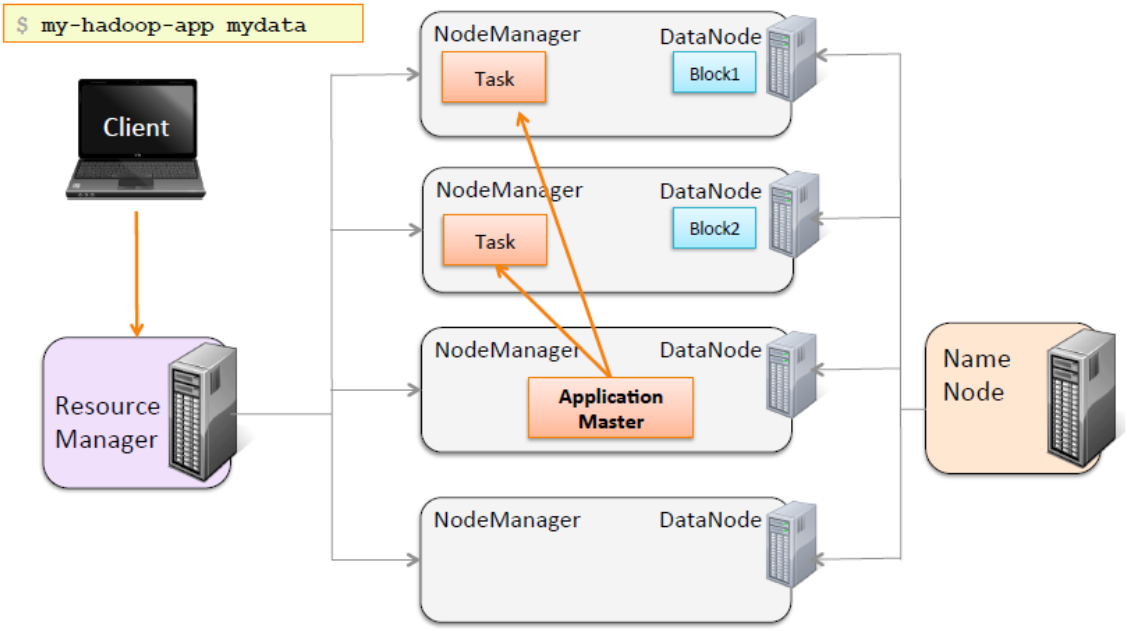



Ilustración 53 - Ejemplo Yarn.

Como se ha comentado una vez reservados los recursos, realiza las tareas, y una vez finalizadas, libera los recursos.

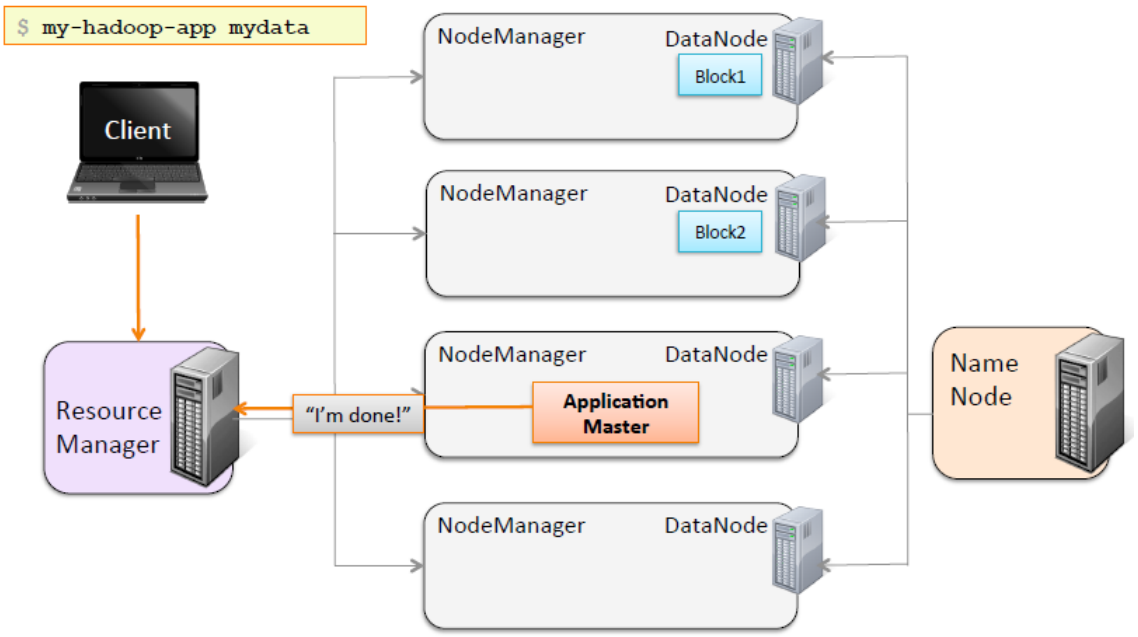



Ilustración 54 - Ejemplo Yarn.

## 4.2.5. Procesamiento Distribuido de Datos

### 4.2.5.1. Apache Spark


Ilustración 55 - Apache Spark.

Tras el hito marcado por la aparición de Apache Hadoop implementando el modelo MapReduce, y dentro del constante desarrollo de nuevas herramientas y plataformas orientadas al procesamiento de Big Data durante estos años, posiblemente la aparición más relevante (por su potencial como "heredero" o sucesor más importante de Hadoop, aunque pueden utilizarse juntos) sea la de Apache Spark, la pieza nuclear de lo que se conoce como la Berkeley Data Analytics Stack (BDAS).

Apache Spark es un framework de procesamiento en modo código abierto desarrollado en 2009 en el AMPLab de la Universidad de California en Berkeley como el núcleo del proyecto de investigación doctoral de Matei Zaharia dirigido por Ion Stoica. El framework fue donado a la Apache Software Foundation en 2013.

El objetivo con el que se diseña Spark es proporcionar una mejor alternativa para el procesamiento de datos masivos ante determinados casos de uso en los que MapReduce no se mostraba del todo eficiente. La alternativa se apoya en el uso de memoria principal en vez de almacenamiento en disco (gracias al abaratamiento de la memoria y por tanto a su disponibilidad en grandes cantidades en los clústers de máquinas utilizados para procesar Big Data). Disponer de memoria donde almacenar datos masivos evita el problema de lentitud que surge al tener que leer y escribir grandes cantidades de datos en disco.

Spark permite realizar de manera rápida análisis de datos y modelos de desarrollo. Spark permite acceder al conjunto de datos evitando la necesidad de realizar submuestras tal y como hacen entornos tipo R. También da soporte a trabajar en streaming lo que permite construir modelos en tiempo real con todo el conjunto de datasets. Si la tarea es demasiado grande para un único servidor, la arquitectura permite dividir la tarea en piezas más manejables según las capacidades de hardware que se disponga en cada momento.

Los casos de uso ante los que se van a encontrar  son principalmente dos:

- Procesos iterativos, que deben ir procesando repetidamente los datos en sucesivas iteraciones. Una aproximación basada en MapReduce requeriría almacenar los datos en disco tras cada iteración y volver a leerlos en la siguiente. Con el uso de memoria para ese almacenamiento entre iteraciones el proceso completo se acelera enormemente.

- Procesos intensivos en consultas, donde hay muchas interacciones para obtener datos, lo que en un escenario con MapReduce requeriría volver a leer los datos de disco cada vez que se lanza la consulta. Si los datos una vez procesados se dejan disponibles en memoria, las consultas que accedan a ellos podrán resolverse más rápido.

Para implementar este esquema, Spark maneja una abstracción de los datos llamada Resilient Distributed Datasets (RDD), colecciones de datos que están segmentados y repartidos a lo largo de un clúster de máquinas y que podrán estar almacenados tanto en disco como en memoria. Así, las tareas de procesamiento de datos masivos que se programan sobre Apache Spark se construyen como una combinación de operaciones sobre estos RDD. Spark dispone de un amplio conjunto de dichas operaciones que se dividen en transformaciones y acciones, con la característica clave de que las transformaciones (map, filter, distinct, …) construyen una cadena de procesamientos en los datos que no son efectivamente ejecutados hasta que se lanza una acción (reduce, take, collect, …) que consolida el resultado de las transformaciones anteriores.

## Problemas a resolver con Spark:

- Memoria compartida vs. Memoria distribuida.
    - Permitir escalado de operaciones en miles de nodos sin perder eficiencia.
- Único paradigma que existía previamente: MapReduce.
    - Poco flexible.
    - Código demasiado complejo.
    - Muy lento.
- Mejor tolerancia a fallos, transparente.
- Gestión más flexible de recursos.

## Mejoras al usar Spark:

- Aumenta rendimiento en cálculos iterativos:
    - Se pueden almacenar resultados intermedios en memoria (caché).
    - Evita contante trasiego de datos al disco.
- Mayor variedad de operaciones para transformar y analizar datos.
- Análisis y control de flujo de operaciones.
- Mantiene integración con Hadoop:
    - Input: Avro, Parquet, S3, CSV …
    - NoSQL: Cassandra, HBase…
    - Spark Streaming: Kafka, Flume…

    - Spark SQL: HIVE…
    - Gestión de recursos: YARN …
- Patrones de programación genéricos:
    - Mismo motor funciona en múltiples plataformas: multi-core o clúster.
    - Misma abstracciones de datos ( RDD, DataFrame) optimizadas en múltiples lenguajes: Scala, Python, R…
- Creación ligera de procesos ( no consume muchos recursos).
- Reparto de datos entre nodos más eficiente:
    - Se podrá evitar excesivo intercambio de datos entre nodos al realizar ciertas operaciones.

## Inconvenientes al usar Spark:

- Todavía es una arquitectura joven, que sufre cambios importantes y drásticos con frecuencia.
    - Suele existir un compromiso de compatibilidad hacia atrás.
    - Sin embargo, extraer toda la potencia de nuevas funcionalidades obliga a actualizar el código.
- Muchas funciones disponibles solo en Scala.
- Integración con R aún en estado preliminar.
    - Solo hay un proceso R.

## Entornos de Programación en Spark:

- Scala ( es el lenguaje nativo de Spark ).
- Java.
- SQL.
- Python.
- R ( todavía preliminar ).
- Con la arquitectura actual, es más fácil poder integrar nuevos lenguajes de programación en el futuro.

Una vez vistos las ventajas e inconvenientes, se va a ver como se realiza el procesamiento de datos en Memoria:

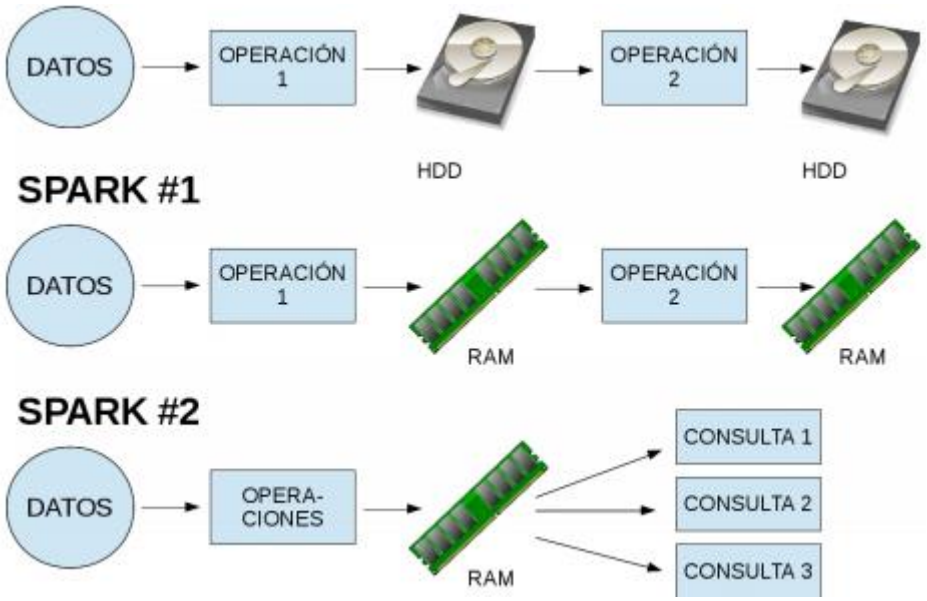



Ilustración 56 - Apache Spark.

Si se pueden almacenar resultados intermedios o datos ya procesados en memoria, se puede acelerar mucho (x10 – x100) accesos posteriores. Se encuentra ante el paradigma de la programación funcional, en la que se definen operaciones ( en una función ) y se aplican en todos los datos ( en varias cores o varios nodos). Un dato importante que se debe tener en cuenta es que se debe garantizar que hay suficiente espacio en memoria para almacenar los datos, de lo contrario puede saltar un error que mate el proceso, por eso como se comentó antes a la hora de explicar YARN , antes de diseñar el clúster se debe saber que memoria se va a necesitar para realizar nuestros procesos.

Actualmente las principales plataformas cloud que permiten desplegar clústeres con Spark son:

- Amazon AWS : ofrece un soporte nativo para desplegar nodos con Spark, la distribución de Spark proporciona un script para lanzar y parar clústeres EC2 con Apache Spark.
- Otras plataformas : Microsoft Azure, OpenStack etc.

## El Stack tecnológico de Spark.

Una Visión simplificada:

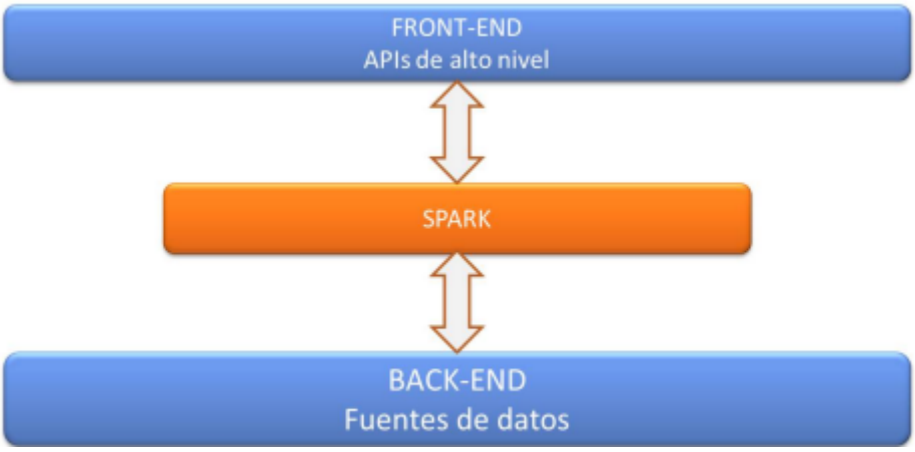



Ilustración 57 - Stack tecnológico de Spark.

Visión detallada:

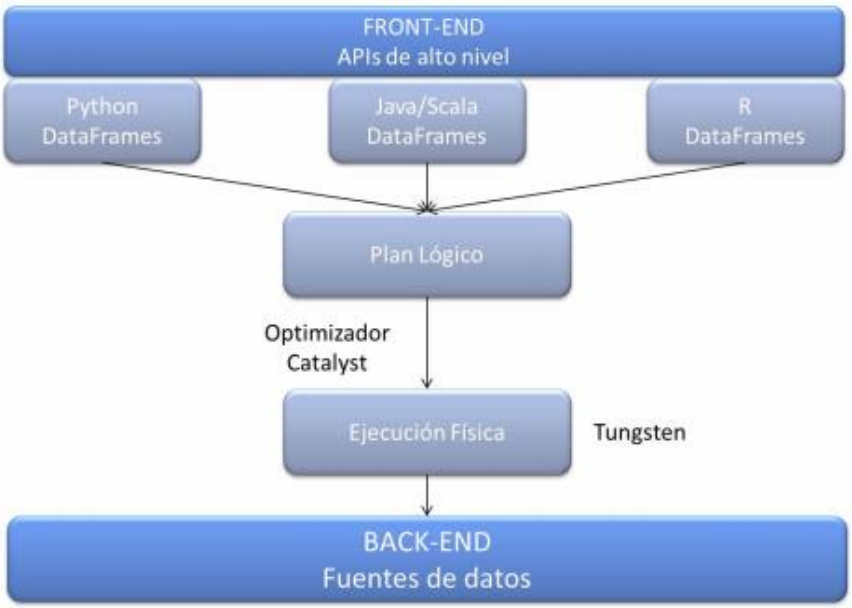



Ilustración 58 - Stack tecnológico de Spark.

Componentes:

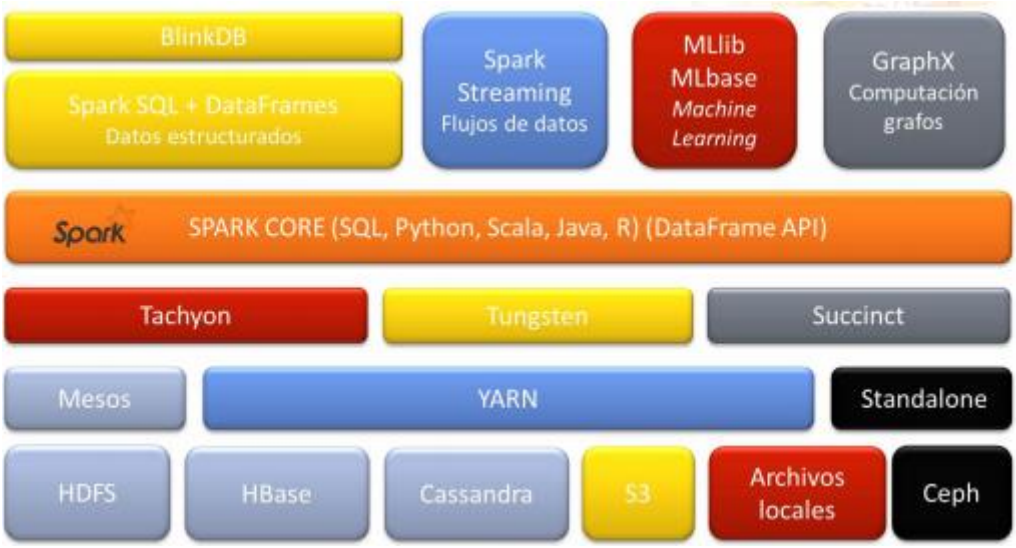



Ilustración 59 - Componentes de Spark.

## Spark Core

- RDD´s
  - Abstracción de datos que permite distribuir un gran número de datos entre los nodos de un clúster, particionando el conjunto original de datos en bloques que se envian a los nodos de trabajo.
  - Existe duplicación para garantizar la recuperación en caso de fallo.
  - En el Stack actual, permite la realización de operaciones con los datos al más bajo nivel posible para el programador.
  - Control fino de operaciones, gestión de memoria y como se particionan y distribuyen los datos entre los nodos.

- Data Frames
  - Abstracción de datos introducida en Spark 1.3.
  - A partir de Spark 1.6 se convierte en abstracción principal para operaciones unificadas en cualquiera de los lenguajes soportados por los componentes del front-end.
  - Optimización a nivel lógico de las operaciones.
  - Optimización a nivel físico de la implementación de las operaciones.

- En teoría, garantiza rendimiento parejo en cualquiera de los lenguajes y entornos soportados.

- Spark SQL
  - Comenzó como una API para ejecutar consultas SQL sobre datos estructurados en SPARK.
  - Soporta SQL estándar y Hive Query Lenguage.
  - Soporta tablas Hive, Parquet y JSON.
  - Permite mezclar SQL con las operaciones sobre datos en los programas Spark en Scala, Java o Python.
  - Actualmente, es el núcleo central de la implementación DataFrames, la nueva abstracción para datos estructurados.

- Catalyst
  - Optimizador a nivel lógico.
  - Gestiona operaciones realizadas a través de SparkSQL o mediante DataFrames.
  - Facilita poder extenderlo con nuevas optimizaciones:
    - Operaciones especiales.
    - Datos semi-estructurados.
    - Analítica avanzada de datos.
  - Basado en árboles ( Trees ) sobre los que se aplican reglas (Rules) para manipularlos.
  - Análisis -> Optimización -> Generación de código.

- Tungsten
  - Optimizador de planes para implementación física de las operaciones.
  - Similar a Catalyst pero a bajo nivel.
    - Gestión más optimizado de huella en memoria.
    - Aprovechamiento de las ventajas de las diferentes tecnologías de gestión de datos en el back-end.
    - Facilita la extensión, por ejemplo para añadir nuevas fuentes de datos.
  - Objetivos importante:
    - Evitar al recolector de basura de JVM.

- Explotar las capacidades de la jerarquía de memoria.
- Aprovechar juegos de instrucciones optimizadas en CPUs modernas.

- Spark Streaming
  - Análisis de datos streaming.
  - Admite diversas fuentes de datos, tales como Kafka o Flume.
  - Operaciones muy similares a las que se pueden realizar con Spark Core ( RDDs).
  - Proporciona la misma tolerancia a fallos, fiabilidad y escalabilidad que Spark Core.

- MLlib
  - Biblioteca de algoritmos de machine learning.
  - Algunas aplicaciones de regresión, clasificación, clústering, filtrado colaborativo…
  - Desde junio de 2014, integra aportaciones que provienen del antiguo proyecto Apache Mahout ( ahora obsoleto ). Esfuerzo de traducción al entorno Spark.
  - Muy lejos todavía del gran repertorio de alternativas disponibles en R, a través de paquetes CRAN.
  - Se puede trabajar ya tanto con RDDs como con DataFrames.

- GraphX
  - Tratamiento y análisis avanzado de datos sobre grafos.
  - Escala a un número arbitrario de nodos, vértices y cualquier combinación de características descriptivas sobre estos.
  - Operaciones fundamentales sobre grafos más Pregel ( arquitectura para procesamiento de grafos desarrollada por Google).
  - Solo está disponible en Scala.

### 4.2.5.2. Programación en Spark

- **Programación con RDDs : Resilient Distributed Datasets**
  - Abstracción principal a bajo nivel para trabajo con datos en Spark.
  - Son inmutables: no pueden cambiar una vez construidos.
  - Permiten paralelizar operaciones en múltiples particiones (bloques) de datos.
  - Permiten construir y trazar el lineage(DAG) de las operaciones para optimización.
  - Se reconstruyen automáticamente ante fallos para su recuperación.
  - Formas de creación:
    - A partir de ficheros o directorios de datos (locales o distribuidos).
    - A partir de colecciones en Python.
    - Aplicando una transformación a otra RDD.

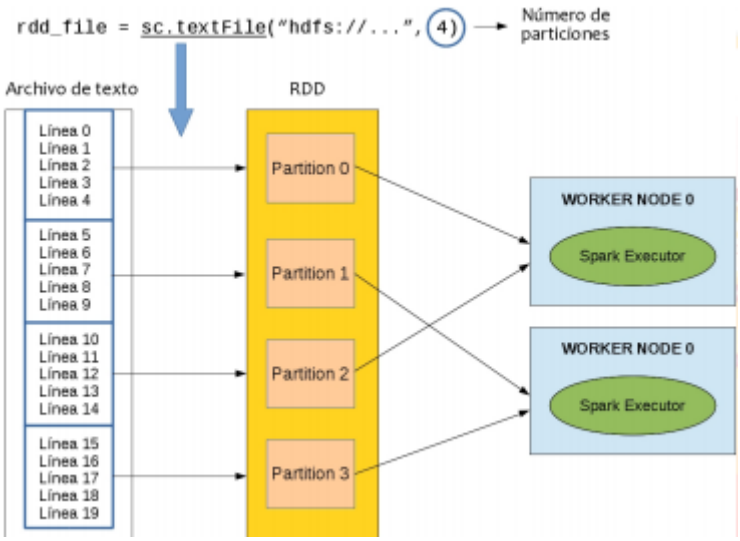



Ilustración 60 - Programación con RDDs.

- Se puede controlar el número de particiones que se fabrican.
- Solución de compromiso: mayor nivel de paralelismo vs. Número de cores y memoria de los nodos.
- En archivos de texto la unidad mínima de procesamiento de datos es una línea -> Particiones de N líneas.
- La distribución de datos entre los nodos dependerá de cómo estén almacenados en el sistema de ficheros distribuidos subyacentes ( HDFS)

- Se verá que hay operaciones que son partition- aware: saben en qué partición reside cada fragmento de los datos para minimizar transferencia de datos entre nodos.

- **Key-Value RDDs**
  - Un RDD estándar contiene una colección de elementos ( líneas de texto ).
  - Pero un RDD también puede contener pares clave-valor -> Key-Value RDDs.
  - La clave debe ser un objeto hashable.
  - Permiten operaciones específicas tales como reducir por clave ( reduceByKey) o JOIN entre elementos de los RDDs ( como en sets de datos).

- **Estructura interna: RDDs**

Internamente, un RDD se caracteriza por cinco propiedades:

- Una lista de particiones.
- Una función para computar sobre cada Split ( partición).
- Una lista de dependencias de otros RDDs.
- Opcional: un particionador para Key-Value RDDs.
- Opcional: lista de ubicaciones preferentes para computar cada partición ( por ejemplo la ubicación de bloques para un archivo HDFS ).

- **Ciclo de ejecución de un RDD**

Existen dos tipos de operaciones que se pueden aplicar a los RDDs, una son las transformaciones, las cuales aplican una operación sobre un RDD de entrada para generar otro RDD como salida, y el otro tipo de operación son las acciones, operaciones que devuelven resultados al programa driver o los salvan de forma permanente.

Las cinco propiedades del ciclo de un RDD:

- Crear RDD inicial ( de fuente de datos o paralelizando una colección de elementos Python).
- Realizar transformaciones para ir creando RDDs que apliquen funciones a los datos distribuidos.

- Cuando sea conveniente, guardar resultados intermedios en memoria (cache) para acelerar cálculos ( evitar computar nuevamente datos).

- Ejecutar una acción para poner en marcha la computación paralela, optimizada y realizada por Spark.

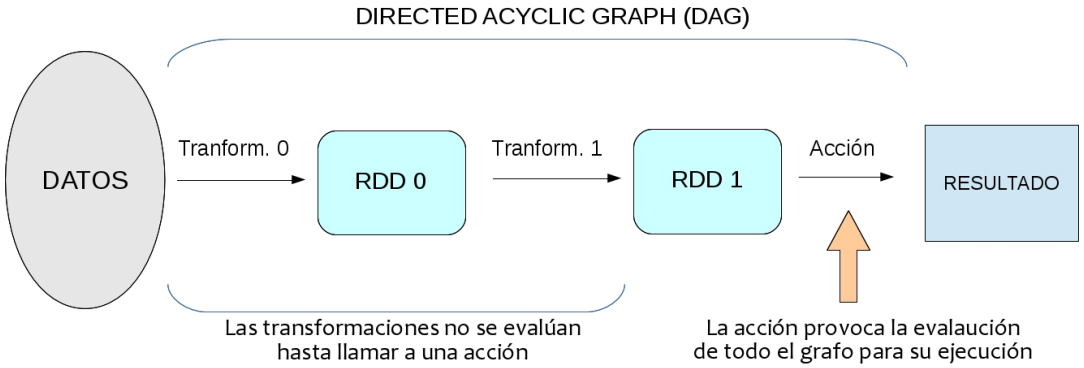



Ilustración 61- Programación con RDDs.

- **Transformaciones de un RDD**

  - Operaciones que se realizan sobre un RDD para generar otro RDD.
  - Lazy evaluation: solo se computa cuando se aplica finalmente una acción para obtener resultados.
  - Mientras solo se apliquen transformaciones nada se ejecuta aún. Se está construyendo la cadena de operaciones que se va a aplicar a los datos.
  - Cuando finalmente se llama a una acción, Spark analiza el grafo de transformaciones completo para optimizar su implementación.
  - Es conveniente usar persistencia de datos en memoria si existen resultados intermedios que se va a utilizar en pasos sucesivos o con frecuencia.
  - Si hay fallos en nodos reinicia trabajos ( por defecto hasta 4 veces como máximo).
  - Lista de transformaciones:

| Nombre | Función | Descripción |
|---|---|---|
| Mapeo | map(función) | Crear nuevo RDD aplicando función a cada elemento del RDD de entrada |
| Mapeo plano | flatMap(función) | Como el anterior pero devuelve una colección, ya que cada elemento de entrada se puede mapear a 0 o varios elementos de salida. |
| Filtrado | filter(función) | Crear nuevo RDD quedándonos solo con los elementos de entrada para los que función devuelve True. |
| Valores distintos | distinct(n Task) | Crear nuevo RDD solo con el conjunto de valores distintos que aparecen en la entrada. |

- Lista de transformaciones ( KEY- Value RDDs):

| Nombre | Función | Descripción |
|---|---|---|
| Reducir por clave (V,V)->V | reduceByKey(func) | Agrega pares (K,V) en cada nodo ( antes de shuffle) por clave K, combinando valores según indique el argumento func. Después vuelve a agregar los valores resultantes de todos los nodos mediante un shuffle y llamando de nuevo a func. |
| Ordenar por clave | sortByKey(Task) | Ordenar pares (K,V) siempre que K implemente Ordered ( sea un tipo de dato ordenable), de forma ascendente o descendente ( según indique el argumento) |

Ejemplo de flatMap:

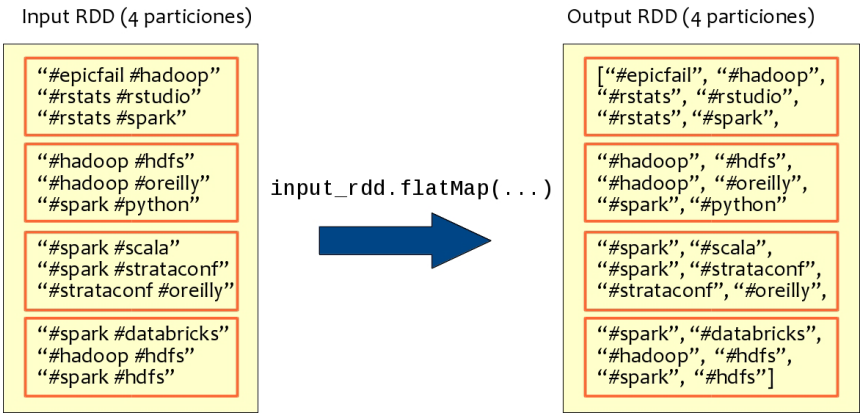



Ilustración 62 - Ejemplo flatMap.

Ejemplo de Map:

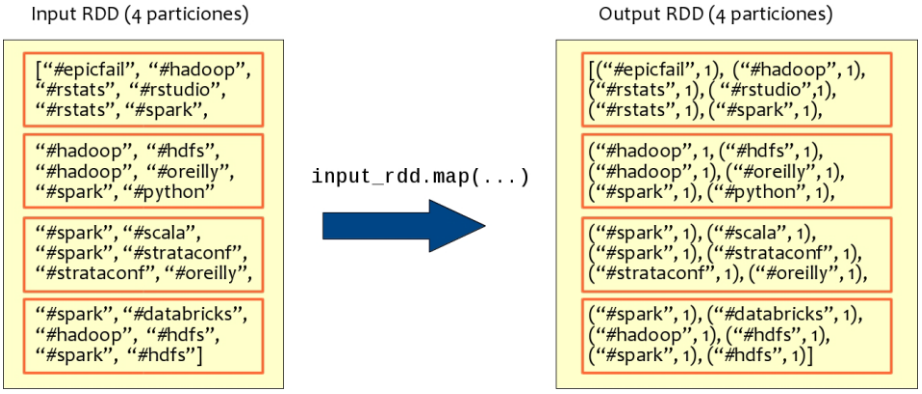



Ilustración 63 - Ejemplo Map.

Ejemplo de filter:

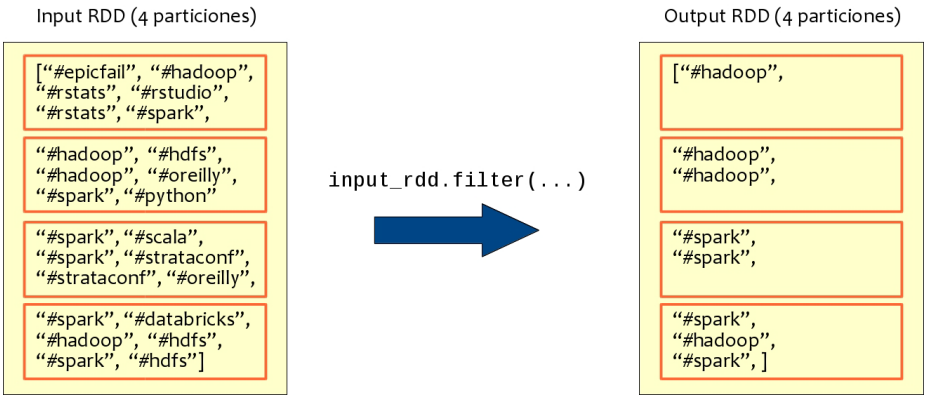



Ilustración 64 - Ejemplo Filter.

Ejemplo de distinct:

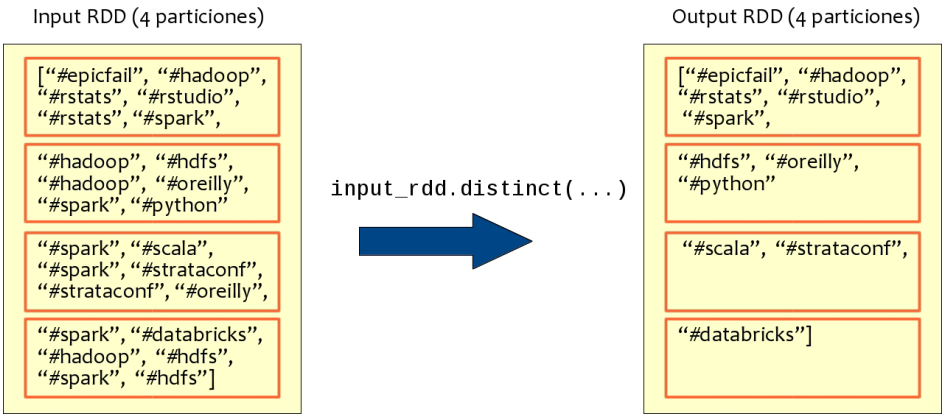



Ilustración 65 - Ejemplo distinct.

- **Acciones de un RDD**
  - Provoca que Spark ejecute la secuencia de operaciones para transformar los datos de entrada y obtener un resultado.
  - Define qué resultados se van a obtener después de la transformación:
    - Se puede recuperar todos los elementos.
    - También puede recuperar solo un subconjunto de resultados.
    - O puede guardar los resultados de forma persistente en disco, usando varios formatos ( como archivos de texto, archivos de secuencia u archivo objeto).

- Lista de acciones:

| Nombre | Función | Descripción |
|---|---|---|
| Reducir | reduce(función) | Agrega los resultados mediante función, función debe tomar dos valores y devolver uno. Además debe ser conmutativa y asociativa, para computarse en paralelo correctamente. |
| Obtener n | take(n) | Devuelve un array con los primeros n elementos del resultado. |
| Obtener todos | collect() | Devuelve al driver (cliente) todos los resultados. Debe de tener espacio suficiente en memoria del driver. |
| Obtener ordenados | takeOrdered(n,key=función) | Devuelve n elementos en orden ascendente o tal y como especifique la función opcional key. |
| Salvar TXT | saveAsTextFile(…) | Almacena los resultados en un archivo de texto en disco. |
| Salvar secuencia | saveAsSequenceFile(…) | Guarda los resultados en formato secuencia binaria de pares clave/valor (K,V), común en Hadoop. |
| Salvar Objetos | saveAsObjectFile(…) | Salva los resultados en un archivo como objetos serializados. |

Ejemplo típico de contar palabras:

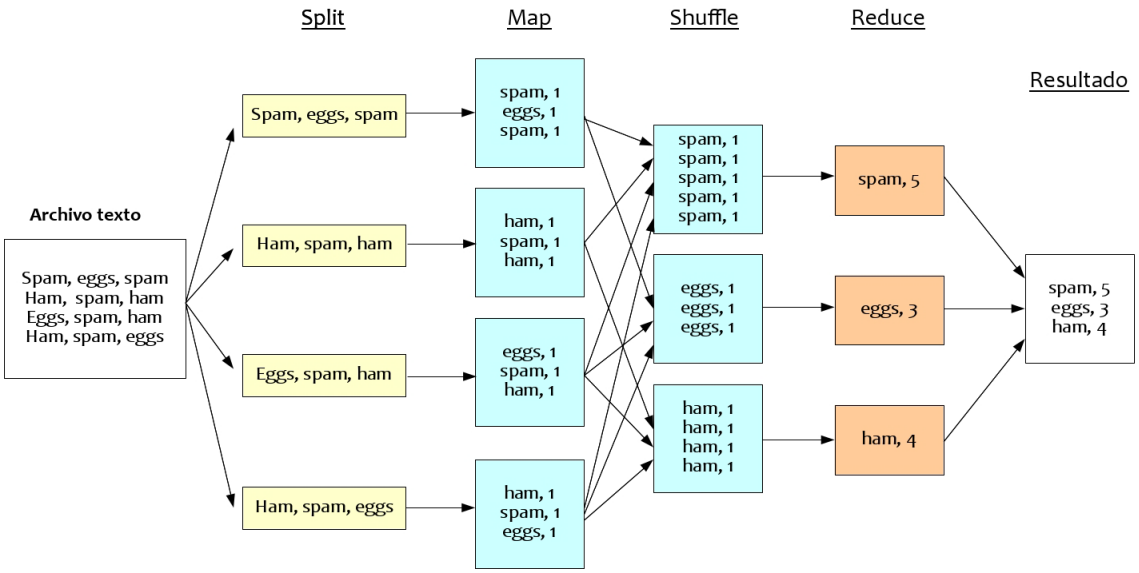



Ilustración 66 - Contar palabras.

### 4.2.5.3. Programación en SparkSQL

Se trata de un módulo que permite trabajar con datos estructurados ofreciendo una interfaz uniforme, con independencia de la fuente de datos que se emplea.

Fuentes posibles:

- Bases de datos relacionales.
- Bases de datos NoSQL.
- Ficheros de texto.
- Datos generados por una aplicación de Spark.

El elemento fundamental de trabajo son objetos de tipo DataFrame, que representan tablas de datos.

- **Programación con DataFrames**
  - Es una clase de Spark SQL que representa una colección de datos estructurados o semi-estructurados, distribuida en el clúster y organizada por columnas.

- A cada columna se le asigna un nombre.
- Es muy similar a las tablas de bases de datos SQL, o a los data.frame del lenguaje R, en el que se inspiran.
- Una cualidad importante: optimizadas para rendimiento y escalabilidad gracias a las capas de optimización lógica y física de Spark.
- Rendimiento optimizado con independencia del lenguaje de programación que se utilice.
- Para crear un DataFrame se puede usar la función createDataFrame aplicada sobre un RDD de tuplas o listas.
- Los datos almacenados en cada columna deben tener el mismo tipo, de lo contrario se lanzará una excepción TypeError.
- En los casos más usuales, la función createDataFrame puede detectar automáticamente el tipo de datos de cada columna, aunque la elección no será muy eficiente en términos de uso de memoria.
- Algunos de los tipos de datos disponibles:

| Clase Spark | Descripción |
|---|---|
| ByteType | Números enteros con signo almacenados en 1 byte ( se trunca si sobrepasa el límite de rango) |
| ShortType | Números enteros con signo almacenados en 2 byte ( se trunca si sobrepasa el límite de rango) |
| IntegerType | Números enteros con signo almacenados en 4 byte |
| LongType | Números enteros con signo almacenados en 8 byte |
| FloatType | Números en coma flotante con precisión simple (4 bytes) |
| DoubleType | Números en coma flotante con precisión doble (4 bytes) |
| DecimalType | Números decimales de punto fijo |
| StringType | Cadena de texto |
| BinaryType | Cadenas binarias de bytes |
| BooleanType | Valores booleanos (True) o (False) |
| TimestampType | Formato de fecha y hora |
| DateType | Fecha |

| | |
|---|---|
| ArrayType | ArrayType(tipoElem, [hasNull=True]) |
| MapType | MapType(tipoClave,tipoValor, [valorHasNull=True]) |

- La clase DataFrame ofrece ciertos métodos que son idénticos a los que se encuentran en la clase RDD:
    - Cache, collect, count, flatMap, foreach, map, persist, unpersist.
- Sin embargo, otros métodos son aplicables solo a objetos DataFrames:
    - Agg(lista_expresiones): el resultado es un DataFrame con una sola fila.
    - El valor de las columnas del resultado viene determinado por las funciones de agregación que se hayan aplicado.
    - Existen funciones predefinidas en Spark SQL:
        - Avg, count first, last, max, mean, mi, sum, sumDistinct.
    - Columns: atributo que contiene la lista de nombres de las columnas del DataFrame.
    - Corr('col1','col2'): método que devuelve el coeficiente de correlación (Pearson) de los valores entre ambas columnas( se supone numéricas).
    - Count(): número de filas del DataFrame.
    - Cov('col1','col2'): covarianza entre los valores de ambas columnas( se suponen numéricas).
    - Crosstab(col1,col2): devuelve un DataFrame con la tabla de contingencias entre los valores de ambas columnas. Se calcula por el orden en el que se especifican los parámetros de la función.
    - Distinct: devuelve un DataFrame en el que se ha eliminado las filas duplicadas.
    - dtypes: Atributo que contiene el nombre y tipo de dato asociado a cada una de las columnas del DataFrame.
    - drop('col'): Retorna un DataFrame en el que se ha eliminado la columna cuyo nombre se indica como argumento.

- dropDuplicates([columnas=None]): Similar a distinct(), pero aquí permite especificar como argumento de la función sobre qué columnas se comprueba si hay valores duplicados.
- dropna([which='any'], [threshold=None], [col=None]): Devuelve un nuevo DataFrame eliminando filas con valores faltantes. Si which='any', entonces elimina cualquier fila que tenga al menos un valor faltante. Si which='all', solo elimina las filas con todos sus valores faltantes. threshold permite definir cuantos valores no faltantes debe tener la columna para mantenerla (invalida el argumento anterior). El último argumento permite controlar sobre qué columnas se hace la comprobación.
- fillna(value, [col=None]: Sustituye los valores faltantes por un nuevo valor, que se pasa como argumento. Se puede indicar sobre qué columnas se quiere operar.
- filter(boolean_condition): Retorna un DataFrame que retiene solo las columnas que cumplen con la condición especificada.
- groupBy(cols): Realiza una función de agrupación en base a los valores de las columnas indicadas. Se pueden usar las funciones de agregación típicas (avg, max, min, etc.).
- intersect(otro_df), substract(otro_df), unionAll(otro_df): Efectúa operaciones de conjuntos entre el DataFrame sobre el que se aplica el método y el que se pasa como argumento.
- limit(n): Limita la salida de resultados a n filas.
- orderBy(cols): Ordena resultados por el valor de las columnas cuyo nombre se pasa como argumento.
- sort(cols): Equivalente al anterior.
- rdd: Atributo que almacena un RDD con las filas del DataFrame.
- select(cols): Selecciona las columnas especificadas como argumento.
- selectExpr(expressions): Calcula un nuevo DataFrame aplicando las expresiones indicadas como argumento a cada columna.
- withColumn(expressions): Añade una nueva columna aplicando la expresión que se pasa como argumento.

# 5. MANUAL DE INSTALACIÓN

## 5.1. Nomenclatura

Aquí se detallan los valores correspondientes al direccionamiento de los nodos del clúster.

| Concepto | Valor en entorno de desarrollo | Descripción |
|---|---|---|
| **MASTER** | cloudera1 | Hostname del nodo Master |
| **IP_MASTER** | 127.0.0.1 | IP del nodo Master |
| **HOST_MASTER** | localhost.localdomain | FQDN del MASTER |
| **WORKER1** | cloudera2 | Hostname del nodo Worker1 |
| **IP_ WORKER1** | 127.0.0.1 | IP del nodo Worker1 |
| **HOST_ WORKER1** | localhost.localdomain | FQDN del Worker1 |
| **WORKER2** | cloudera3 | Hostname del nodo Worker2 |
| **IP_ WORKER2** | 127.0.0.1 | IP del nodo Worker2 |
| **HOST_ WORKER2** | localhost.localdomain | FQDN del Worker2 |
| **WORKER3** | cloudera4 | Hostname del nodo Worker3 |
| **IP_ WORKER3** | 127.0.0.1 | IP del nodo Worker3 |
| **HOST_ WORKER3** | localhost.localdomain | FQDN del Worker3 |

## 5.2. Detalles de la instalación

La instalación de Cloudera a Hadoop que se recoge a continuación es de tipo PATH A o instalación Automática Se utiliza el asistente de Cloudera Manager para automatizar la instalación de agentes y Cloudera Distribution including Hadoop (CDH).

El clúster de desarrollo se compondrá de los siguientes nodos:

| Tipología | Referencia del nodo |
|---|---|
| **Worker** | $WORKER1 |
| | $WORKER2 |
| | $WORKER3 |
| **Master, Gestión** | $MASTER1 |

Versiones de software para la instalación:

| Software | Versión |
|---|---|
| **Sistema Operativo** | Red Hat 6.7 |
| **Cloudera Manager** | 5.8.0 (Cloudera) |
| **Cloudera Distribution including Hadoop (CDH)** | CDH5.8 (Cloudera) |
| **Java Development Kit** | 1.7.0_67 (Cloudera) |

## 5.3. Requisitos iniciales

Se detalla a continuación los prerrequisitos o acciones previas que realizar para proceder a la instalación de la plataforma Cloudera:

- **Requisito inicial 0**: Tener acceso root a los sistemas de ficheros de los nodos.
- **Requisito inicial 1**: Los nombres DNS de los host deben estar acorde a la norma FQDN (hostname.dominio.com).

Ejemplo de configuración para la máquina c-xxxx, IP = xxx.xxx.xxx.xxx

```
#     IP          FQDN              Aliases

# Definición del localhost
127.0.0.1         localhost
# Identificación de la máquina
xxx.xxx.xxx.xxx   c-xxxx.cm.es            c-xxxx
```

- **Requisito inicial 2:** Todos los nodos tienen que poder comunicarse entre ellos
- **Requisito inicial 3:** Conexión *SSH* a cualquier nodo del clúster.
- **Requisito inicial 4**: Deshabilitar *IPv6*. Para ello, se puede hacer uso de las instrucciones siguientes:

En el fichero etc/sysconfig/network, se actualiza la siguiente linea

```
# NETWORKING_IPV6=no
```

- **Requisito inicial 5**: Deshabilitar *SELinux (POLITICAS DE SEGURIDAD)*. Para ello, se puede hacer uso de las instrucciones siguientes:

```
# vi /etc/sysconfig/selinux
```

Cambiar el parámetro SELINUX a disabled

```
SELINUX=disabled
```

Guardar y reiniciar el sistema operativo

- **Requisito inicial 6**: Configurar la propiedad *swapiness*. Esta propiedad se utiliza para establecer balance entre el espacio de intercambio y la memoria RAM. Si se establece a 0, no habrá intercambio. Si se establece a 100, el sistema intentará mantener la RAM lo más libre posible haciendo intercambio. Para configurar la propiedad, utilizar las instrucciones:

```
# echo "vm.swappiness = 0" >> /etc/sysctl.conf
# echo 0 > /proc/sys/vm/swappiness
# reboot
```

Además se modifica el valor de la propiedad defrag .

```
# echo never > /sys/kernel/mm/redhat_transparent_hugepage/defrag
```

Se introduce esa directiva en el fichero de arranque de cada nodo para que se modifique dicha propiedad con cada reinicio. Se introduce lo siguiente en el fichero /etc/rc.d/rc.local

```
echo never > /sys/kernel/mm/redhat_transparent_hugepage/defrag
```

- **Requisito inicial 7:** Verificar que la configuración de la internacionalización del sistema sea en_US.UTF-8

```
# vi /etc/sysconfig/i18n
```

Modificar la variable LANG si procede.

```
LANG=”en_US.UTF-8”
```

- **Requisito inicial 8** Desactivar el servicio IPTables e IP6Tables de todos los nodos del clúster.

```
# service iptables stop
# chkconfig iptables off
# service ip6tables stop
# chkconfig ip6tables off
```

- **Requisito inicial 9:** el umask para los usuarios root deberá ser de **0022.**

- **Requisito inicial 10:** Los permisos de las carpetas /usr y /usr/bin deberán ser los permisos por defectos definidos en el sistema RHEL.
    - **/usr :** uid=root, gid=root, mode=755.
    - **/usr/bin :** uid=root, gid=root, mode=555.

- **Requisito inicial 11:** Desactivar THP (Transparent HugePages) en todas las máquinas del clúster. Para ello, acceder a los ficheros:

```
# vi /etc/grub.conf
# vi /boot/efi/EFI/redhat/grub.conf
```

Modificar la variable:

```
transparent_hugepage=never
```

- **Requisito inicial 12:** Sincronizar los relojes de los nodos :

Para ello editar el fichero siguiente añadiendo las lineas mostradas a continuación :

```
# vi /etc/rc.local
```

Añadir el contenido siguiente:

```
/etc/init.d/ntpd stop
sleep 2 ;
ntpdate -s es.pool.ntp.org
/etc/init.d/ntpd start
```

## 5.4. Instalación de Cloudera Manager

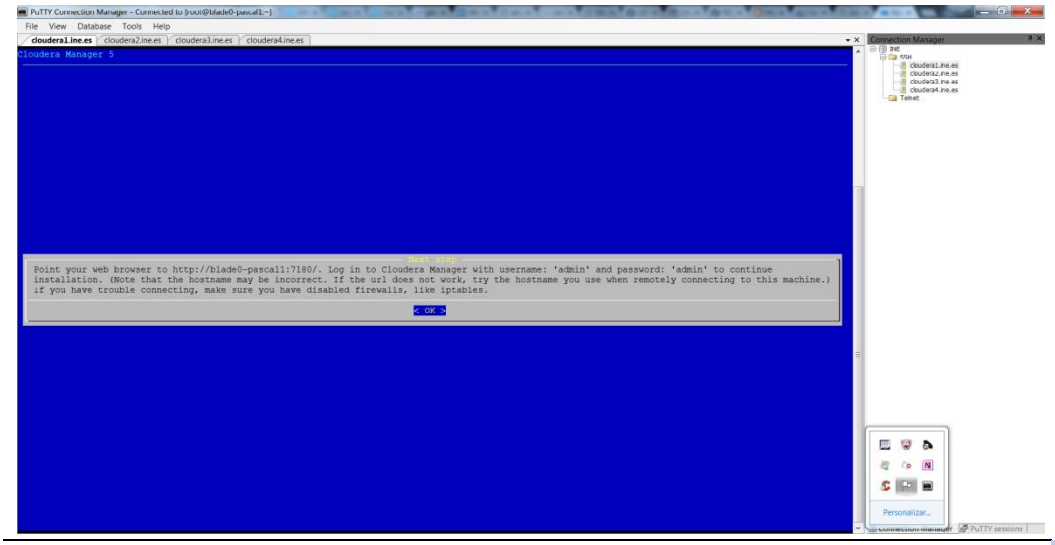
Se descarga de la página de Cloudera el Installer: **cloudera-manager-installer.bin**

Se realiza la copia del fichero de Cloudera Manager en el master, se modifican los permisos y se ejecuta:

```
$ chmod u+x cloudera-manager-installer.bin
```

```
$ sudo ./cloudera-manager-installer.bin
```

Una vez finalizado este proceso estará instalada la versión inicial de Cloudera Manager, a partir de la cual comenzará el proceso de instalación y configuración del clúster.



### 5.4.1. Configuración de Cloudera Manager

Desde un navegador web, acceder a la URL http://localhost:7180

Elegir la opción de instalación Cloudera Express y pulsar Continuar:

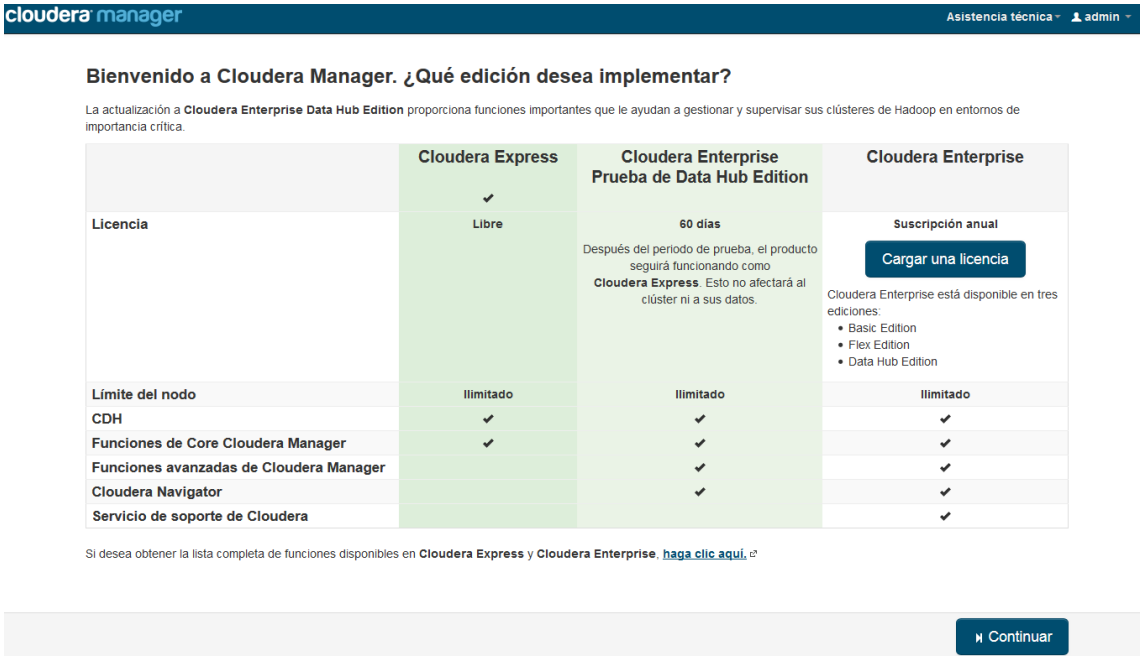



Ilustración 67 - Configuración de Cloudera Manager.

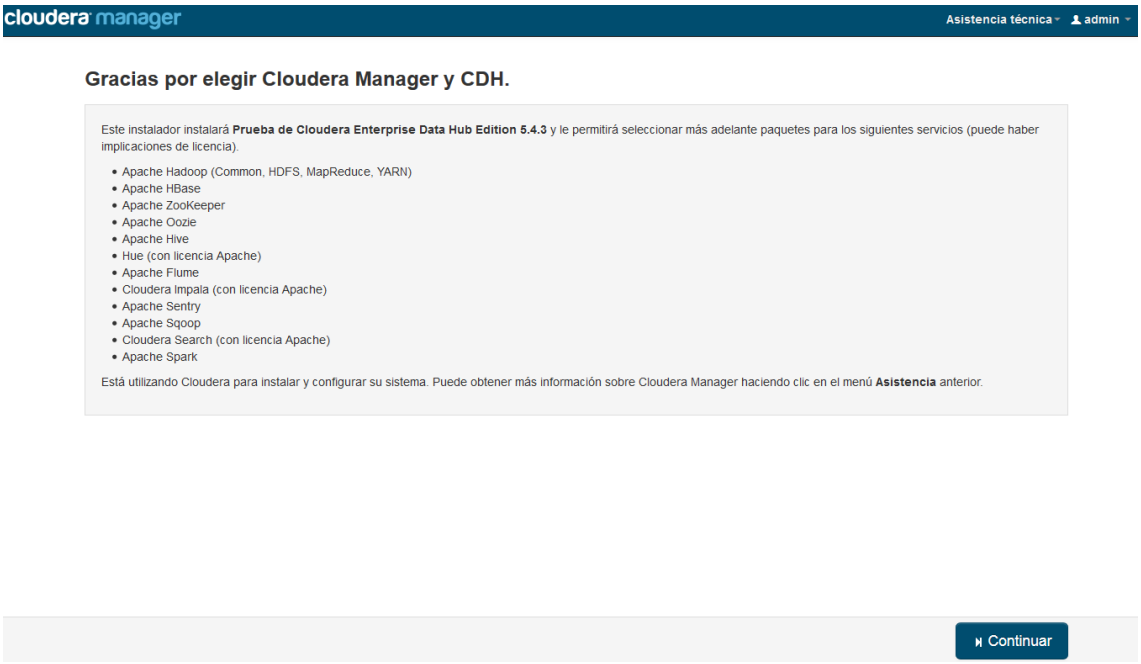



Ilustración 68 - Configuración de Cloudera Manager.

Marcar la lista de las IPs o los FQDNs de los nodos que componen el clúster, cambiar el puerto SSH para adaptarlo a la configuración vigente, y pulsar el botón Búsqueda.


Ilustración 69 - Configuración de Cloudera Manager.

Seleccionar los nodos que agregar al clúster y pulsar Continuar


**Ilustración 70 - Configuración de Cloudera Manager.**

Elegir el método de instalación por paquetes

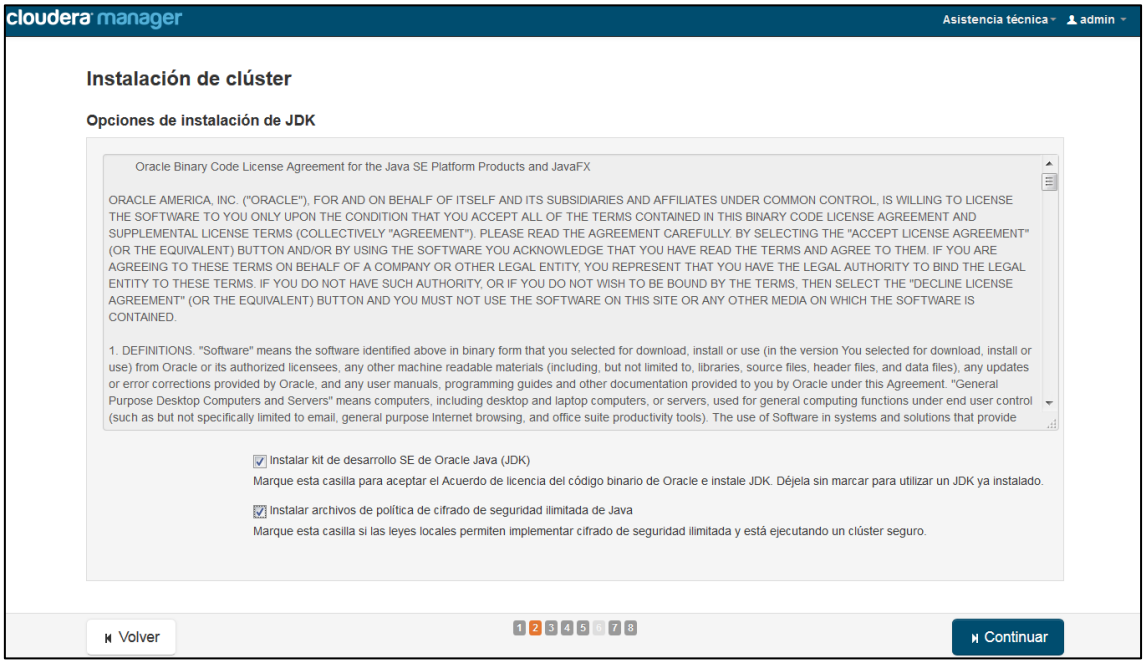



Ilustración 71 - Configuración de Cloudera Manager.

Marcar las opciones siguientes y pulsar Continuar:

- Instalar Kit de desarrollo SE de Oracle Java (SDK) para instala el Java 1.7 necesario.
- Marcar la casilla Instalar archivos de política de cifrado de seguridad ilimitada de Java. Permitirá en un futuro activar las conexiones TLS necesarias.

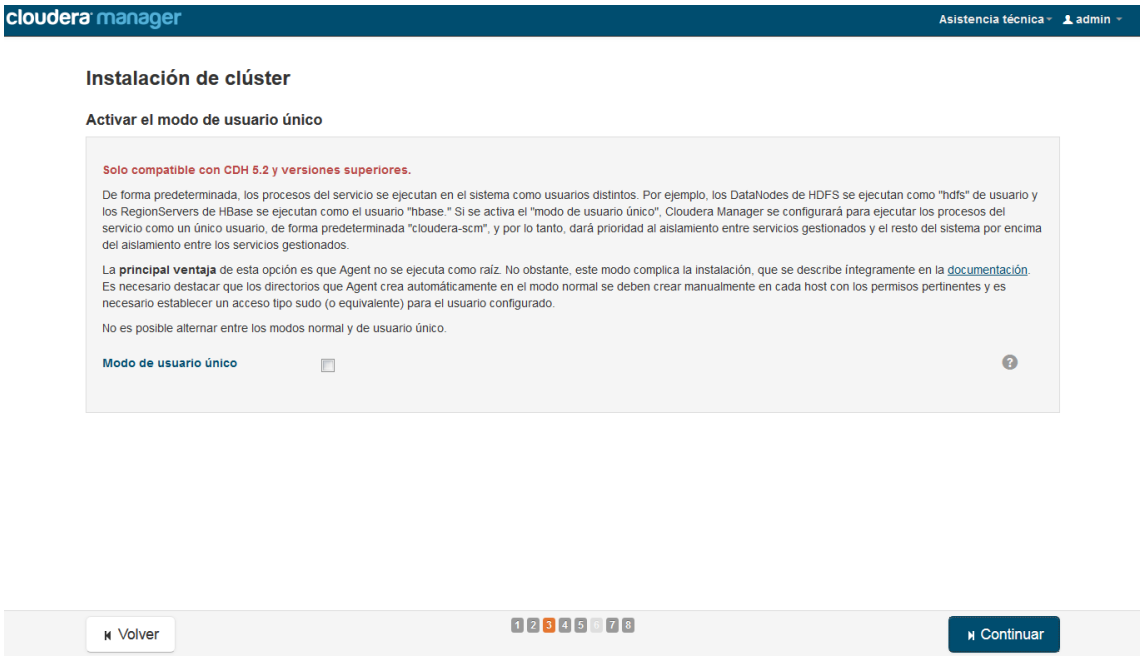



Ilustración 72 - Configuración de Cloudera Manager.

Pulsar continuar


Ilustración 73 - Configuración de Cloudera Manager.

Elegir el usuario root (contraseña changeme) para proceder a la instalación, y el método de autenticación. Todos los hosts aceptan la misma contraseña. Marcar la contraseña y adaptar el puerto SSH a las necesidades, y pulsar Continuar.


Ilustración 74 - Configuración de Cloudera Manager.

Una vez inicializadas las máquinas, pulsar Continuar.

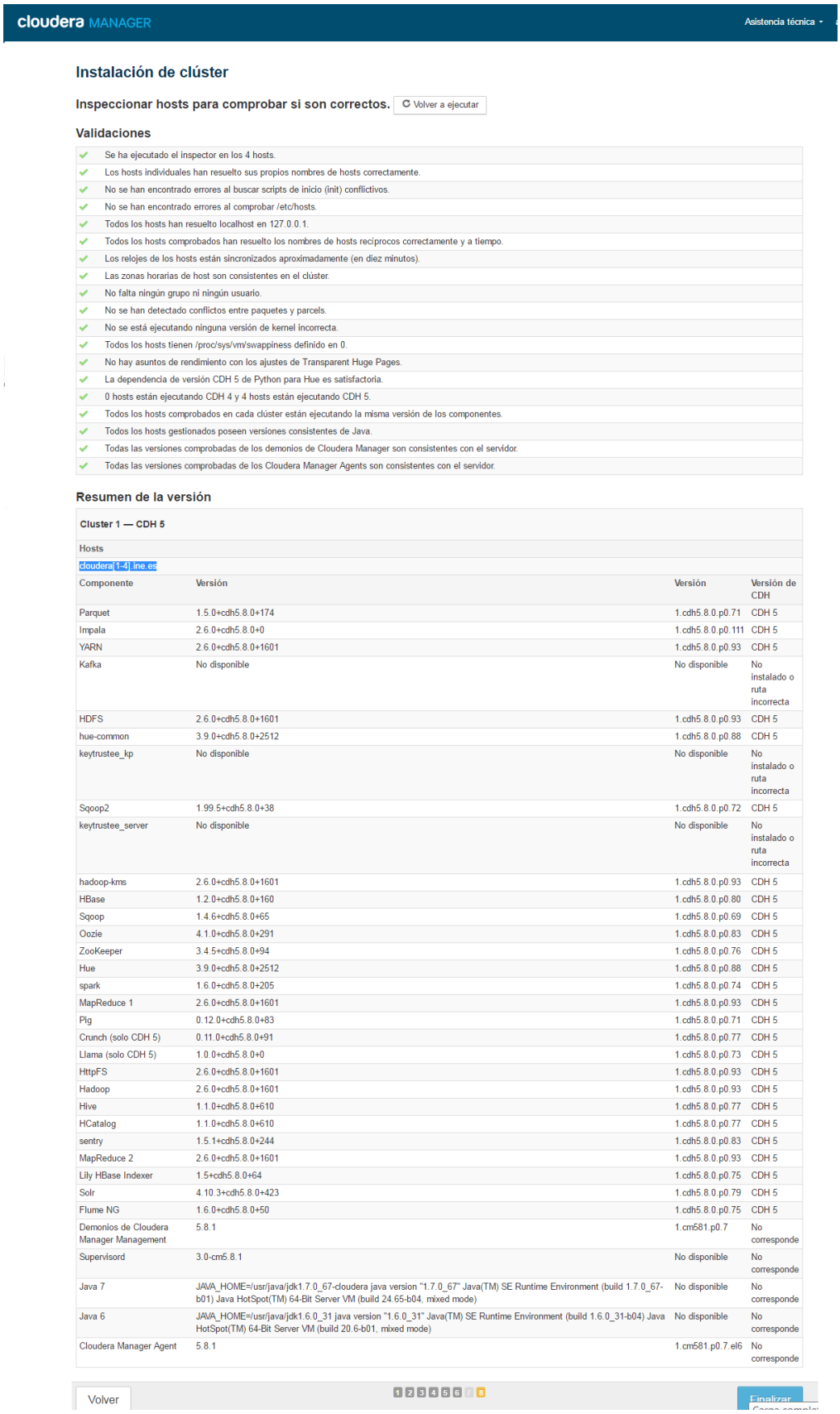



Ilustración 75 - Configuración de Cloudera Manager.

En el caso de que Cloudera Manager detecte un error de configuración de algún nodo, revisar los prerrequisitos y volver a ejecutar la inspección pulsando el botón Volver a ejecutar. Si la configuración de los hosts es válida, pulsar Finalizar.

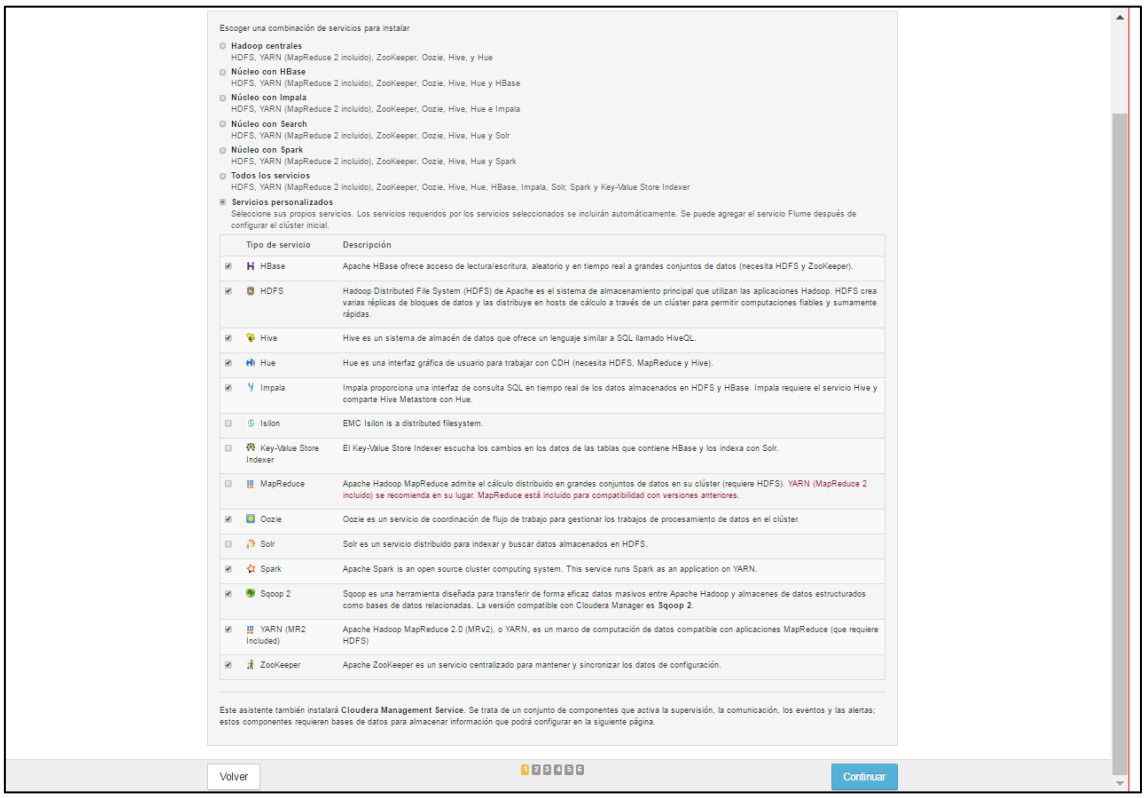



Ilustración 76 - Configuración de Cloudera Manager.

Elegir los servicios básicos a utilizar y pulsar Continuar:

- HBASE
- HDFS
- Hive
- Hue
- Impala
- Oozie
- Spark
- Sqoop2
- YARN (MR2 included)
- Zookeeper


Ilustración 77 - Configuración de Cloudera Manager.

Una vez terminada la asignación de servicios a los diferentes nodos nos quedara así:

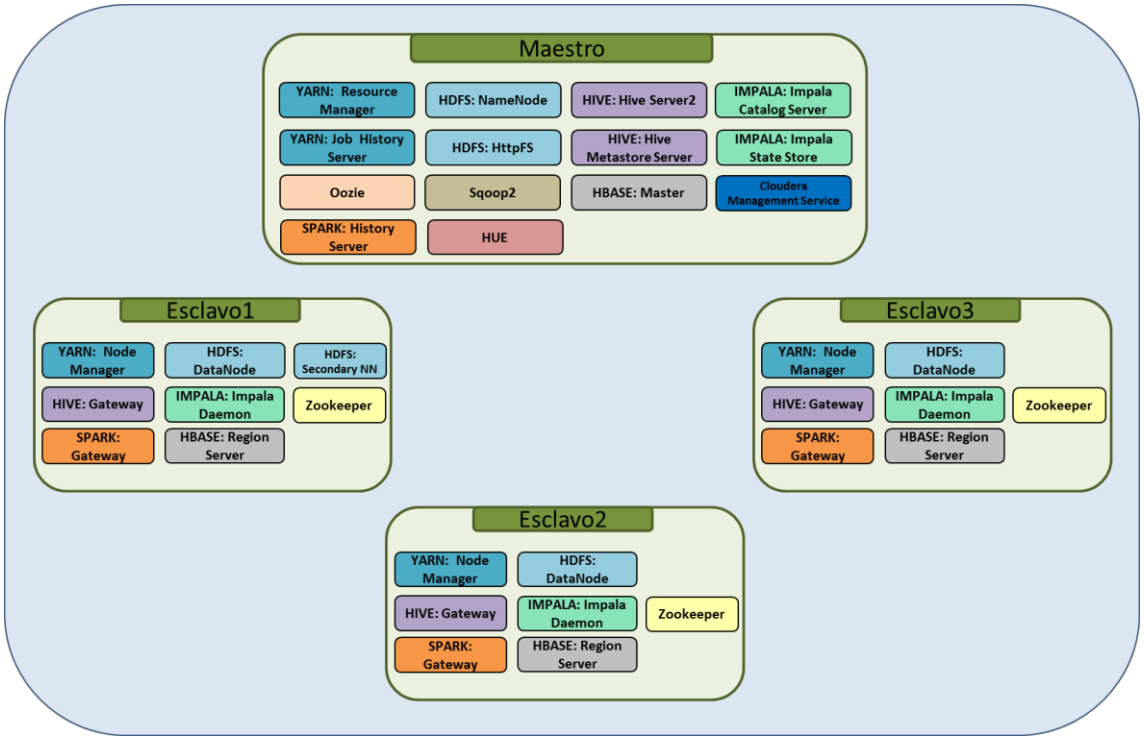



Ilustración 78 - Clúster gráfico.

Se modifican las password con valores más sencillos, y pulsar Probar conexión.


Ilustración 79 - Configuración del clúster.

Una vez las conexiones validadas, pulsar Continuar.

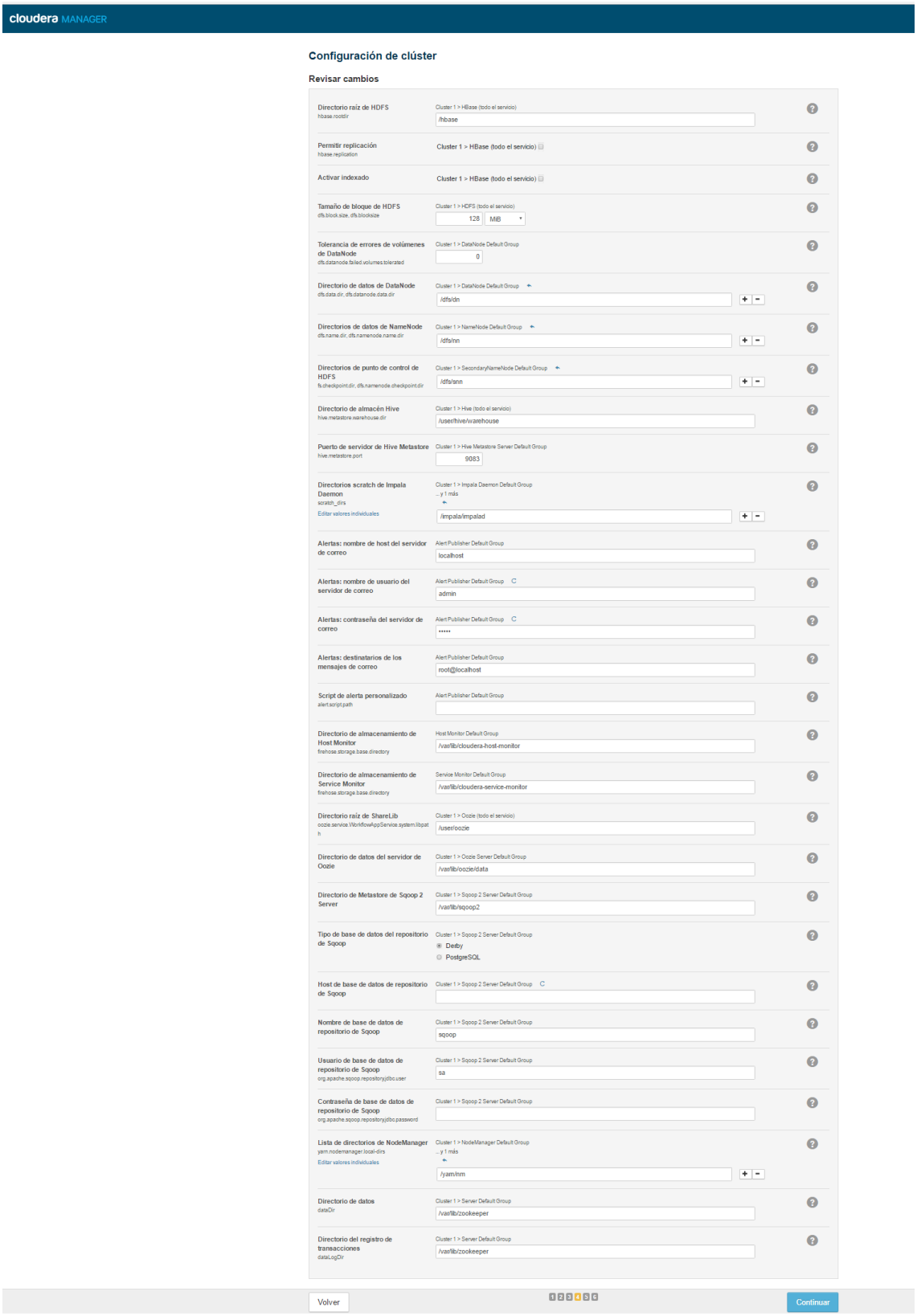



Ilustración 80 - Configuración del clúster.

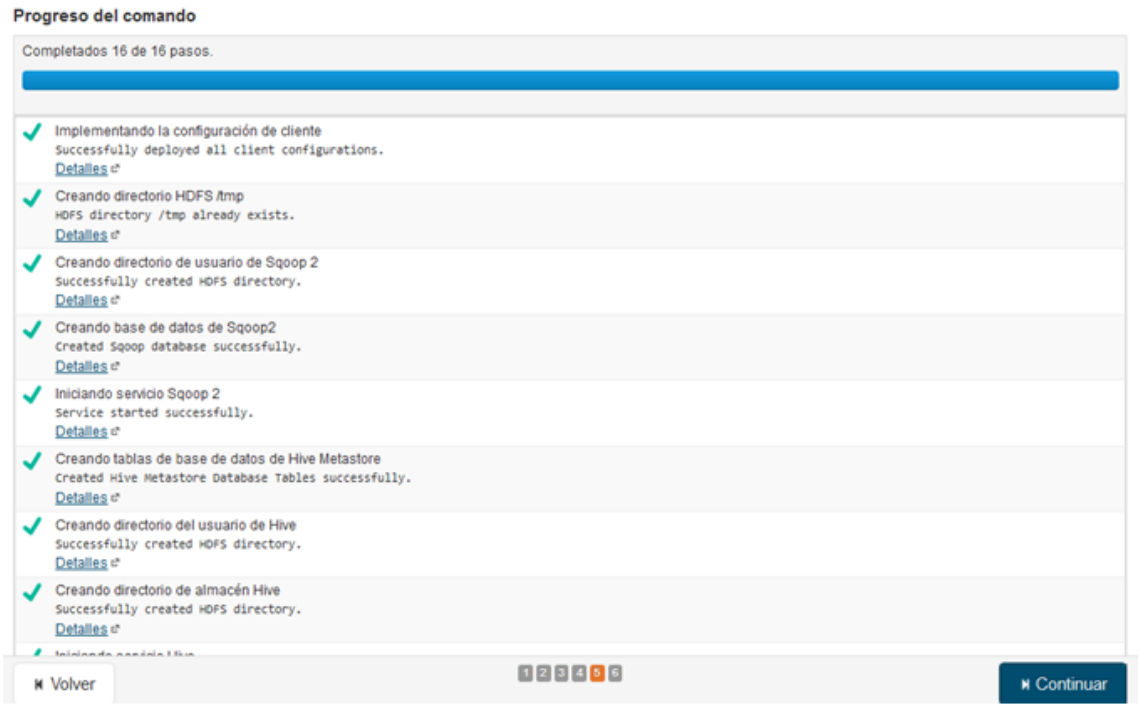



Ilustración 81 - Configuración del clúster.

Una vez la configuración finalizada, pulsar Continuar


Ilustración 82 - Configuración del clúster.

Pulsar Finalizar.

## 5.4. Inicio de Cloudera Manager Server y Agents

Ejecutar el comando siguiente para el server:

```
# service cloudera-scm-server start
```

Ejecutar el comando siguiente para los agents:

```
# service cloudera-scm-agent restart
```

Se puede consultar el estado de los servicios de este modo:

```
# service cloudera-scm-server status
```

Ejecutar el comando siguiente para los agents:

```
# service cloudera-scm-agent status
```

El servicio tarda en arrancar. Se puede seguir el avance del arranque a través del comando siguiente:

```
# tail -f /var/log/cloudera-scm-server/cloudera-scm-server.log
```

El servicio estará totalmente arrancado cuando el log enseñe el mensaje siguiente:

```
INFO WebServerImpl:com.cloudera.server.cmf.WebServerImpl: Started Jetty server.
```

## 5.5. Test de Instalación

Una vez instalado y configurado el clúster, la página de inicio de Cloudera Manager debe tener un aspecto similar a la imagen siguiente:

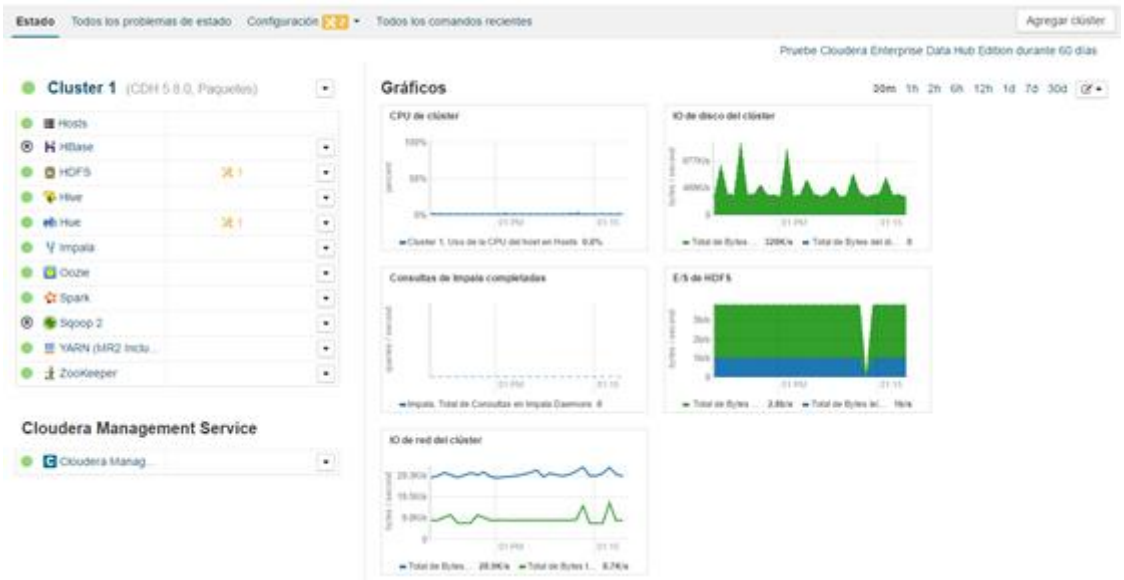



Ilustración 83 - Test de instalación.

En el lado izquierdo de la pantalla se muestra una lista de los servicios actualmente en ejecución con su información de estado. Todos los servicios en ejecución deben tener el semáforo en verde, signo de ausencia de errores. Haciendo clíc en cada servicio se puede ver información más detallada acerca del mismo.

Existe otro tipo de test llamado test de latido, es una forma de comprobar si todos los agentes se están ejecutando correctamente; es mirar el tiempo transcurrido desde su último latido. Esto puede comprobarse haciendo clíc en la pestaña Hosts donde se puede ver una lista de todos los nodos y el valor de su último latido. De forma predeterminada, cada agente debe responder con un latido con éxito cada 15 segundos. Un valor reciente del último latido significa que el servidor y los agentes se comunican correctamente.

## 5.6. Ejecución de un trabajo MapReduce y Spark

Realizar esta  prueba desde línea de comandos.

Hay que tener permisos sobre la ruta /user, por ello:

```
$ groupadd supergroup
$ usermod -a -G supergroup root
$ sudo -u hdfs hdfs dfs -chown root:root /user
```

Ejecutar el ejemplo Hadoop PiEstimator con el siguiente comando:

```
HDFS$ hadoop jar hadoop-mapreduce-examples-0.23.6.jar pi 10 100
```

Para ver los resultados de la ejecución, pulsar la siguiente opción en la barra de navegación superior de la consola de administración de Cloudera Manager:

**Clústeres> Aplicaciones  YARN.**

Aparecerá una entrada como la siguiente:

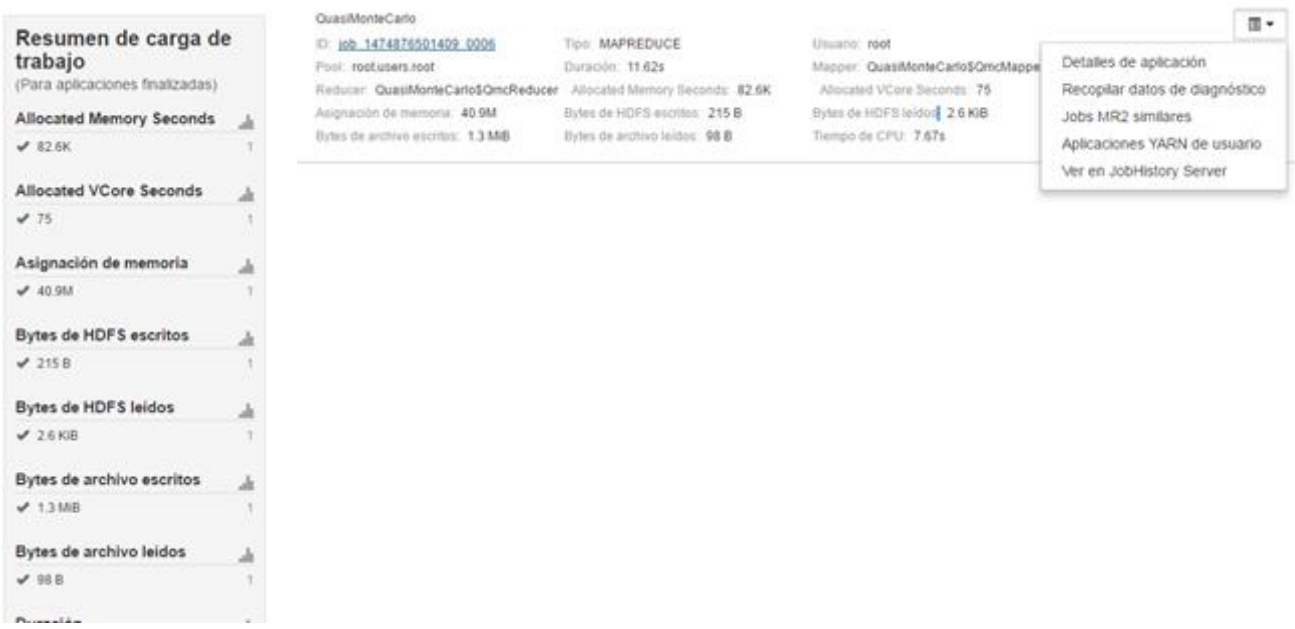



Ilustración 84 - Resumen de carga de trabajo.

También es posible ver los resultados de esta ejecución desde la aplicación web Hue. Hue es una interfaz gráfica de usuario que permite interactuar con el clúster mediante la ejecución de aplicaciones que permiten navegar HDFS, administrar la base de datos de Hive, ejecutar Hive, Impala, scripts de Pig y flujos Oozie.

Para acceder a Hue utilizar el siguiente enlace desde un navegador: http://localhost:8888/

Para Iniciar sesión usar las credenciales:

- Nombre de usuario: hue.
- Contraseña: hue.

Desde la pantalla principal de Hue hacer clic en “Job Browser” en la barra superior.

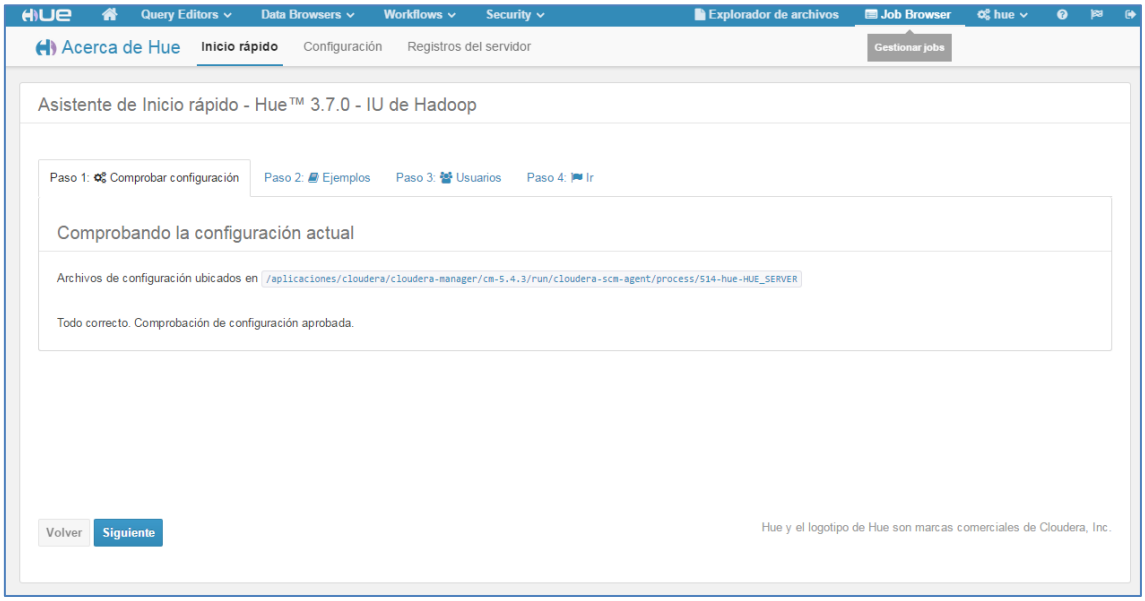



Ilustración 85 - Hue.

Desde el Job Browser de Hue es posible visualizar los resultados de la ejecución, así como una serie de estadísticas básicas de la misma.

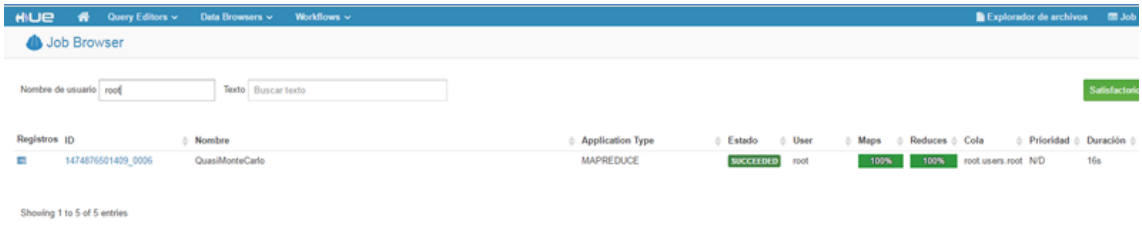



Ilustración 86 - Hue.

# 6. DESCRIPCION EXPERIMENTAL

Big data es o bien Volumen o bien Velocidad o bien Variabilidad.

Para el primero se necesita fuerza bruta y supongo que no se dispone del "hierro" para demostrar el potencial con grandes cantidades de datos, así que se irá por Velocidad y Variabilidad que es cruzar información dispar, ¿Qué información? Toda la que se pueda extraer desde Twitter sobre un evento deportivo en este caso la carrera de moto GP , en cruzar toda la información disponible en Twitter y procesarla con tecnología Big Data. Para saber por ejemplo los usuarios con más followers y el grado de influencia que tienen, los países que más tweets twittean, "X" usuarios con más tweets, numero de tweets por día y hora…

## 6.1. Motivación

- Una de cada diez personas en el mundo tienen una cuenta activa en Facebook.
- El 71% de los adultos en EEUU utilizan Facebook.
    - El 84% de las personas entre 18-29 años.
- Diariamente se envían más de 400 millones de tweets.
    - Hay más de 200 millones de usuarios activos
- El 23% de los adultos en EEUU utilizan Twitter.
- El 31% de las personas entre 18-29 años.
- Instagram tiene más de 300 millones de usuarios activos.
- El 26% de los adultos de EEUU.
- Existe un gran número de empresas que hoy en día centran su negocio en el análisis de redes sociales con diferentes fines.
    - Pirendo: http://pirendo.com
    - Dogtrack: http://www.dogtrack.es/es/inicio/
    - Kantar Media: http://www.kantarmedia.com/es
        - Tuitele: http://blog.tuitele.tv
        - http://www.elmundo.es/television/2014/04/02/533bf0d5e2704e8f3b8b4570.html
    - Graphext: https://graphext.com/
        - http://politica.elpais.com/politica/2016/01/30/actualidad/1454183015_952770.html

- http://elpais.com/elpais/2015/12/19/media/1450556560_905182.html
- http://embeddables.graphext.com/ec/graphext_tuenti_mafia_full.html

## 6.2. Proceso de autenticación

- Para poder acceder a los datos de una cuenta de cualquier red social, es necesario llevar a cabo un proceso de autenticación que nos valide como usuarios.
- Sin embargo, sería interesante que nuestra aplicación pudiera autenticarse sin necesidad de introducir usuario y contraseña, pero sin perder seguridad.
- OAuth (open authorization) proporciona a los usuarios los medios necesarios para autorizar a una aplicación a acceder a información limitada de su cuenta a través de un API.
- Facilita la integración entre aplicaciones sin necesidad de intercambios de contraseñas, permitiendo al usuario revocar los privilegios cuando lo necesite.
- Puede utilizarse en todo tipo de aplicaciones, incluyendo web, de escritorio y móviles.
- Existen cuatro roles principales:
    - Propietario del recurso: el usuario que autoriza a la aplicación a acceder a su cuenta. El acceso puede limitarlo el propio usuario.
    - Servidor de recurso: almacena las cuentas de usuario protegidas.
    - Servidor de autorización: verifica la identidad del usuario y crea el token de acceso para la aplicación.
    - Cliente: la aplicación que quiere acceder a la cuenta de usuario. Antes de acceder, debe ser autorizada por el usuario, y debe ser validada por los servidores de recurso y autorización.

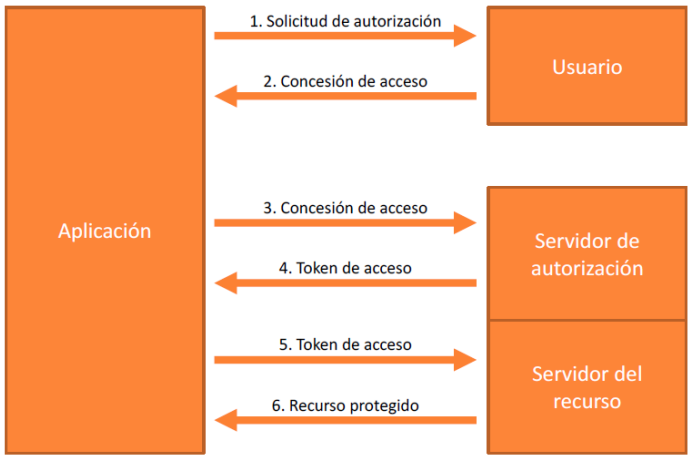



Ilustración 87 - Diagrama acceso.

## 6.3. Apps de Twitter

- En primer lugar necesitamos crear la aplicación de Twitter que accederá a nuestros datos.

- https://apps.twitter.com/


- A continuación, se incluye la información de la aplicación


Ilustración 88 - Twitter Apps.

- Finalmente se verá la información referente a la aplicación.
- Para poder utilizarla, se deberá generar un token de acceso.

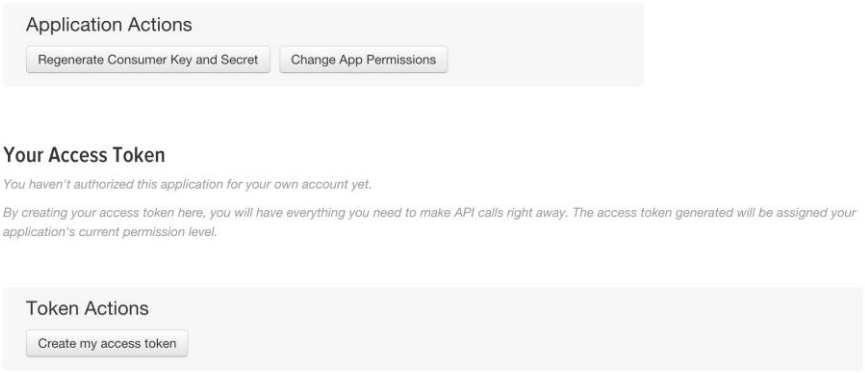



Ilustración 89 - Twitter Apps.

## 6.4. API de Twitter

- API REST
  - Permite preguntar sobre información específica de usuarios y tweets.
  - Por ejemplo: perfil del usuario, lista de seguidores y amigos, tweets generados por un usuario (timeline), listas de usuarios, etc.

- API Streaming
  - Permite conectarse a un stream de tweets de manera que se puedan recuperar en cuanto son publicados.
  - Existen tres streams principales:
    - Filter stream: filtra los tweets según una serie de palabras clave.
    - Geo stream: filtra los tweets según localización.
    - Sample stream: obtiene un 1% de tweets aleatorios.

## 6.5. API REST de Twitter

- Existen numerosas librerías para diferentes lenguajes que permiten el acceso al API de twitter.
  - C++: twitcurl.
  - Java: Twitter4J.
  - Javascript: TwitterJSClient, user-stream
  - Objective-C: STTwitter, FHSTwitterEngine

- PHP: tmhOAuth, twitteroauth, codebird-php, etc.
- Python: tweepy, twitter-ads, python-twitter, etc.

- La elección de una librería dependerá principalmente de dos factores:
  - El lenguaje de la aplicación donde se quiera integrar twitter.
  - Las características que ofrece cada una de las librerías disponibles para el lenguaje.
  - En nuestro caso nos centraremos en la librería twitter disponible para Python.
    - Pip install twitter
    - https://pypi.python.org/pypi/twitter

## 6.6. Formato JSON

- JavaScript Object Notation.
- Formato utilizado para el intercambio de datos que nace como alternativa a XML.
- Su sencillez ha conseguido que sea más utilizado que XML.
- Se puede analizar desde cualquier lenguaje de programación.
- Existen numerosas herramientas de visualización de ficheros JSON:
  - http://www.jsoneditoronline.org/
  - http://json.parser.online.fr/

- Los datos recuperados con el API de Twitter están en formato JSON.
- Para imprimir los datos en un formato legible, se puede utilizar la librería Json.
  - json.dumps(datos, indent=1)
- Un conjunto de datos JSON en Python se comporta como un diccionario.
  - Se puede acceder a cada campo con el operador [].

```
{
  "users": [
    {
      "name": "Pepe",
      "address": "La calle de Pepe"
    },
    {
      "name": "Juan",
      "address": "La calle de Juan"
    }
  ]
}
```


Ilustración 90 - Twitter Apps.

- Es conveniente siempre tener disponible la documentación de las consultas que se puede hacer.
    - https://dev.twitter.com/rest/public
- El API REST de Twitter tiene ciertas limitaciones que se deba considerar:
    - Los límites de acceso se establecen por cada access token que se controla.
    - Los límites están divididos por ventanas de 15 minutos.
        - Cada 15 minutos se resetean las peticiones realizadas.
        - Límites de cada tipo de petición:
            - https://dev.twitter.com/rest/public/rate-limits

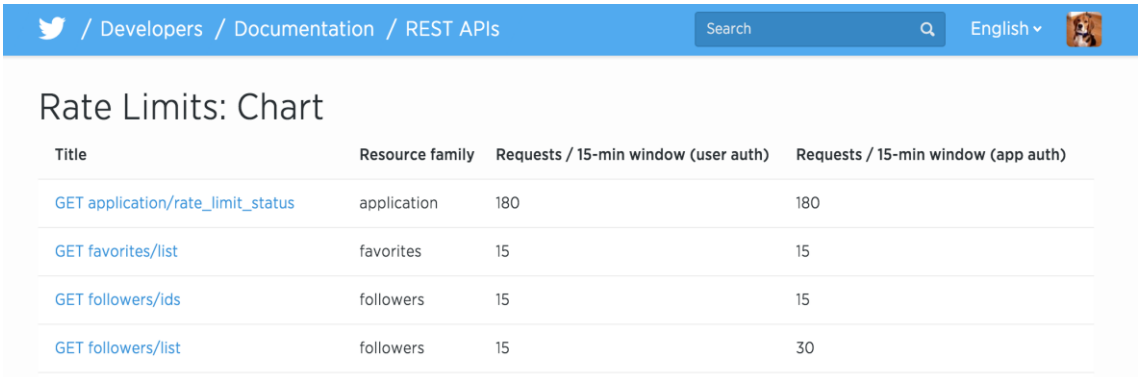



Ilustración 91 - Twitter Apps.

- Si se produce un error al realizar una petición, el API elevará una excepción de tipo TwitterHTTPError.
- Se debe controlar qué tipo de error se ha producido: límite de peticiones, error de usuario/contraseña, fallo de red.
- En concreto, la excepción debida a haber superado el límite de peticiones es un JSON con el siguiente contenido:

```
{
  "errors": [
    {
      "message": "Rate limit exceeded",
      "code": 88
    }
  ]
}
```

Ilustración 92 - Twitter Apps.

## 6.7. Caso de uso

Una vez sacados los datos de twitter, en este caso de la carrera de Qatar del 2014, puesto que son las únicas fechas en las que se ha podido sacar realmente datos, sin que saltasen errores o sin que se cortase la conexión, las capturas de datos son del 23 al 26 de Marzo del 2014.

Muestra:

| | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | ID,"PARENT-SYS-ID","Source","Mentions","Target","NAME Source","BODY","PUBDATE","URLs coma separated","Type TW-RT-MT","LINK","n1 Link","n1 Picture","PERSONAL-WEBSITE","COUNTRY","ALL-NICK-ACTIVITY- | | | | | | | | | | | | | | | |
| 2 | 600621,"sin padre","Henrito_fx3",,,,"Henrit?. ?","- #henritofresh alex mÃ¡rquez (honda): 'he cometido un error de novato': el piloto espaÃ±ol de moto3 alex mÃ¡rque... http://bit.ly/olxd7p",23/03/2014 22:32,"http://b | | | | | | | | | | | | | | | |
| 3 | 604076,"sin padre","Henrito_fx3",,,,"Henrit?. ?","- #henritofresh con mÃ¡rquez, todo igual: los focos, puestos en Ã©l desde que debutara en el mundial en 2013, vo... http://bit.ly/1deycvz",23/03/2014 22:48,"http://b | | | | | | | | | | | | | | | |
| 4 | 512641,"sin padre","Henrito_fx3",,,,"Henrit?. ?","- #henritofresh en vivo: carrera de moto2: ... http://bit.ly/1hbliq7",23/03/2014 18:37,"http://bit.ly/1hbliq7","TW","http://twitter.com/Henrito_fx3/statuses/447789265 | | | | | | | | | | | | | | | |
| 5 | 643917,"sin padre","Henrito_fx3",,,,"Henrit?. ?","- #henritofresh mÃ¡rquez (honda): 'la disputa con rossi ha sido lo mÃ¡s bonito de la carrera': el piloto espaÃ±ol... http://bit.ly/1ggugx0",24/03/2014 01:54,"http://bit.ly | | | | | | | | | | | | | | | |
| 6 | 643962,"sin padre","Henrito_fx3",,,,"Henrit?. ?","- #henritofresh pedrosa (honda): 'el tercer puesto ha sido un resultado positivo': el piloto espaÃ±ol de motogp... http://bit.ly/1ggugx1",24/03/2014 01:54,"http://bit.ly | | | | | | | | | | | | | | | |
| 7 | 643963,"sin padre","Henrito_fx3",,,,"Henrit?. ?","- #henritofresh pedrosa (honda): 'el tercer puesto ha sido un resultado positivo': el piloto espaÃ±ol de motogp... http://bit.ly/1ggugx1",24/03/2014 01:54,"http://bit.ly | | | | | | | | | | | | | | | |
| 8 | 724977,"sin padre","Nina1DHoran","marcmarquez93,valeyellow46",,"Nina","' #qatar #motogp was a hell of a battle between @marcmarquez93 & @valeyellow46! http://instagram.com/p/l5cnxevnxo/ http://t.co/aci | | | | | | | | | | | | | | | |
| 9 | 676025,"sin padre","gittguut","aan__",,"gitta puspita","--' '@aan__: do you know how valentino rossi trains? they release 50 mbak2 naik mio at the circuit and he has to survive all of'em.'",24/03/2014 13:29,0,"MT","ht | | | | | | | | | | | | | | | |
| 10 | 617815,"sin padre","AndreaVillena_5","marcmarquez93",,"Andrea Villena ","'' @marcmarquez93 :increÃ-ble, aÃºn no me lo creo! / incredible, i can't believe it! ",23/03/2014 23:49,0,"MT","http://twitter.com/avillena | | | | | | | | | | | | | | | |
| 11 | 588781,"sin ţ)",23/03/2014 21:54,0,"MT","http://twitter.com/carmensiitta93/statuses/447838857472770048",0,0,"http://twitter.com/carmensiitta93","es",1700,224,471,"not public" | | | | | | | | | | | | | | | |
| 12 | 736358,"sin padre","moto_2014","marcmarquez93",,"moto gp 2014","' @marcmarquez93: 'ojalÃ¡ tenga mÃ¡s batallas con valentinoâ€. #qatar014 . http://t.co/b189v9rcnf",25/03/2014 17:47,"http://t.co/b189v9rcnf","M | | | | | | | | | | | | | | | |
| 13 | 566848,"sin padre","VR_FortySix","rossistas,valeyellow46",,"Valent_Samawa","' @rossistas: rt si hoy .@valeyellow46 te ha emocionado. http://t.co/u8wybspvpn",23/03/2014 21:13,"http://t.co/u8wybspvpn","MT","h | | | | | | | | | | | | | | | |
| 14 | 736332,"sin padre","moto_2014","valeyellow46",,"moto gp 2014","' @valeyellow46:el motociclismo no es un deporte fÃ-sico como el atletismo, hay futbolistas muy buenos mÃ¡s viejos que http://t.co/hsxxsefgcm","' | | | | | | | | | | | | | | | |
| 15 | 516281,"sin padre","IrfanVale46",,,"Irfan Othman","- 1st esteve rabat - 2nd takaaki nakagami - 3rd mika kallio #moto2 @ losail international circuit, grandprix of qatar",23/03/2014 19:11,0,"MT","http://twitter.com/Irf | | | | | | | | | | | | | | | |
| 16 | 554225,"sin padre","IrfanVale46","valeyellow46,",,"Irfan Othman","- 1st marc marquez - 2nd irfan vale @valeyellow46 - 3rd dani pedrosa #motogp @ losail international circuit, grandprix of qatar",23/03/2014 20:48,0, | | | | | | | | | | | | | | | |
| 17 | 526021,"sin padre","robiangeli","marcmarquez93",,"Roberta","- 30 minuti e si parte con la #motogp :d buena suerte @marcmarquez93!!",23/03/2014 19:24,0,"MT","http://twitter.com/robiangeli/statuses/4478010712 | | | | | | | | | | | | | | | |
| 18 | 765188,"sin padre","MotoFamilyGP","jessansan,88jorgemartin,pitoslocosmclub,circuitodejerez,luissalom39",,"MotoFamily GP","- 39 y ya estamos en #jerez014. @jessansan @88jorgemartin @pitoslocosmclub @circ | | | | | | | | | | | | | | | |
| 19 | 681401,"sin padre","MotoFamilyGP","jessansan,88jorgemartin,pitoslocosmclub,circuitodejerez,aleixespargaro",,"MotoFamily GP","- 41 y ya estamos en #jerez014. @jessansan @88jorgemartin @pitoslocosmclub @c | | | | | | | | | | | | | | | |
| 20 | 605778,"sin padre","eternalcouple6","valeyellow46",,"EternalCouple","- dia perfecto, mi @valeyellow46 hace un carreron, y mi barÃ§a da un golpe en la liga ganando al madrid! vamoooooooooos!",23/03/2014 23:06,0 | | | | | | | | | | | | | | | |
| 21 | 567301,"sin padre","moto_2014",,,"moto gp 2014","' el campeÃ³n puede con el 'campeonÃ-simo': mÃ¡rquez se lleva el duelo con rossi http://mdia.st/olli9y http://t.co/logxxvxam7",23/03/2014 21:06,"http://mdia.st/ | | | | | | | | | | | | | | | |
| 22 | 605124,"sin padre","SERgete_",,,"Tara tachunda","- en una hora telecinco echa la carrera de #motogp, que emociÃ³n, quien ganarÃ¡? - mÃ¡rquez. - quÃ©? cÃ³mo lo sabes?",23/03/2014 23:08,0,"TW","http://twitter.co | | | | | | | | | | | | | | | |
| 23 | 582357,"sin padre","rubenT46",,,"ruben trigo cuellar","- fÃºtbol motor... esperando a las 00:00 para ver moto gp gas!!",23/03/2014 21:50,0,"TW","http://twitter.com/rubenT46/statuses/447837811987988480",0,0,"http | | | | | | | | | | | | | | | |
| 24 | 463879,"sin padre","ashifanurhaliza",,,"Wholocked","' go sleep now and wake up at 00.00, and you can watch motogp ' -dad . gracias, padre, you make my day.....",23/03/2014 12:29,0,"TW","http://twitter.com/ashifar | | | | | | | | | | | | | | | |

**Ilustración 93 - Muestra datos extraídos de Twitter.**

Muestra de una línea:

474676,"sin padre","MOTOaceleracion",,,"MOTOaceleracion","! en breve nos ponemos en modo warm up qatar, primero con #moto3 #moto2 y #motogp y luego las carreras!! http://fb.me/2kozivvsl",23/03/2014 14:38,"http://fb.me/2kozivvsl","TW","http://twitter.com/MOTOaceleracion/statuses/447729016066564096",1,0,"http://twitter.com/MOTOaceleracion","es",33946,555,527,"espaÃ±a"

**Descripción del DATASET**: Captura de datos del 23 al 26 de marzo. Contiene todo el ruido social generado en Twitter durante la carrera de Qatar Moto GP 2014 y los tres días posteriores.

**Número total de variables :** 19

**Descripción detallada de variables:**

- ID: Valor numérico N. Identificador de cada post.
- PARENT-SYS-ID: Valor numérico o nulo (="sin padre"). El valor es un ID que relaciona el post con aquel del cual depende por ser un retweet o una respuesta a un tweet
- Source: Nicks del usuario que emitió el post.
- Mentions: Nicks de usuarios de Twitter separados por comas presentes en el contenidos del mensaje. En definitiva son nicks de los usuarios retwitteados y/o mencionados.

- Target: Nick del usuario que es retwitteado.
- NAME Source: Nombre del usuario de Twitter que emitió el post.
- BODY: Contenido del mensaje.
- PUBDATE: Fecha y hora de emisión del post.
- URLs coma separated: URLs separadas por comas existentes en el contenido del mensaje.
- Type TW-RT-MT: Variable categorial. Valores:
    - TW: Tweet original
    - MT: Tweet que contiene una mención
    - RT: Tweet que es un retweet
- LINK: Link que da acceso al post de Twitter
- n1 Link: Variable categorial numérica. Valores:
    - 0: (cero) El contenido del mensaje no contiene ningún link
    - 1: (uno) El contenido del mensaje contiene algún link
- n1 Picture: Variable categorial numérica. Valores:
    - 0: (cero) El contenido del mensaje no contiene ningún link que sea una foto.
    - 1: (uno) El contenido del mensaje contiene algún link que es una foto
- PERSONAL-WEBSITE: Link a la página personal indicada en el perfil de Twitter del usuario o en su defecto link al perfil de Twitter.
- COUNTRY: Código del país donde el usuario reside
- ALL-NICK-ACTIVITY-EVER: Número de tweets emitidos por el perfil desde que el usuario creó la cuenta en Twitter.
- NICK-FOLLOWERS: Número de seguidores
- FRIENDS-FOLLOWING-AUDIENCE: Número de perfiles que sigue

- LOCATION: Variable de Geo localización. Si comienza por "ÜT", se proporcionan las coordenadas exactas desde donde se emitió el tweet. En su defecto contiene la ubicación declarada por el perfil.

Una vez identificadas todas estas variables, lo que se va a hacer es procesar este dataset con Spark, para ello se debe introducir en el clúster y subir el fichero a HDFS:

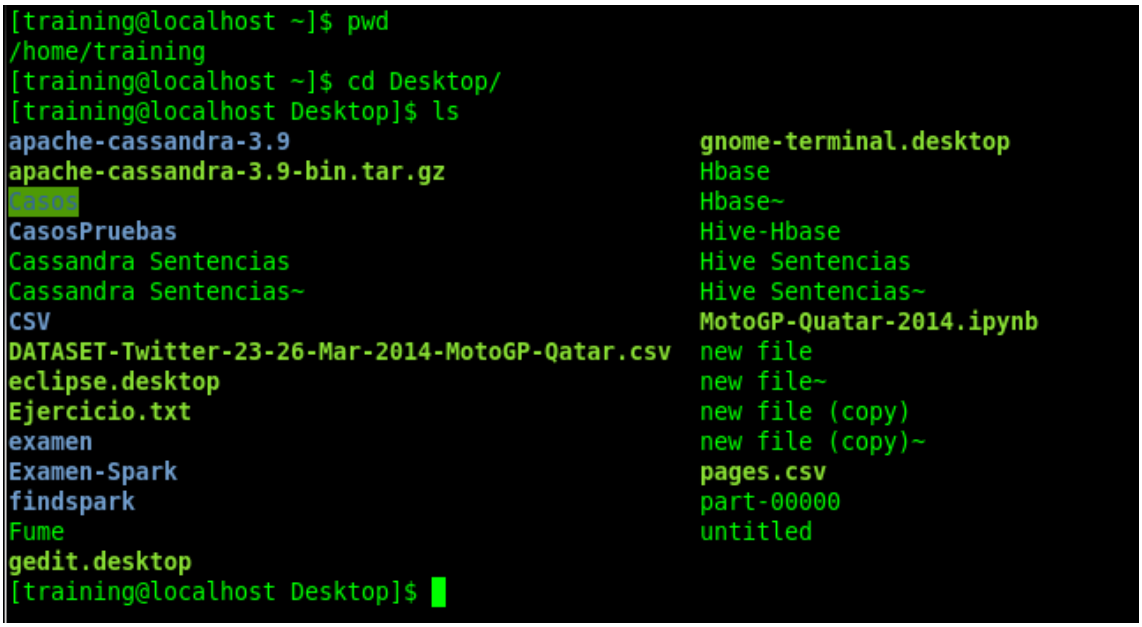



Ilustración 94 - Interacción HDFS.

Se ha localizado nuestro fichero en la ruta : /home/training/Desktop

Con esta linea de comando: hdfs dfs -ls /

Se ven todos los directorios que se tienen dentro de HDFS, se crea uno para subir nuestro DATASET y luego se llama con SPARK. Se crea la carpeta TFG, y se sube.

```
[training@localhost Desktop]$ hdfs dfs -mkdir /tfg
[training@localhost Desktop]$ hdfs dfs -ls /
Found 9 items
drwxrwxrwx   - training supergroup          0 2017-03-28 04:02 /casos
drwxrwxrwx   - training supergroup          0 2016-11-04 05:45 /ejerciciospark
drwxrwxrwx   - hbase    supergroup          0 2016-10-28 16:11 /hbase
drwxrwxrwx   - training supergroup          0 2017-03-22 02:07 /loudacre
drwxrwxrwx   - solr     supergroup          0 2015-08-27 05:32 /solr
drwxrwxrwx   - training supergroup          0 2017-09-03 14:14 /tfg
drwxrwxrwt   - hdfs     supergroup          0 2017-03-29 01:52 /tmp
drwxrwxrwx   - hdfs     supergroup          0 2016-10-28 05:15 /user
drwxrwxrwx   - hdfs     supergroup          0 2015-08-27 05:31 /var
[training@localhost Desktop]$ hdfs dfs -put /home/training/Desktop/DATASET-Twitt
er-23-26-Mar-2014-MotoGP-Qatar.csv /tfg
[training@localhost Desktop]$ hdfs dfs -ls /tfg
Found 1 items
-rw-rw-rw-   1 training supergroup   90724255 2017-09-03 14:15 /tfg/DATASET-Twit
ter-23-26-Mar-2014-MotoGP-Qatar.csv
[training@localhost Desktop]$
```

Ilustración 95 - Interacción HDFS.

Como se puede observar ya se tiene en HDFS nuestro fichero insertado.

Lo siguiente será lanzar Spark, para ello en la terminal se escribe pyspark.

```
[training@localhost findspark]$ pyspark
[I 14:17:40.905 NotebookApp] Using MathJax:
[I 14:17:40.926 NotebookApp] Serving notebooks from local directory: /home/train
ing/Desktop/findspark
[I 14:17:40.926 NotebookApp] 0 active kernels
[I 14:17:40.926 NotebookApp] The IPython Notebook is running at: http://127.0.0.
1:3333/
[I 14:17:40.926 NotebookApp] Use Control-C to stop this server and shut down all
 kernels (twice to skip confirmation).
[W 14:17:41.995 NotebookApp] 404 GET /api/kernels/3deb3b86-1cb8-430f-8f16-c9b970
c6d24d/channels?session_id=BEBA622E90A44457879DB8A6407BFA67 (127.0.0.1): Kernel
does not exist: 3deb3b86-1cb8-430f-8f16-c9b970c6d24d
[W 14:17:42.025 NotebookApp] 404 GET /api/kernels/3deb3b86-1cb8-430f-8f16-c9b970
c6d24d/channels?session_id=BEBA622E90A44457879DB8A6407BFA67 (127.0.0.1) 78.27ms
referer=None
```

Ilustración 96 - Interacción PySpark.

Lo que se consigue con esto es abrir Spark con Python, sino la forma corriente sería escribiendo Spark-shell.

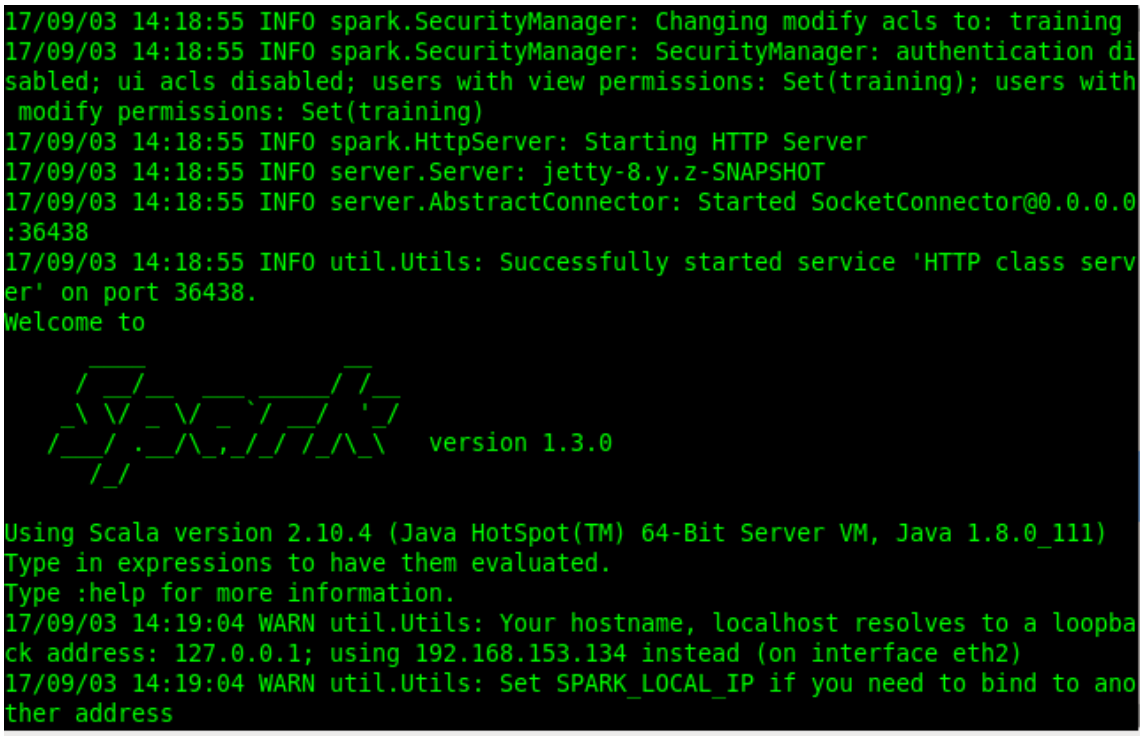



Ilustración 97 - Spark-shell.

La diferencia entre usar uno u otro es que Spark normal como ya se comentó funciona con el lenguaje de programación Scala, y pyspark funciona con Python , en este caso se va a utilizar con Python porque posee un túnel a una interfaz web la cual va a ir sacando el resultado a medida que se vayan haciendo las operaciones , lo que en este caso va a resultar más interesante.

El túnel que se crea es en la dirección http://127.0.0.1:3333

**Resultado Obtenido:**

# Datos MotoGP Qatar 2014 - Luis Nueda

In [2]:
```
import findspark
findspark.init("/home/luis/Development/spark/spark-2.0.2-bin-hadoop2.7/")

import pyspark
from pyspark.sql import SparkSession
spark = (SparkSession.builder
    .master("local[*]")
    .config("spark.driver.cores", 1)
    .appName("understanding_sparksession")
    .getOrCreate() )
sc = spark.sparkContext
print(spark)
print(sc)
```

```
<pyspark.sql.session.SparkSession object at 0x7f6db45b91d0>
<pyspark.context.SparkContext object at 0x7f6dd41cfb38>
```

Lectura de eventos, en primer lugar, se definen los tipos de datos específicos para cada campo con un esquema personalizado.

In [3]:
```
from pyspark.sql.types import *
from pyspark.sql.functions import *
customSchema = StructType([StructField("Id", LongType(), True),
                           StructField("Parent_sys_id", StringType(), True),
                           StructField("Source", StringType(), True),
                           StructField("Mentions", StringType(), True),
                           StructField("Target", StringType(), True),
                           StructField("Name_source", StringType(), True),
                           StructField("Body", StringType(), True),
                           StructField("Pub_date", TimestampType(), True),
                           StructField("URLs", StringType(), True),
                           StructField("Tipe_action", StringType(), True),
                           StructField("Link", StringType(), True),
                           StructField("Has_link", ByteType(), True),
                           StructField("Has_picture", ByteType(), True),
                           StructField("Website", StringType(), True),
                           StructField("Country", StringType(), True),
                           StructField("Activity", LongType(), True),
                           StructField("Followers", LongType(), True),
                           StructField("Following", LongType(), True),
                           StructField("Location", StringType(), True)
                          ])
```

A continuación, se usa el esquema personalizado para la lectura de los datos. Hay que tener en cuenta que puede que haya campos con datos mal formados (malformed data lines).

En la página de documentación del método read.csv (DataFrameReader csv), el modo por defecto de lectura de los datos es PERMISSIVE. Este modo pone a null todos los campos del

esquema que encuente para un campo con valor corrupto, y puede llegar a hacer que no se lea alguna línea completa del archivo. En este caso particular, en el campo del texto del tweet aparecen en ciertas ocasiones el caracter \ como parte de emoticonos textuales o por otros motivos, justo al final del string del campo. Eso puede hacer que el parser se confunda, interpretando erróneamente \" como un intento de escapar la comilla doble, cuando en realidad se trata del caracter \ seguido de la comilla de cierre del campo de texto. También se puede controlar en la exportación de datos que se escape explicitamente cualquier carácter especial como \, pero este posible fallo nos da la oportunidad de usar un mecanismo para resolverlo.

Si se cambia el modo de lectura a mode="FAILFAST", eso hará que el parser falle de inmediato ante una línea con campos corruptos (respecto del esquema que se ha proporcionado). Buscando en la traza de error, se ve la línea que produjo la excepción y se puede buscar en el fichero de datos para solucionar el error.

```
In [4]: events = spark.read.csv("data/ALTO-DATABASE/DATASET-Twitter-23-26-Mar-2014-MotoGP-Qatar.csv",
                                header=True, schema=customSchema, timestampFormat="dd/MM/yyyy HH:mm")
                                #mode="FAILFAST") #
```

- Se imprime el esquema creado para el conjunto de datos:

```
In [5]: events.printSchema()

root
 |-- Id: long (nullable = true)
 |-- Parent_sys_id: string (nullable = true)
 |-- Source: string (nullable = true)
 |-- Mentions: string (nullable = true)
 |-- Target: string (nullable = true)
 |-- Name_source: string (nullable = true)
 |-- Body: string (nullable = true)
 |-- Pub_date: timestamp (nullable = true)
 |-- URLs: string (nullable = true)
 |-- Tipe_action: string (nullable = true)
 |-- Link: string (nullable = true)
 |-- Has_link: byte (nullable = true)
 |-- Has_picture: byte (nullable = true)
 |-- Website: string (nullable = true)
 |-- Country: string (nullable = true)
 |-- Activity: long (nullable = true)
 |-- Followers: long (nullable = true)
 |-- Following: long (nullable = true)
 |-- Location: string (nullable = true)
```

- Se hace un conteo del numero total de datos:

**Conjunto de datos y valores**

```
In [6]: events.count()
Out[6]: 257680
```

- Se cogen las 5 primeras lineas:

```
In [7]: events.take(5)
Out[7]: [Row(Id=600621, Parent_sys_id='sin padre', Source='Henrito_fx3', Mentions=None,
 #henritofresh alex márquez (honda): 'he cometido un error de novato': el pilot
olxd7p", Pub_date=datetime.datetime(2014, 3, 23, 22, 32), URLs='http://bit.ly/c
om/Henrito_fx3/statuses/447848348674838531', Has_link=1, Has_picture=0, Website
 public', Activity=82710, Followers=1485, Following=477, Location='???????? '),
 Row(Id=604076, Parent_sys_id='sin padre', Source='Henrito_fx3', Mentions=None,
 #henritofresh con márquez, todo igual: los focos, puestos en él desde que debu
1deycvz', Pub_date=datetime.datetime(2014, 3, 23, 22, 48), URLs='http://bit.ly/
r.com/Henrito_fx3/statuses/447852358957944834', Has_link=1, Has_picture=0, Webs
ot public', Activity=82710, Followers=1485, Following=477, Location='????????
 Row(Id=512641, Parent_sys_id='sin padre', Source='Henrito_fx3', Mentions=None,
 #henritofresh en vivo: carrera de moto2: ... http://bit.ly/1hbliq7', Pub_date=
tp://bit.ly/1hbliq7', Tipe_action='TW', Link='http://twitter.com/Henrito_fx3/st
ure=0, Website='http://twitter.com/Henrito_fx3', Country='not public', Activity
='???????? '),
 Row(Id=643917, Parent_sys_id='sin padre', Source='Henrito_fx3', Mentions=None,
 #henritofresh márquez (honda): 'la disputa con rossi ha sido lo más bonito de
1ggugx0", Pub_date=datetime.datetime(2014, 3, 24, 1, 54), URLs='http://bit.ly/1
com/Henrito_fx3/statuses/447899325058670592', Has_link=1, Has_picture=0, Websit
```

Resumen de datos en DataFrames.

- Timestamp y contenido del último tweet.

```
In [99]: max_ts = events.agg(max("Pub_date")).collect()[0][0]
(events.select("Pub_date","Body")
 .filter(events.Pub_date == max_ts)
 .show())
```

```
+--------------------+--------------------+
|            Pub_date|                Body|
+--------------------+--------------------+
|2014-03-26 14:15:...|@marcmarquez93 go...|
|2014-03-26 14:15:...|@marcmarquez93 si...|
|2014-03-26 14:15:...|rt @_supportvale4...|
|2014-03-26 14:15:...|rt @jessansan: es...|
|2014-03-26 14:15:...|rt @marcmarquez93...|
|2014-03-26 14:15:...|rt @marcmarquez93...|
|2014-03-26 14:15:...|rt @marcmarquez93...|
|2014-03-26 14:15:...|rt @marcmarquez93...|
|2014-03-26 14:15:...|rt @marcmarquezte...|
|2014-03-26 14:15:...|rt @marcmarquezte...|
|2014-03-26 14:15:...|rt @motogp: rt @m...|
|2014-03-26 14:15:...|rt @motogp: so wh...|
|2014-03-26 14:15:...|rt @motogp: so wh...|
|2014-03-26 14:15:...|rt @motogp: so wh...|
|2014-03-26 14:15:...|rt @motogp: so wh...|
|2014-03-26 14:15:...|rt @motorpasion: ...|
|2014-03-26 14:15:...|rt @scottredding4...|
+--------------------+--------------------+
```

- Usuario con más followers:

```
In [20]: max_followers = events.agg(max("Followers")).collect()[0][0]
(events.select("Source", "Followers")
 .filter(events.Followers == max_followers)
 .distinct().show())
```

```
+------------+---------+
|      Source|Followers|
+------------+---------+
|justinbieber| 51669863|
+------------+---------+
```

- Cinco países con más tweets:

```
In [92]: (events.filter(events.Country != "not public")
          .groupBy("Country")
          .agg(count("Id").alias("tweets"))
          .orderBy("tweets", ascending=False)
          .limit(5).show())
```

```
+-------+------+
|Country|tweets|
+-------+------+
|     es|172577|
|     us| 12722|
|     gb| 12588|
|     id|  8725|
|     it|  1843|
+-------+------+
```

- Diez usuarios con más tweets:

```
In [97]: (events
          .groupBy("Source")
          .agg(count("Id").alias("tweets"))
          .orderBy("tweets", ascending=False)
          .limit(10).show())
```

```
+------------+------+
|      Source|tweets|
+------------+------+
|  m_azharaji|   486|
|  twitMOTOGP|   401|
|   johnbokke|   297|
|qatarflights|   283|
|  box_repsol|   267|
| yolandaa_95|   185|
| AlessiaPont|   182|
| motomatters|   169|
|  MM93Lovers|   169|
|  Sonic_Moto|   165|
+------------+------+
```

- Número de tweets por día:

**Número de tweets por día**

```
In [88]: (events.groupBy(dayofmonth("Pub_date").alias("day_month"))
          .agg(count("Id").alias("tweets"))
          .orderBy("day_month").show())
```

```
+---------+------+
|day_month|tweets|
+---------+------+
|       23|154395|
|       24| 67945|
|       25| 20557|
|       26| 14783|
+---------+------+
```

- Número de tweets por día y hora:

```
In [31]: (events.select("Id", "Pub_date")
          .groupBy(dayofmonth("Pub_date").alias("day_month"),
                   hour("Pub_date").alias("hour"))
          .agg(count("Id").alias("tweets"))
          .orderBy("day_month", "hour").show())
```

```
+---------+----+------+
|day_month|hour|tweets|
+---------+----+------+
|       23|   0|  1628|
|       23|   1|  1216|
|       23|   2|   960|
|       23|   3|   751|
|       23|   4|   595|
|       23|   5|   427|
|       23|   6|   503|
|       23|   7|   455|
|       23|   8|   971|
|       23|   9|  1355|
|       23|  10|  2080|
|       23|  11|  4733|
|       23|  12|  5194|
|       23|  13|  3745|
|       23|  14|  3436|
|       23|  15|  3645|
|       23|  16|  6977|
|       23|  17| 14959|
|       23|  18|  7289|
|       23|  19| 11940|
+---------+----+------+
only showing top 20 rows
```

- Número de tweets por día y hora, a partir de 2014-03-24 05:00 :

```
In [36]: (events.select("Id", "Pub_date")
          .filter(( (dayofmonth("Pub_date") == 24) & (hour("Pub_date") >= 5)) | (dayofmonth("Pub_date") >= 25) )
          .groupBy(dayofmonth("Pub_date").alias("day_month"),
                   hour("Pub_date").alias("hour"))
          .agg(count("Id").alias("tweets"))
          .orderBy("day_month", "hour").show(30) )
```

```
+---------+----+------+
|day_month|hour|tweets|
+---------+----+------+
|       24|   5|   970|
|       24|   6|  1162|
|       24|   7|  1831|
|       24|   8|  1852|
|       24|   9|  1774|
|       24|  10|  2415|
|       24|  11|  3376|
|       24|  12|  2590|
|       24|  13|  5332|
|       24|  14|  3984|
|       24|  15|  2903|
|       24|  16|  2277|
|       24|  17|  2678|
|       24|  18|  1830|
|       24|  19|  1597|
|       24|  20|  1746|
|       24|  21|  2303|
|       24|  22|  1679|
|       24|  23|  1124|
|       25|   0|  1002|
|       25|   1|   582|
|       25|   2|   382|
|       25|   3|   369|
|       25|   4|   348|
|       25|   5|   340|
```

Resumen de datos con SparkSQL

- Timestamp y contenido del último tweet:

```
In [12]: spark.sql("""
         SELECT Pub_date, Body
         FROM events
         WHERE Pub_date = (SELECT MAX(Pub_date) from events)
         LIMIT 5
         """).show()
```

```
+--------------------+--------------------+
|            Pub_date|                Body|
+--------------------+--------------------+
|2014-03-26 14:15:...|@marcmarquez93 go...|
|2014-03-26 14:15:...|@marcmarquez93 si...|
|2014-03-26 14:15:...|rt @_supportvale4...|
|2014-03-26 14:15:...|rt @jessansan: es...|
|2014-03-26 14:15:...|rt @marcmarquez93...|
+--------------------+--------------------+
```

- Usuario con más followers:

```
In [86]: spark.sql("""
         SELECT DISTINCT(Source), Followers
         FROM events
         WHERE Followers = (SELECT MAX(Followers) FROM events)
         """).show()
```

```
+------------+---------+
|      Source|Followers|
+------------+---------+
|justinbieber| 51669863|
+------------+---------+
```

- Cinco países con más tweets:

```
In [94]: spark.sql("""
         SELECT Country, COUNT(Id) AS `tweets`
         FROM events
         WHERE Country != "not public"
         GROUP BY Country
         ORDER BY tweets DESC
         LIMIT 5
         """).show()
```

```
+-------+------+
|Country|tweets|
+-------+------+
|     es|172577|
|     us| 12722|
|     gb| 12588|
|     id|  8725|
|     it|  1843|
+-------+------+
```

- Diez usuarios con más tweets:

```
In [95]: spark.sql("""
SELECT Source, COUNT(Id) AS `tweets`
FROM events
GROUP BY Source
ORDER BY tweets DESC
LIMIT 10
""").show()
```

```
+------------+------+
|      Source|tweets|
+------------+------+
|  m_azharaji|   486|
|  twitMOTOGP|   401|
|   johnbokke|   297|
|qatarflights|   283|
|  box_repsol|   267|
| yolandaa_95|   185|
| AlessiaPont|   182|
|  MM93Lovers|   169|
| motomatters|   169|
|  Sonic_Moto|   165|
+------------+------+
```

- Número de tweets por día

```
In [27]: spark.sql("""
SELECT DAYOFMONTH(Pub_date) AS `day_month`,
COUNT(id) AS `tweets`
FROM events
GROUP BY DAYOFMONTH(Pub_date)
ORDER BY DAYOFMONTH(Pub_date)
""").show()
```

```
+---------+------+
|day_month|tweets|
+---------+------+
|       23|154395|
|       24| 67945|
|       25| 20557|
|       26| 14783|
+---------+------+
```

- Número de tweets por día y hora:

```
In [26]: spark.sql("""
SELECT DAYOFMONTH(Pub_date) AS `day_month`, HOUR(Pub_date) as `hour`,
COUNT(id) AS `tweets`
FROM events
GROUP BY DAYOFMONTH(Pub_date), HOUR(Pub_date)
ORDER BY DAYOFMONTH(Pub_date), HOUR(Pub_date)
""").show()
```

```
+---------+----+------+
|day_month|hour|tweets|
+---------+----+------+
|       23|   0|  1628|
|       23|   1|  1216|
|       23|   2|   960|
|       23|   3|   751|
|       23|   4|   595|
|       23|   5|   427|
|       23|   6|   503|
|       23|   7|   455|
|       23|   8|   971|
|       23|   9|  1355|
|       23|  10|  2080|
|       23|  11|  4733|
|       23|  12|  5194|
|       23|  13|  3745|
|       23|  14|  3436|
|       23|  15|  3645|
|       23|  16|  6977|
|       23|  17| 14959|
|       23|  18|  7289|
|       23|  19| 11940|
+---------+----+------+
only showing top 20 rows
```

- Número de tweets por día y hora, a partir de 2014-03-24 05:00 :

```
In [33]: spark.sql("""
SELECT DAYOFMONTH(Pub_date) AS `day_month`, HOUR(Pub_date) as `hour`,
COUNT(id) AS `tweets`
FROM events
WHERE ((DAYOFMONTH(Pub_date) = 24 AND HOUR(Pub_date) >= 5) OR DAYOFMONTH(Pub_date) >= 25)
GROUP BY DAYOFMONTH(Pub_date), HOUR(Pub_date)
ORDER BY DAYOFMONTH(Pub_date), HOUR(Pub_date)
""").show(30)
```

```
+---------+----+------+
|day_month|hour|tweets|
+---------+----+------+
|       24|   5|   970|
|       24|   6|  1162|
|       24|   7|  1831|
|       24|   8|  1852|
|       24|   9|  1774|
|       24|  10|  2415|
|       24|  11|  3376|
|       24|  12|  2590|
|       24|  13|  5332|
|       24|  14|  3984|
|       24|  15|  2903|
|       24|  16|  2277|
|       24|  17|  2678|
|       24|  18|  1830|
|       24|  19|  1597|
|       24|  20|  1746|
|       24|  21|  2303|
|       24|  22|  1679|
|       24|  23|  1124|
|       25|   0|  1002|
|       25|   1|   582|
|       25|   2|   382|
```

# 7. CONCLUSIÓN

Como ocurre con cualquier tecnología, se debe hacer una pregunta ¿realmente se necesita Big Data? A pesar de todo lo explicado a lo largo del TFG, la respuesta es, depende. Eso no quiere decir que Big Data sea valioso o no, sólo que las organizaciones deben tener cuidado al decidir si Big Data es adecuado para ellos. Eso sí, bien empleada se trata de un arma competitiva muy poderosa respecto a la competencia.

Aquí hay algunas preguntas que se puede hacer al intentar tomar la decisión. ¿Los datos son lo suficientemente grandes? Al igual que con la tecnología tradicional, el punto de Big Data es utilizar los datos para responder a las preguntas. Así que al tomar una decisión, se necesita empezar por averiguar qué preguntas se quieren responder, así como qué datos se tienen o se pueden acceder. Por ejemplo, si todo lo que se necesita saber es quién está comprando qué y dónde, probablemente no se necesita Big Data, los datos estructurados en una base de datos tradicional harán el trabajo. Además, se recuerda que el volumen no es el único atributo clave en la definición de Big Data. Tener un montón de datos estructurados no lo hacen grande. ¿estamos preparados? Prepararse para utilizar Big Data requiere algo más que instalar software, se necesita estar listos para que su implementación tenga éxito. Eso incluye hardware tanto como otras consideraciones. Hadoop es conocido por ser capaz de funcionar con hardware estándar de la industria, sin embargo, las empresas que quieran encontrar sus propias metas requerirán diferentes configuraciones de hardware. Además las tecnologías cloud computing han llegado en el momento oportuno para abaratar los costes de hardware, seguridad y conectividad. También se debe asegurar de que se tienen los procesos listos para recopilar los datos correctos, asegurando su integridad y manteniendo el gobierno de los datos. ¿ expectativas realistas? Las empresas también deben tener las expectativas correctas en cuanto a cuánto tiempo abarca un proyecto de Big Data y cuánto costará. Ése es a menudo uno de los desafíos más grandes. La mayoría de los líderes empresariales creen que el Big Data puede ayudar a tomar decisiones mejores y más informadas, según una encuesta de IDG Research. Sin embargo, sólo el 23% de los encuestados dice que sus proyectos de Big Data han tenido éxito completo, mientras que el 52% dice que ha tenido cierto éxito.

Como conclusión y bajo mi punto de vista, el exponencial crecimiento de Big Data es independiente de la existencia de Hadoop, pero sin este software de código abierto resulta difícil, si no imposible, concebir tanto un almacenamiento, como un procesamiento y extracción de valor de los grandes datos a un bajo coste.

Para analizar Big Data sin Hadoop, es decir, para aprovechar las ventajas estratégicas que ello supone para la ciencia y también para las organizaciones en general, habría que buscar otra tecnología que permitiera hacerlo de un modo eficiente. O quizá deberíamos decir mejor que habríamos de crearla, aunque a buen seguro sería complicado que pudiera ofrecer todas sus ventajas. No en vano, la arquitectura de Hadoop tiene unas características que se adaptan a la perfección a las necesidades del universo Big Data, tanto para su almacenamiento como para permitir el intercambio de archivos y la posibilidad de llevar a cabo análisis de datos heterogéneos de forma rápida, flexible, escalable, a bajo coste y resistente a fallos.

# 8. PRESUPUESTO

Se recomienda comenzar el proyecto (primeros pilotos y funcionalidades) sobre la nube de Azure, desplegando la distribución Open Source de Cloudera. Es posible su implementación tanto en la nube de Azure, cómo en el CPD Central (On Premise), teniendo en cuenta las siguientes consideraciones:

| | Ventajas | Inconvenientes |
|---|---|---|
| Cloud | • Rápida puesta en marcha sin depender de aprovisionamiento Hardware.<br>• Capacidad de escalado casi infinita de forma ágil.<br>• Ideal para disponer de entornos previos a Producción, donde las exigencias de rendimiento no son tan altas (sobre todo para el entorno de Desarrollo). | • Mayores costes de mantenimiento de infraestructura ante grandes volumetrías.<br>• Menor rendimiento, si se trabaja con volúmenes de almacenamiento no locales a los nodos (workers). |
| On Premises | • Permite diseñar el cluster ad-hoc de forma optimizada para los requerimientos y volumetrías necesarios.<br>• Permite ofrecer mejor rendimiento al poder escojer componentes hardware high-end.<br>• Mayores capacidades de integración con otros sistemas y con mejor rendimiento. | • Inversión en Hardware.<br>• Tiempos de disposición de Hardware y set-uo.<br>• Costes de mantenimiento del Cluster. |

Ilustración 98 - Cloud Vs. On Premises

Más adelante, se podrá valorar si se migra el Clúster a una instalación On Premises sobre la misma distribución, lo que tendrá cero impacto en los desarrollos realizados. A continuación, se ilustra un dimensionamiento tentativo para un primer escenario en la nube con capacidades de albergar considerables volúmenes de información y procesarlos ágilmente, aplicando analítica avanzada con Spark.

| 2x Manager | | |
|---|---|---|
| On Premise | | |
| O.S. | RHEL 6.7 | |
| Cores | 16 | |
| RAM | 16 GB | |
| S.O. + Logs | RAID1 | 2x250GB |
| PostgreSQL | RAID1 | 2x200GB |
| Conectividad | 10 Gb/s | |

| 3x Master | | |
|---|---|---|
| On Premise | | |
| O.S. | RHEL 6.7 | |
| Cores | 16 | |
| RAM | 32 GB | |
| S.O | RAID1 | 2x50GB |
| Masters + Logs | RAID1 | 2x500GB |
| Zookeeper | RAID0 | 500GB |
| Conectividad | 10 Gb/s | |

| 3x Worker | | |
|---|---|---|
| On Premise | | |
| O.S. | RHEL 6.7 | |
| Cores | 24 | |
| RAM | 256 GB | |
| S.O. + logs | RAID1 | 2x500GB |
| Data | JBOD | 8x2 TB |
| Conectividad | 10 Gb/s | |
| Número nodos | 3 | |
| HDFS Space | 40 TB | |

| 2x Ingest | | |
|---|---|---|
| On Premise | | |
| O.S. | RHEL 6.7 | |
| Cores | 16 | |
| RAM | 32 GB | |
| S.O. + Data | RAID1 | 2x500GB |
| Conectividad | 10 Gb/s | |

Ilustración 99 - Esquema de las máquinas.

Una vez se han estimado aproximadamente los costes mensuales que supondría tener el clúster hadoop en la nube de Azure. Se han mapeado las características a nivel de infraestructura con la disponibilidad que ofrece Azure en la web.

Se han realizado las siguientes acciones para reducir costes:

- Unificación de las máquinas de Ingesta y Manager .
- Reducción de una de las máquinas Master.
- Se ha seleccionado el tipo de disco Standard, lo que implica que se tendrá que pagar por cada operación de lectura/escritura sobre ellos. Con los discos SSD no ahorraríamos este coste, son unas 7 veces más caros.
- Se ha verificado que no hay coste de subida de datos a Azure pero si al acceder a la información y al operar sobre los discos.
- Sería necesario saber el volumen de información con la que se va a trabajar y el número de operaciones estimadas sobre los datos.

| NODOS | COMPONENTE | CANTIDAD | CORES | RAM (GB) | DISCO (GB) | TIPO | PRECIO (€) | PERIODO |
|---|---|---|---|---|---|---|---|---|
| MASTER + INGESTA | MAQUINA | 2 | 16 | 32 | 256 | N/A | 2.174,22 | MENSUAL |
| MANAGER | | 2 | 16 | 32 | 256 | | 2.174,22 | |
| WORKERS | | 3 | 16 | 112 | 800 | | 4.061,26 | |
| MASTER + INGESTA | MANAGED DISK | 10 | N/A | N/A | 512 | STANDARD | 91,75 | |
| MANAGER | | 12 | | | 512 | | 110,1 | |
| WORKERS | | 6 | | | 512 | | 55,05 | |
| | | 24 | | | 1024 | | 414,5 | |
| | BLOB EN BLOQUES | 3 | | | 40960 | | 2.279,73 | |
| | | | | | | TOTAL | 11.360,83 | |

Los precios que se detallan son sin IVA. Como se observa en la tabla se tiene dos tipos de configuración y precio, para distinguirla hay que fijarse en la columna componente, la cual estará formada por la configuración a nivel On Premise ( Máquina ) y a nivel Cloud ( Managed Disk ) que a su vez tiene dos tipos de almacenamiento con discos duros tradicionales, o con blob en bloques. La elección del tipo de almacenaje dependerá del tipo de proyecto, para proyectos en tiempo real se recomienda elegir blob en bloques.

# 9. GLOSARIO

- **Algoritmo**: Secuencia lógica, finita y con instrucciones que forman una fórmula matemática o estadística para realizar el análisis de datos.
- **HDFS**: Sistema de ficheros que distribuye los datos en múltiples nodos. Este sistema implementa redundancia y tolerancia frente a fallos. Además no depende de Hardware específico.
    - Los componentes de HDFS son:
        - **Name Node**: que gestiona la estructura del sistema de ficheros. Puede estar redundado en Secundary Name Node.
        - **Data Nodes**: que almacenan bloques de los ficheros. Por defecto los bloques se encuentran replicados en 3 Data Nodes.
- **MapReduce**: Tecnología de acceso y procesado distribuido de los datos. Accede a los datos mediante el procesado de los datos en cada nodo (**Map**) y la unificación de estos datos procesados de los diferentes nodos (**Reduce**).
    - Los componentes de MapReduce son:
        - **Job Tracker** que gestiona el procesado de las tareas.
        - **TaskTracker** realizan las tareas.
- **YARN**: Es un motor gestión de recursos y servicios que se incorpora Hadoop 2.0. Mejora la eficiencia de MapReduce y además permite poner otros motores de procesado distribuido de datos más eficientes como Tez.
- **HIVE**: Ejecuta comandos de Sql (HiveQL) que generan el proceso de MapReduce.
- **Sqoop**: Transfiere los datos de HDFS desde/hacia bases de datos.
- **Flume**: Transfiere los datos de Streaming hacia HDFS.
- **Worker Nodes**: nodos encargados de realizar las operaciones.
- **SparkContext**: coordina los Worker Nodes.

- **RDD**: colecciones de datos distribuidos en memoria o disco basados en HDFS o colecciones.

- **Operaciones:** Es el equivalente a MapReduce. El procesado Map se llama Tranformación y nos retorna un RDD. El procesado Reduce se llama Acción y nos retorna el resultado al SparkContext o a un fichero.

- **Latencia**: medida de tiempo demorado en un sistema Sistema Legacy.

- **Terabytes**: aproximadamente 1.000 gigabytes. Un terabyte puede almacenar hasta 300 horas de vídeo de alta definición Análisis de series de tiempo.

- **Variabilidad**: el significado de los datos puede cambiar (rápidamente). En los mismos tuits, por ejemplo una palabra puede tener un significado totalmente diferente.

- **Variedad**: datos que vienen en muchos formatos diferentes: datos estructurados, datos semi-estructurados y datos no estructurados.

- **Velocidad**: la velocidad a la cual se crea el dato, almacena, analiza y visualiza.

- **Veracidad**: los datos son correctos, así como los análisis realizados en los datos. La veracidad se refiere a la exactitud de los datos.

- **Visualización**: con las visualizaciones adecuadas, los datos brutos pueden ser objeto de un uso. Las visualizaciones por supuesto no significan gráficas ordinarias o gráficos circulares. Significan gráficos complejos que pueden incluir muchas variables de datos sin dejar de ser comprensible y legible.

- **Volumen**: la cantidad de datos.

- **Analytics**: Es la forma de capturar informaciones, procesarlas y analizarlas para que se conviertan en conocimiento.

- **Data Scientist**: Es el analista de datos. La persona que capturará la principal información dentro de un gran volumen de información.

- **Datos estructurados y no estructurados**: Los estructurados tienen una organización lógica, por otro lado, los no estructurados son desorganizados, como los mensajes en emails y redes sociales.

- **REHL:** sistema operativo , Red Hat Enterprise.

- **Blobs en bloques:** Optimizados para el streaming y para el almacenamiento de objetos en la nube, constituyen una opción idónea para almacenar documentos, archivos multimedia y copias de seguridad.

- **Managed disk:** Almacenamiento de disco persistente y seguro para Azure Virtual Machines.

- **Big Data**: es la expresión utilizada para designar un conjunto de datos tan grande que es difícil trabajar con los medios habituales (bases de datos). Se suele decir que el Big Data responde a las tres V: volumen, variedad y velocidad.

- **Clúster** : Conjunto de servidores (o nodos) que permiten garantizar la continuidad del servicio y distribuir la carga de procesamiento/red.

- **Hadoop:** Framework de aplicaciones distribuidas de Java de código abierto, destinado a procesar volúmenes de datos de varios petabytes y con miles de nodos.

- **RAID:** Redundant Array of Independent/Inexpensive Disks, tecnología que permite utilizar varios discos duros en paralelo.

- **RAM**: Es el acrónimo de «Random Access Memory». Designa la memoria de acceso aleatorio.

- **Rack virtual**: Tecnología de OVH que permite reunir virtualmente varios servidores (independientemente de su número y su localización física en nuestros datacenters) y conectarlos a un switch virtual dentro de una misma red privada. De este modo, sus servidores pueden comunicarse de manera privada y segura entre ellos (dentro de una VLAN dedicada).

- **Replicación**: Procedimiento para compartir archivos que permite mejorar la fiabilidad y limitar la tolerancia a fallos.

- **Root**: es la palabra inglesa para «raíz». Tener acceso «root» significa tener acceso a la raíz del servidor y tener todos los permisos de administración sin restricciones. También implica que es responsable de todas las acciones que se realicen con esos permisos en el servidor.

- **SQL/NoSQL** : El Structured Query Language es el lenguaje informático más utilizado para estructurar bases de datos y realizar consultas para extraer la información. Estos últimos años ha aparecido un nuevo sistema de bases de datos «noSQL», que se distingue por tener una mayor flexibilidad de las bases de datos y una arquitectura de cluster.

- **VLAN**: «Virtual Local Area Network»: red local virtual.

# 10. BIBLIOGRAFÍA


- https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html
- https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html#NameNode+and+DataNodes
- https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html#Introduction
- https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html#Data+Replication
- http://www.monografias.com/trabajos/histocomp/histocomp.shtml#ixzz4i0uPY6mm
- https://sg.com.mx/buzz/big-data-el-desaf%C3%ADo-la-explosi%C3%B3n-datos#.WTpmjOvyjIV
- https://www.ibm.com/developerworks/ssa/local/im/que-es-big-data/
- http://hadoop.apache.org/
- https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html#Data+Organization
- http://shop.oreilly.com/product/0636920025085.do
- https://es.hortonworks.com/apache/hadoop/
- https://www.slideshare.net/mulesoft/welcome-to-the-api-economy-developing-your-api-strategy
- https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html#Cluster+Rebalancing
- http://research.google.com/archive/gfs.html
- http://www.w3resource.com/mongodb/nosql.php
- http://www.monografias.com/trabajos/histocomp/histocomp.shtml#ixzz4i0uTiZJD
- https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html
- https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html#Prerequisites
- http://shop.oreilly.com/product/0636920033448.do
- http://www.michael-noll.com/tutorials/writing-an-hadoop-mapreduce-program-in-python/
- https://www.oreilly.com/learning/hadoop-with-python
- https://es.slideshare.net/DonaldMiner/mapreduce-design-patterns
- http://shop.oreilly.com/product/0636920025122.do
- http://shop.oreilly.com/product/0636920025122.do
- https://www.manning.com/books/hadoop-in-action
- https://blog.matthewrathbone.com/2013/04/17/what-is-hadoop.html
- http://research.google.com/archive/gfs.html

- http://en.wikipedia.org/wiki/Apache_Hadoop
- https://pythonhosted.org/mrjob/guides/writing-mrjobs.html
- http://zookeeper.apache.org/
- http://shop.oreilly.com/product/0636920028901.do
- https://www.usenix.org/legacy/event/usenix10/tech/full_papers/Hunt.pdf
- https://es.hortonworks.com/hadoop-tutorial/learning-the-ropes-of-the-hortonworks-sandbox/
- https://es.hortonworks.com/tutorials/
- Addison.Wesley.Learn.Python.The.Hard.Way.3rd.Edition
- Apress.Beginning.Python.From.Novice.To.Professional.2nd.Edition.Sep.2008.ISBN.1590599829
- Apress.Dive.Into.Python.Jul.2004.ISBN.1590593561
- Big Data Analytics with Spark.
- DeployBigInsights_4.2_SpectrumScale_HDFS_Transparency_with_Ambari_2.2_v1.0
- Hack_x_Crack_de0aPython
- OReilly.Data.Jan.2016.ISBN.1491948817
- OReilly.Introducing.Python.Dec.2014.ISBN.1449359361
- OReilly.Python.4th.Edition.Jan.2010.ISBN.0596158106
- OReilly.Python.Cookbook.3rd.Edition.Jun.2013.ISBN.1449340377
- Spark Cookbook By Rishi Yadav
- Wrox.Beginning.Python.Using.Python.2.6.And.Python.3.1.Feb.2010.ISBN.0470414634
- Zaharia M., et al. Learning Spark (O'Reilly, 2015)(274s)

Universidad de Alcalá
Escuela Politécnica Superior



ESCUELA POLITECNICA
SUPERIOR

Universidad
de Alcalá