UNIVERSIDAD NACIONAL DE INGENIERÍA

CTIC UNI
CENTRO DE TECNOLOGÍAS DE INFORMACIÓN Y COMUNICACIONES
UNIVERSIDAD NACIONAL DE INGENIERÍA

# CIENCIA DE DATOS II


## SESION 02

## HADOOP & MAP-REDUCE

*Las imágenes y graficos no me pertenecen y son usados solo con fines educativos

# ÍNDICE

# OBJETIVO

El curso de capacitación Big Data y Hadoop está diseñado para proporcionar conocimientos y habilidades basicas para convertirse en un exitoso desarrollador de Hadoop. En este curso se cubrirá un conocimiento profundo de conceptos tales como Hadoop Distributed File System, Hadoop Cluster: nodo único y múltiple, Map-Reduce, etc.

Después de completar el curso, los estudiantes podrán analizar y trabajar con datos voluminosos de cualquier organización desde varias perspectivas y podrán desarrollar informes y se podrán ver tendencias y tomar decisiones con respecto a las actividades comerciales que se ejecutan en las organizaciones.

# INTRODUCCIÓN


*¿Qué es Hadoop?*

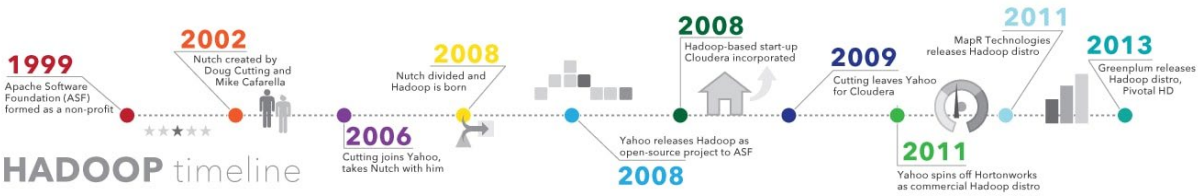
La biblioteca de software Apache Hadoop es un framework que permite el procesamiento distribuido de grandes conjuntos de datos en clusters de computadoras utilizando modelos de programación simples. Está diseñado para escalar desde servidores individuales a miles de máquinas, cada una de las cuales ofrece computación y almacenamiento local. En lugar de depender del hardware para brindar alta disponibilidad, la biblioteca en sí está diseñada para detectar y manejar fallas en la capa de la aplicación, por lo que brinda un servicio de alta disponibilidad sobre un grupo de computadoras, cada una de las cuales puede ser propensa a fallas.

Asi mismo proporciona un framework para el almacenamiento distribuido y el procesamiento de grandes datos utilizando el modelo de programación MapReduce. Hadoop se diseñó originalmente para clústeres de computadoras creados a partir de hardware básico, que sigue siendo el uso común. Desde entonces, también ha encontrado uso en clústeres de hardware de gama alta. Todos los módulos en Hadoop están diseñados con la suposición fundamental de que las fallas de hardware son ocurrencias comunes y deben ser manejadas automáticamente por el framework.

El núcleo de Apache Hadoop consta de una parte de almacenamiento, conocida como Sistema de archivos distribuidos de Hadoop (HDFS), y una parte de procesamiento que es un modelo de programación MapReduce.

Hadoop divide los archivos en grandes bloques y los distribuye entre los nodos de un clúster. Luego transfiere el código empaquetado a los nodos para procesar los datos en paralelo. Este enfoque aprovecha la localidad de los datos, donde los nodos manipulan los datos a los que tienen acceso. Esto permite que el conjunto de datos se procese de manera más rápida y eficiente de lo que sería en una arquitectura de supercomputadora más convencional que se basa en un sistema de archivos paralelo donde la computación y los datos se distribuyen a través de redes de alta velocidad.

*Historia*



# INTRODUCCIÓN

A medida que la World Wide Web creció a finales del siglo XX y principios del 2000, se crearon motores de búsqueda e índices para ayudar a localizar información relevante en medio del contenido basado en texto. En los primeros años, los resultados de búsqueda eran devueltos por humanos. Pero a medida que la web creció de docenas a millones de páginas, se necesitó automatización. Se crearon web crawlers, muchos como proyectos de investigación dirigidos por universidades, y despegaron las nuevas empresas de motores de búsqueda (Yahoo, AltaVista, etc.).

Uno de esos proyectos fue un motor de búsqueda web de código abierto llamado Nutch, una creación de Doug Cutting y Mike Cafarella. Querían obtener resultados de búsqueda web más rápido mediante la distribución de datos y cálculos en diferentes computadoras para poder realizar múltiples tareas simultáneamente.

Durante este tiempo, estaba en marcha otro proyecto de motor de búsqueda llamado Google. Se basaba en el mismo concepto: almacenar y procesar datos de forma distribuida y automatizada para que los resultados de búsqueda web relevantes pudieran obtenerse más rápido.

En 2006, Cutting se unió a Yahoo y se llevó consigo el proyecto Nutch, así como ideas basadas en los primeros trabajos de Google con la automatización del almacenamiento y procesamiento de datos distribuidos.

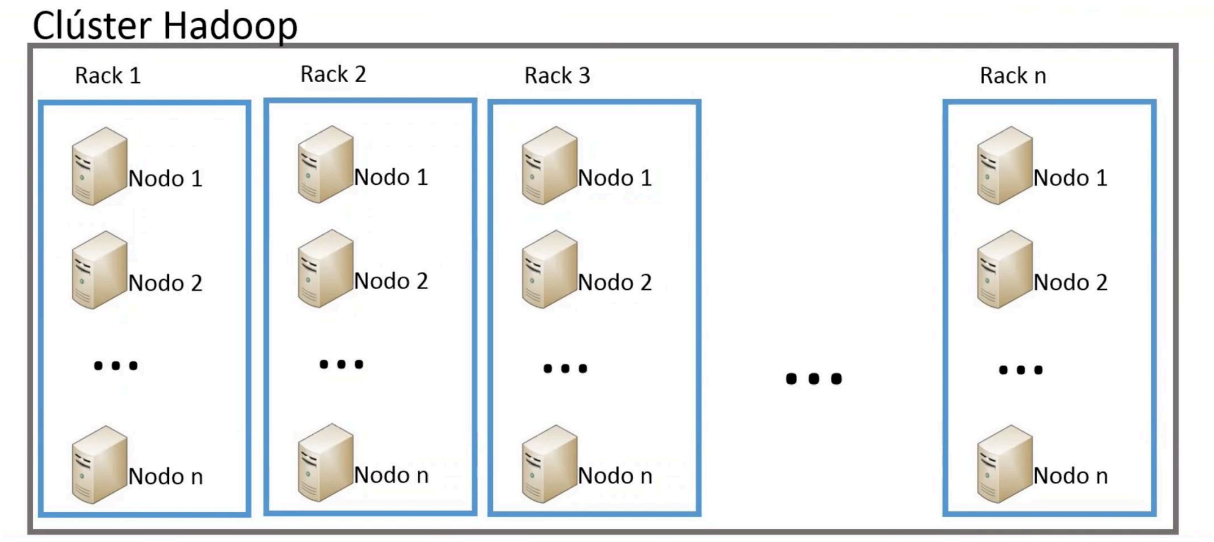
El proyecto Nutch se dividió: la parte del web crawler permaneció como Nutch y la parte de computación y procesamiento distribuidos se convirtió en Hadoop (llamado así por el elefante de juguete del hijo de Cutting). En 2008, Yahoo lanzó Hadoop como un proyecto de código abierto. Hoy en día, el framework y el ecosistema de tecnologías de Hadoop son administrados y mantenidos por Apache Software Foundation (ASF), una comunidad global de desarrolladores y colaboradores de software sin fines de lucro.

*¿Por qué es importante Hadoop?*

- Capacidad para almacenar y procesar grandes cantidades de cualquier tipo de datos rápidamente. Con volúmenes y variedades de datos en constante aumento, especialmente de las redes sociales y el Internet de las cosas (IoT), esa es una consideración clave.
- Poder computacional. El modelo de computación distribuida de Hadoop procesa big data rápidamente. Cuantos más nodos informáticos utilice, más potencia de procesamiento tendrá.

# INTRODUCCIÓN

- Tolerancia a fallos. El procesamiento de datos y aplicaciones está protegido contra fallas de hardware. Si un nodo deja de funcionar, los trabajos se redirigen automáticamente a otros nodos para asegurarse de que la computación distribuida no falle. Múltiples copias de todos los datos se almacenan automáticamente.
- Flexibilidad. A diferencia de las bases de datos relacionales tradicionales, no es necesario preprocesar los datos antes de almacenarlos. Puede almacenar tantos datos como desee y decidir cómo usarlos más adelante. Eso incluye datos no estructurados como texto, imágenes y videos.
- Bajo costo. El marco de código abierto es gratuito y utiliza hardware básico para almacenar grandes cantidades de datos.
- Escalabilidad. Puede hacer crecer fácilmente su sistema para manejar más datos simplemente agregando nodos. Se requiere poca administración.




*¿Cuáles son los desafíos de usar Hadoop?*

La programación de MapReduce no es una buena combinación para todos los problemas. Es bueno para solicitudes de información simples y problemas que se pueden dividir en unidades independientes, pero no es eficiente para tareas analíticas iterativas e interactivas. MapReduce utiliza muchos archivos. Debido a que los nodos no se comunican entre sí, excepto a través de ordenaciones y reorganizaciones, los algoritmos iterativos requieren múltiples fases de asignación aleatoria/ordenación y reducción para completarse. Esto crea múltiples archivos entre las fases de MapReduce y es ineficiente para la computación analítica avanzada.

# INTRODUCCIÓN

Hay una brecha de talento ampliamente reconocida. Puede ser difícil encontrar programadores de nivel de entrada que tengan suficientes conocimientos de Java para ser productivos con MapReduce. Esa es una de las razones por las que los proveedores de distribución están compitiendo para poner la tecnología relacional (SQL) por encima de Hadoop. Es mucho más fácil encontrar programadores con conocimientos de SQL que con conocimientos de MapReduce. Y, la administración de Hadoop parece en parte arte y en parte ciencia, lo que requiere un conocimiento de bajo nivel de los sistemas operativos, el hardware y la configuración del kernel de Hadoop.

Seguridad de datos. Otro desafío se centra en los problemas de seguridad de datos fragmentados, aunque están surgiendo nuevas herramientas y tecnologías. El protocolo de autenticación Kerberos es un gran paso para hacer que los entornos de Hadoop sean seguros.

Gestión y gobierno de datos completos. Hadoop no tiene herramientas completas y fáciles de usar para la gestión de datos, la limpieza de datos, el gobierno y los metadatos. Se carece especialmente de herramientas para la calidad y la estandarización de los datos.

*¿Cómo se utiliza Hadoop?*

Al ir más allá de su objetivo original de buscar en millones (o miles de millones) de páginas web y obtener resultados relevantes, muchas organizaciones buscan a Hadoop como su próxima plataforma de big data. Los usos populares hoy en día incluyen:

Almacenamiento y archivo de datos de bajo costo:

El costo modesto del hardware básico hace que Hadoop sea útil para almacenar y combinar datos tales como transaccionales, redes sociales, sensores, máquinas, datos científicos, secuencias de clics, etc. El almacenamiento de bajo costo le permite conservar información que actualmente no se considera crítica pero que podría querer analizar más adelante.

Sandbox para descubrimiento y análisis:

Debido a que Hadoop fue diseñado para manejar volúmenes de datos en una variedad de formas, puede ejecutar algoritmos analíticos. El análisis de big data en Hadoop puede ayudar a su organización a operar de manera más eficiente, descubrir nuevas oportunidades y obtener una ventaja competitiva del siguiente nivel. El enfoque de sandbox brinda la oportunidad de innovar con una inversión mínima.

Data Lakes:

Data Lakes admiten el almacenamiento de datos en su formato original o exacto. El objetivo es ofrecer una vista sin procesar o sin refinar de los datos a los científicos y analistas de datos para el descubrimiento y el análisis. Les ayuda a hacer preguntas nuevas o difíciles sin restricciones. Los lagos de datos no reemplazan a los almacenes de datos. De hecho, cómo proteger y gobernar

# INTRODUCCIÓN

los lagos de datos es un tema muy importante para TI. Pueden basarse en técnicas de federación de datos para crear estructuras de datos lógicas.

Complemente su almacén de datos (datawarehouse):

Ahora vemos que Hadoop comienza a ubicarse junto a los entornos de almacenamiento de datos, así como ciertos conjuntos de datos que se descargan del almacenamiento de datos a Hadoop o nuevos tipos de datos que van directamente a Hadoop. El objetivo final de cada organización es tener una plataforma adecuada para almacenar y procesar datos de diferentes esquemas, formatos, etc. para admitir diferentes casos de uso que se pueden integrar en diferentes niveles.

IoT y Hadoop:

Las cosas en el IoT necesitan saber qué comunicar y cuándo actuar. En el núcleo de IoT se encuentra una transmisión, siempre en forma de torrente de datos. Hadoop se usa a menudo como almacén de datos para millones o miles de millones de transacciones. Las capacidades masivas de almacenamiento y procesamiento también le permiten usar Hadoop como un espacio aislado para el descubrimiento y la definición de patrones que se monitorearán para la instrucción prescriptiva. Luego puede mejorar continuamente estas instrucciones, porque Hadoop se actualiza constantemente con nuevos datos que no coinciden con los patrones definidos anteriormente.

*¿Como está compuesto Hadoop?*

- HDFS (Hadoop Distributed File System):

  NameNode: Como se explicó, gestiona la metadata y la estructura del sistema de archivos.
  DataNodes: Almacenan los bloques de datos y ejecutan las operaciones de lectura/escritura.

- MapReduce:

  Un modelo de programación y un motor de procesamiento de datos que permite a los desarrolladores escribir programas que procesan grandes cantidades de datos en paralelo.

  Job Tracker: (En Hadoop 1) Se ejecuta en el nodo maestro y gestiona la asignación de trabajos y tareas.

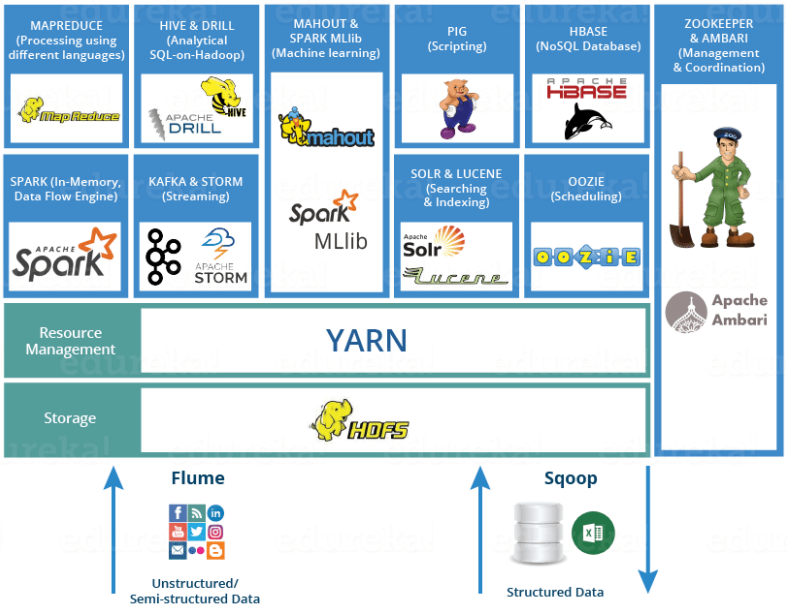
  Task Trackers: (En Hadoop 1) Se ejecutan en los nodos esclavos y ejecutan las tareas asignadas por el Job Tracker.

- YARN (Yet Another Resource Negotiator):

Introducido en Hadoop 2, es el gestor de recursos que reemplaza al Job Tracker y Task Trackers de Hadoop 1.
ResourceManager: Gestiona los recursos del clúster y las aplicaciones de ejecución.
NodeManager: Se ejecuta en cada nodo esclavo y gestiona los recursos y la ejecución de tareas en ese nodo.

- Otros Componentes del Ecosistema Hadoop:

Hive: Un sistema de data warehousing que proporciona una interfaz SQL para Hadoop.
Pig: Una plataforma para analizar grandes conjuntos de datos utilizando un lenguaje de scripts de alto nivel llamado Pig Latin.
HBase: Una base de datos NoSQL distribuida que se ejecuta sobre HDFS.
Spark: Un motor de procesamiento de datos en memoria que puede ser ejecutado en un clúster Hadoop.



# HDFS

HDFS (Hadoop Distributed File System) es un sistema de archivos distribuido diseñado para almacenar grandes volúmenes de datos en una red de computadoras. Vamos a explicarlo de manera intuitiva utilizando una analogía sencilla.

**Analogía Intuitiva:**

Imaginemos una biblioteca enorme que contiene millones de libros. Sin embargo, esta biblioteca es tan grande que no cabe en un solo edificio. Entonces, se distribuye en varios edificios diferentes, pero todos estos edificios funcionan juntos como una sola biblioteca. Aquí está cómo se compara esto con HDFS:

- Biblioteca (HDFS):

  La biblioteca completa, con todos sus edificios, representa HDFS. Es un sistema de archivos que distribuye y gestiona datos en múltiples computadoras.

- Edificios de la Biblioteca (DataNodes):

  Cada edificio en la biblioteca es como un DataNode en HDFS. Los DataNodes son las computadoras que realmente almacenan los datos.
  Cada edificio almacena una parte de los libros (datos).

- Librería Principal (NameNode):

  Hay una librería principal central que sabe exactamente en qué edificio se encuentra cada libro. Esta es como el NameNode en HDFS.

  El NameNode no almacena los datos reales, sino la información sobre dónde se encuentra cada pedazo de datos en los DataNodes.

- Catálogo de la Biblioteca (Metadata):

  El NameNode actúa como el catálogo central de la biblioteca. Mantiene un registro de toda la metadata: qué libros existen, dónde están almacenados, cuántas copias hay, etc.

- Acceso a los Libros (Clientes y Operaciones de Lectura/Escritura):

  Cuando alguien quiere leer un libro, primero pregunta a la librería principal (NameNode) para saber en qué edificio está.

  Luego, va directamente a ese edificio (DataNode) para obtener el libro.

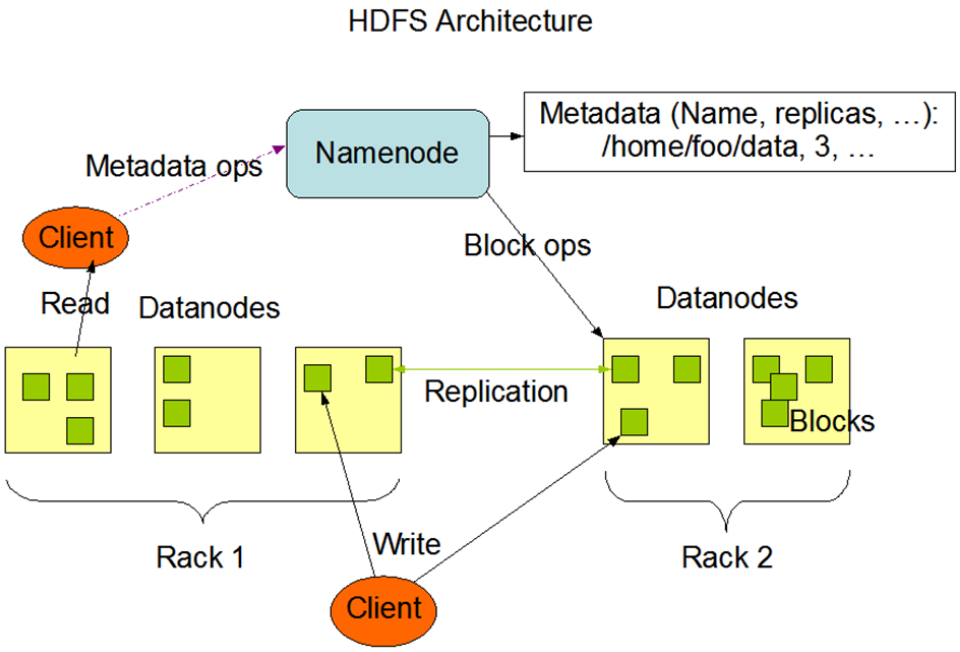
## HDFS

Similarmente, cuando se quiere guardar un nuevo libro, se consulta al NameNode para saber en qué edificios se puede guardar el libro.

- Copia de Seguridad (Replicación):

Para asegurarse de que los libros no se pierdan si un edificio se daña, la biblioteca hace copias de cada libro y las guarda en varios edificios. Esto es como la replicación en HDFS.

Cada pedazo de datos en HDFS se replica en varios DataNodes para asegurar la disponibilidad y la fiabilidad.




El diagrama ilustra cómo HDFS gestiona la distribución de datos y la comunicación entre los componentes principales (NameNode, DataNodes y Client). La arquitectura de HDFS está diseñada para ser escalable, redundante y tolerante a fallos, lo que la hace ideal para manejar grandes volúmenes de datos en entornos distribuidos.

**NameNode:**

Es el nodo maestro que gestiona la metadata del sistema de archivos.
La metadata incluye información sobre los nombres de los archivos, las réplicas de los bloques, y otras propiedades del archivo.
Los clientes interactúan con el NameNode para realizar operaciones de metadata, como abrir, cerrar y renombrar archivos o directorios.

# HDFS

**DataNodes:**

Son los nodos esclavos responsables de almacenar los bloques de datos.

Los DataNodes manejan las operaciones de lectura y escritura de los bloques de datos.
Reportan regularmente al NameNode con la lista de bloques que almacenan.

**Cliente:**

El cliente interactúa con el sistema de archivos HDFS para leer y escribir datos.
Para operaciones de lectura y escritura, el cliente primero se comunica con el NameNode para obtener la información de metadata necesaria.

**Flujo de Operaciones**

**Operaciones de Metadata:**

Cuando un cliente necesita realizar una operación de archivo, primero se comunica con el NameNode para obtener la metadata.

Ejemplo: Para leer un archivo, el cliente solicita al NameNode las ubicaciones de los bloques del archivo.

**Operaciones de Bloques:**

Una vez que el cliente obtiene la metadata del NameNode, puede comunicarse directamente con los DataNodes para leer o escribir bloques de datos.

- Lectura (Read):

  El cliente solicita al NameNode la metadata del archivo que desea leer.
  El NameNode proporciona las ubicaciones de los bloques del archivo.
  El cliente se comunica directamente con los DataNodes correspondientes para leer los bloques de datos.

- Escritura (Write):

  El cliente solicita al NameNode para crear un archivo y obtener las ubicaciones de los DataNodes donde se escribirán los bloques.

  El cliente escribe los datos directamente en los DataNodes.

## HDFS

Durante el proceso de escritura, los bloques de datos se replican automáticamente en múltiples DataNodes para garantizar la redundancia y la alta disponibilidad.

Las réplicas de los bloques se distribuyen entre los racks para mejorar la tolerancia a fallos (Rack Awareness).

**Replicación:**

HDFS replica los bloques de datos en varios DataNodes para garantizar la redundancia y la disponibilidad.

El nivel de replicación por defecto suele ser 3, pero puede ser configurado.

Los bloques se replican en diferentes racks para asegurar la disponibilidad en caso de fallo de un rack completo.

**Ejemplo de Flujo de Datos**

- Lectura:

  El cliente solicita al NameNode la metadata del archivo /home/foo/data.
  El NameNode responde con las ubicaciones de los bloques en los DataNodes.
  El cliente se comunica directamente con los DataNodes para leer los bloques de datos.

- Escritura:

  El cliente solicita al NameNode para crear un archivo nuevo.
  El NameNode proporciona las ubicaciones de los DataNodes para los nuevos bloques.
  El cliente escribe los datos en el primer DataNode.
  El primer DataNode replica los bloques a otros DataNodes en diferentes racks para asegurar la redundancia.

## HDFS

En el diagrama de HDFS analizado previamente, Hadoop encaja de la siguiente manera:

**Almacenamiento (HDFS):**

HDFS es la capa de almacenamiento distribuido de Hadoop. Todos los datos se almacenan en HDFS, distribuidos entre los DataNodes, y gestionados por el NameNode.

**Procesamiento (MapReduce/YARN):**

MapReduce y YARN son los componentes de procesamiento de Hadoop que trabajan con los datos almacenados en HDFS.

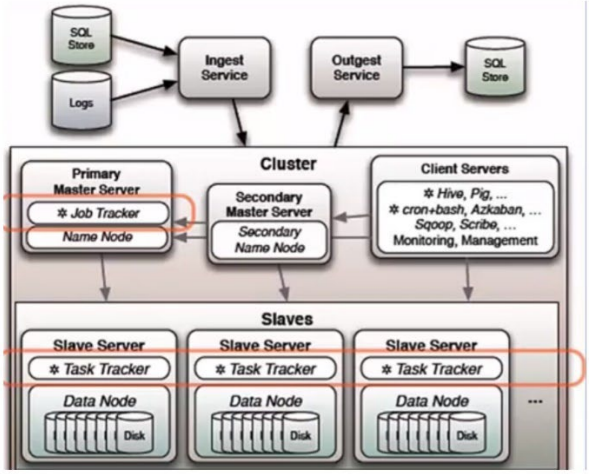
- MapReduce: Procesa los datos en paralelo mediante trabajos de MapReduce que se dividen en tareas.

  Job Tracker y Task Trackers en Hadoop 1 se encargan de la gestión y ejecución de trabajos de MapReduce.

- YARN: En Hadoop 2 y posteriores, YARN gestiona los recursos del clúster y las aplicaciones de procesamiento de datos.

  ResourceManager y NodeManagers se encargan de la asignación de recursos y la gestión de tareas.

# HADOOP




Por encima de los sistemas de archivos se encuentra el motor MapReduce, que consta de un JobTracker, al que las aplicaciones cliente envían trabajos de MapReduce. El JobTracker empuja el trabajo a los nodos TaskTracker disponibles en el clúster, esforzándose por mantener el trabajo lo más cerca posible de los datos.

Con un sistema de archivos compatible con racks, JobTracker sabe qué nodo contiene los datos y qué otras máquinas están cerca. Si el trabajo no se puede alojar en el nodo real donde residen los datos, se da prioridad a los nodos en el mismo rack. Esto reduce el tráfico de red en la red troncal principal.

Si un TaskTracker falla o se agota, esa parte del trabajo se reprograma. El TaskTracker en cada nodo genera un proceso de máquina virtual Java independiente para evitar que el propio TaskTracker falle si el trabajo en ejecución bloquea la JVM. Se envía un latido (heartbeat) desde TaskTracker a JobTracker cada pocos minutos para verificar su estado. Jetty (Java web server) expone el estado y la información de Job Tracker y TaskTracker y se puede ver desde un navegador web.

- Job Tracker: Este servicio se encarga de recibir los trabajos (jobs) desde los clientes, dividirlos en tareas más pequeñas y distribuir estas tareas a los nodos esclavos (slave nodes). Es responsable de monitorear la ejecución de estas tareas y gestionar las fallas.

- Task Trackers: Estos son servicios que se ejecutan en los nodos esclavos. Reciben las tareas asignadas por el Job Tracker y se encargan de ejecutarlas. Informan al Job Tracker sobre el progreso y el estado de las tareas.

# HADOOP

**Componentes del Sistema:**

- Primary Master Server: Este servidor actúa como el nodo principal que ejecuta el Job Tracker y el Name Node. El Job Tracker gestiona la distribución y seguimiento de los trabajos, mientras que el Name Node gestiona el sistema de archivos distribuido (por ejemplo, HDFS en Hadoop), manteniendo la información sobre dónde están almacenados los bloques de datos.

- Secondary Master Server: Este servidor actúa como un respaldo del Primary Master Server. Tiene su propio Name Node secundario, que es una copia del Name Node principal para tolerancia a fallos.

- Client Servers: Estos servidores ejecutan aplicaciones y herramientas que envían trabajos al sistema. Incluyen herramientas como Hive, Pig, cron jobs, Oozie, Flume, Sqoop, Scribe, y otras para la ingestión, procesamiento y monitoreo de datos.

- Nodos Esclavos (Slaves):

  Cada nodo esclavo tiene:

  Task Tracker: Gestiona la ejecución de las tareas asignadas a ese nodo.

  Data Node: Maneja el almacenamiento de los datos en el sistema distribuido, siguiendo las instrucciones del Name Node.

**Flujo de Trabajo:**

Ingest Service: Servicio que recibe y almacena datos entrantes en el sistema de almacenamiento distribuido.

Outgest Service: Servicio que maneja la salida de datos procesados hacia sistemas externos, como bases de datos SQL.

*¿Entonces qué es un JobTracker en Hadoop?*

JobTracker es el servicio daemon para enviar y rastrear trabajos de MapReduce en Hadoop. Job Tracker se ejecuta en su propio proceso JVM. En un clúster de producción típico, se ejecuta en una máquina separada. Cada nodo esclavo está configurado con la ubicación del nodo de

seguimiento de trabajos. JobTracker es un punto único de falla para el servicio Hadoop MapReduce. Si se cae, todos los trabajos en ejecución se detienen.

*¿Qué es una JVM?*

Tradicionalmente, el compilador de un lenguaje de alto nivel se encargaba de traducir ese lenguaje "sencillo" en lenguaje máquina, directamente utilizable por el computador a través del sistema operativo. Es decir, cuando compilamos un programa en C++ lo que obtenemos es un programa ejecutable, por ejemplo, para Windows, que este sistema operativo es capaz de ejecutar directamente contra el procesador, en un lenguaje "entendible" por este.

Sin embargo, muchos lenguajes modernos como Java o C# (y otros lenguajes de la plataforma .NET), lo que hacen es utilizar un paso intermedio entre estos dos estados: entre el código de alto nivel en el que escribimos las aplicaciones y el de bajo nivel que sale del proceso de compilación.

Cuando compilas una aplicación escrita en lenguaje Java, en realidad éste no se compila a lenguaje máquina, directamente entendible por el sistema operativo, sino a un lenguaje intermedio denominado Byte Code. Lo mismo ocurre con las aplicaciones .NET que se compilan también a un lenguaje intermedio llamado MSIL (MicroSoft Intermediate Language).

Entre el Byte Code (o el MSIL en el caso de .NET) y el sistema operativo se coloca un componente especial llamado Máquina virtual que es el que realmente va a ejecutar el código. Esta idea, por cierto, no tiene nada que ver con las máquinas virtuales a las que estamos acostumbrados hoy en día que ejecutan sistemas operativos completos, sino que es un concepto mucho más antiguo.

En el caso de Java, La **Java Virtual Machine o JVM** toma el código *Byte Code* resultante de compilar tu aplicación Java y lo compila a su vez a código nativo de la plataforma en la que se está ejecutando. La ventaja principal de este esquema es que es muy fácil crear un programa en Java y que luego éste se pueda ejecutar en cualquier sistema operativo para el cual exista una implementación de la JVM (hoy en día, casi literalmente todos).

*¿Cuántas instancias de JobTracker se ejecutan en un Hadoop Cluster?*

Solo se ejecuta un proceso de seguimiento de trabajos en cualquier clúster de Hadoop.

*¿Qué acciones realiza JobTracker en hadoop?*

Las aplicaciones cliente envían trabajos al rastreador de trabajos. JobTracker habla con NameNode para determinar la ubicación de los datos. JobTracker ubica los nodos de TaskTracker con slots disponibles en los datos o cerca de ellos. JobTracker envía el trabajo a los nodos de TaskTracker elegidos.

# HADOOP

Los nodos TaskTracker son monitoreados. Si no envían señales de latido con la frecuencia suficiente, se considera que han fallado y el trabajo se programa en un TaskTracker diferente. Un TaskTracker notificará al JobTracker cuando una tarea falle. El JobTracker decide qué hacer entonces: puede volver a enviar el trabajo a otro lugar, puede marcar ese registro específico como algo que debe evitarse e incluso puede incluir en la lista negra al TaskTracker como poco confiable. Cuando se completa el trabajo, JobTracker actualiza su estado. Las aplicaciones cliente pueden sondear JobTracker para obtener información.

*¿Cómo programa JobTracker una tarea?*

Los TaskTrackers envían mensajes de latido al JobTracker, generalmente cada pocos minutos, para asegurarle a JobTracker que aún está activo. Estos mensajes también informan a JobTracker sobre la cantidad de slots disponibles, por lo que JobTracker puede mantenerse actualizado con respecto a dónde se puede delegar el trabajo en el clúster. Cuando JobTracker intenta encontrar un lugar para programar una tarea dentro de las operaciones de MapReduce, primero busca un slot vacío en el mismo servidor que aloja el DataNode que contiene los datos y, si no, busca un slot vacío en una máquina en el mismo rack.

El tipo de slot indica qué tipo de tareas (map o reduce) pueden servir. En un momento dado, solo se puede ejecutar una tarea por slot. Si bien la configuración de los slots es fundamental para el rendimiento, Hadoop utiliza de manera predeterminada números fijos de map slots y reduce slots en cada nodo durante la vida útil de un clúster.

*¿Qué es un rastreador de tareas en Hadoop?*

Un TaskTracker es un demonio de nodo esclavo en el clúster que acepta tareas (operaciones Map, Reduce y Shuffle) de un JobTracker. Task Tracker se ejecuta en su propio proceso JVM. Cada TaskTracker está configurado con un conjunto de ranuras, estas indican la cantidad de tareas que puede aceptar. TaskTracker inicia un proceso JVM separado para hacer el trabajo real (llamado Instancia de tarea), esto es para garantizar que la falla del proceso no elimine el rastreador de tareas. TaskTracker monitorea estas instancias de tareas, capturando los códigos de salida. Cuando las instancias de Tarea finalizan, con éxito o no, el rastreador de tareas notifica al JobTracker. Los TaskTrackers también envían mensajes de latido al JobTracker, generalmente cada pocos minutos, para asegurarle a JobTracker que aún está activo. Estos mensajes también informan a JobTracker sobre la cantidad de slots disponibles, por lo que JobTracker puede mantenerse actualizado con respecto a dónde se puede delegar el trabajo en el clúster.

*¿Cuántas instancias de TaskTracker se ejecutan en un Hadoop Cluster?*

Solo se ejecuta un proceso de seguimiento de tareas en cualquier nodo esclavo de Hadoop.

# HADOOP

*¿Qué es una instancia de tarea en Hadoop?*

Las instancias de tareas son los trabajos reales de MapReduce que se ejecutan en cada nodo esclavo. TaskTracker inicia un proceso JVM separado para hacer el trabajo real (llamado Instancia de tarea), esto es para garantizar que la falla del proceso no elimine el rastreador de tareas. Puede haber múltiples procesos de instancia de tarea ejecutándose en un nodo esclavo. Esto se basa en la cantidad de slots configuradas en el rastreador de tareas. De forma predeterminada, se genera un nuevo proceso JVM de instancia de tarea para una tarea.

*¿Dónde se ejecuta una instancia de tarea?*

Cada instancia de tarea se ejecuta en su propio proceso JVM.

*¿Cuántos procesos Daemon se ejecutan en un sistema Hadoop?*

Hadoop se compone de cinco demonios separados. Cada uno de estos demonios se ejecuta en su propia JVM.

Los siguientes 3 demonios se ejecutan en nodos maestros:

- NameNode: este demonio almacena y mantiene los metadatos para HDFS.
- NameNode secundario: realiza funciones de limpieza para NameNode.
- JobTracker: administra trabajos de MapReduce, distribuye tareas individuales a máquinas que ejecutan Task Tracker.

Los siguientes 2 demonios se ejecutan en cada nodo esclavo.

- Nodo de datos: almacena bloques de datos HDFS reales.
- TaskTracker: responsable de instanciar y monitorear tareas individuales de Map y Reduce.

*¿Cuál es la configuración de un nodo esclavo típico en un clúster de Hadoop? ¿Cuántas JVM se ejecutan en un nodo esclavo?*

Se ejecuta una única instancia de un rastreador de tareas en cada nodo esclavo. El rastreador de tareas se ejecuta como un proceso JVM separado.

# HADOOP

Se ejecuta una única instancia de un daemon DataNode en cada nodo esclavo. El demonio DataNode se ejecuta como un proceso JVM separado.

Una o varias instancias de instancia de tarea se ejecutan en cada nodo esclavo.

Cada instancia de tarea se ejecuta como un proceso JVM independiente.

La cantidad de instancias de tareas se puede controlar mediante la configuración.

Por lo general, una máquina de gama alta está configurada para ejecutar más instancias de tareas.

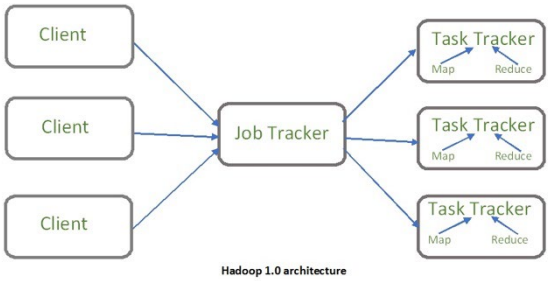
# LOAD BALANCING

Hadoop divide la entrada de un trabajo de MapReduce en partes de tamaño fijo denominadas divisiones de entrada, o simplemente divisiones. Hadoop crea una tarea de mapa para cada división, que ejecuta la función de mapa definida por el usuario para cada registro en la división.

Tener muchas divisiones significa que el tiempo necesario para procesar cada división es pequeño en comparación con el tiempo para procesar toda la entrada. Por lo tanto, si estamos procesando las divisiones en paralelo, el procesamiento tiene mejor balanceo de carga si las divisiones son pequeñas, ya que una máquina más rápida podrá procesar proporcionalmente más divisiones en el transcurso del trabajo que una máquina más lenta. Incluso si las máquinas son idénticas, los procesos fallidos u otros trabajos que se ejecutan simultáneamente hacen deseable el balanceo de carga, y la calidad del balanceo de carga aumenta a medida que las divisiones se vuelven más finas. Por otro lado, si las divisiones son demasiado pequeñas, la sobrecarga de administrar las divisiones y la creación de tareas de mapas comienza a dominar el tiempo total de ejecución del trabajo. Para la mayoría de los trabajos, un buen tamaño de división tiende a ser el tamaño de un bloque HDFS, 64 MB de forma predeterminada, aunque esto se puede cambiar para el clúster (para todos los archivos recién creados) o se puede especificar cuando se crea cada archivo.

# YARN (Hadoop 2.0)

YARN significa "*Yet Another Resource Negotiator*". Se introdujo en Hadoop 2.0 para eliminar el cuello de botella en Job Tracker que estaba presente en Hadoop 1.0. YARN se describió como un "Administrador de recursos rediseñado" en el momento de su lanzamiento, pero ahora ha evolucionado para ser conocido como un sistema operativo distribuido a gran escala utilizado para el procesamiento de Big Data.



Hadoop 1.0 architecture

Vamos a desglosar la arquitectura de YARN (Yet Another Resource Negotiator) y cómo difiere de la versión anterior de Hadoop (Hadoop 1.0) en términos de gestión de recursos y procesamiento.

**Arquitectura de Hadoop 1.0**

En Hadoop 1.0, la gestión de recursos y el procesamiento de datos estaban muy integrados en un componente principal llamado JobTracker. Aquí repasemos cómo funcionaba:

**Componentes Clave en Hadoop 1.0**

**JobTracker:**

- Responsabilidad:

  Gestionaba tanto los recursos del clúster como la ejecución de los trabajos de MapReduce.

- Funcionalidades:

  Recibía trabajos de los clientes.
  Dividía los trabajos en tareas.
  Asignaba tareas a los nodos disponibles (TaskTrackers).
  Monitoreaba la ejecución de las tareas.
  Gestionaba fallos de tareas y reasignaba tareas fallidas.

# YARN (Hadoop 2.0)

**TaskTracker:**

- Responsabilidad:

  Ejecutaba las tareas asignadas por el JobTracker en cada nodo del clúster.

- Funcionalidades:

  Informaba al JobTracker sobre el estado de las tareas.
  Ejecutaba las tareas de Map y Reduce.

**Limitaciones de Hadoop 1.0**

- Escalabilidad: El JobTracker se convertía en un cuello de botella a medida que el tamaño del clúster aumentaba debido a que gestionaba tanto la asignación de recursos como la ejecución de trabajos.
- Gestión de Recursos: No había separación clara entre la gestión de recursos y la gestión de trabajos, lo que complicaba la eficiencia y la flexibilidad en el uso de recursos.

**Arquitectura de YARN (Hadoop 2.0 y posteriores)**

YARN (Yet Another Resource Negotiator) fue introducido en Hadoop 2.0 para superar las limitaciones de Hadoop 1.0. La arquitectura de YARN separa claramente la gestión de recursos del procesamiento de datos. Aquí están los componentes clave y sus funciones:

**Componentes Clave en YARN**

**ResourceManager:**

- Responsabilidad:

  Gestiona los recursos del clúster de manera global.

- Funcionalidades:

  Conoce el estado y la disponibilidad de recursos en todo el clúster.
  Asigna recursos a las aplicaciones en ejecución basándose en políticas de planificación y uso eficiente.
  Maneja la cola de trabajos y la asignación de contenedores de recursos a las aplicaciones.

# YARN (Hadoop 2.0)

**NodeManager:**

- Responsabilidad:

  Gestiona los recursos y la ejecución de contenedores en un nodo específico.

- Funcionalidades:

  Monitorea el uso de recursos (CPU, memoria) en su nodo.
  Inicia y supervisa los contenedores solicitados por el ResourceManager.
  Informa periódicamente al ResourceManager sobre el estado y la disponibilidad de recursos.

**ApplicationMaster:**

- Responsabilidad:

  Gestiona la ejecución de una aplicación específica.

- Funcionalidades:

  Solicita recursos al ResourceManager.
  Coordina la ejecución de las tareas dentro de los contenedores asignados.
  Maneja la lógica específica de la aplicación para la reintentos, fallos y finalización.

**Contenedores:**

- Responsabilidad:

  Aíslan y gestionan los recursos asignados para ejecutar una tarea específica.

- Funcionalidades:

  Proporcionan un entorno controlado y limitado para la ejecución de procesos.
  Cada contenedor tiene recursos específicos como CPU, memoria y disco.

**Flujo de Trabajo en YARN**

1. Inicio de Aplicación:

   El cliente envía una solicitud de ejecución de aplicación al ResourceManager.
   El ResourceManager asigna un contenedor inicial para el ApplicationMaster de esa aplicación.

2. Gestión de Recursos:

   El ApplicationMaster solicita recursos adicionales (contenedores) al ResourceManager para ejecutar las tareas.

# YARN (Hadoop 2.0)

El ResourceManager asigna contenedores basándose en la disponibilidad de recursos y políticas de planificación.

3. Ejecución de Tareas:

   El ApplicationMaster coordina la ejecución de tareas dentro de los contenedores asignados.
   Los NodeManagers gestionan la ejecución de contenedores en sus nodos respectivos y reportan el estado al ResourceManager.

4. Monitoreo y Finalización:

   El ApplicationMaster monitorea la ejecución de todas las tareas.
   Al completar la ejecución, el ApplicationMaster informa al ResourceManager y finaliza la aplicación.

**Ventajas de YARN sobre Hadoop 1.0**

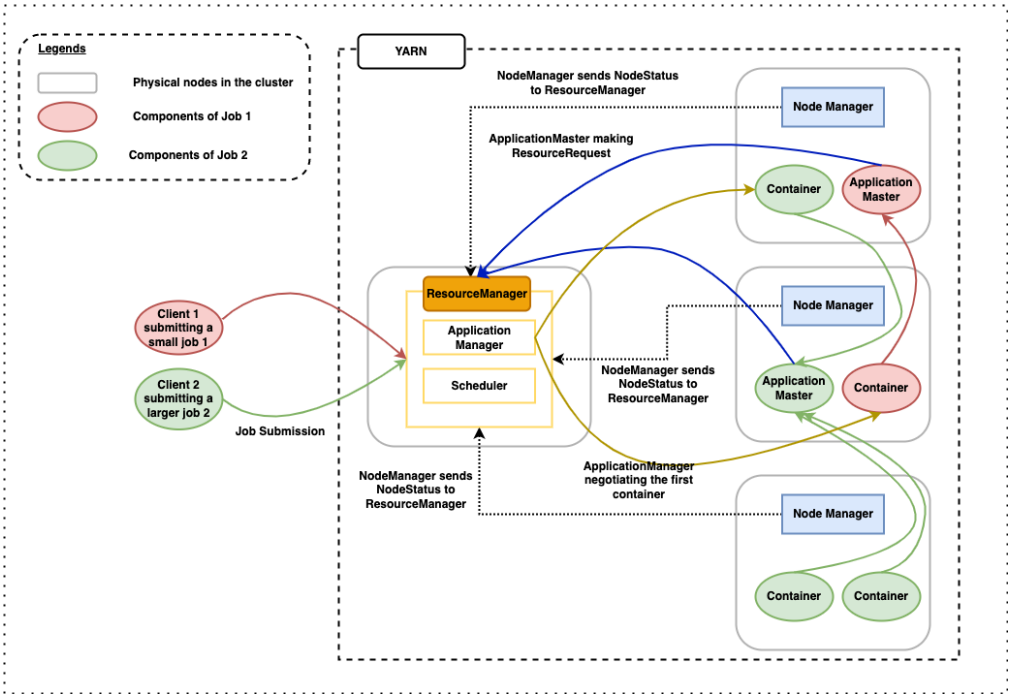
- Escalabilidad Mejorada: Al separar la gestión de recursos y la gestión de aplicaciones, YARN puede escalar mejor en grandes clústeres.

- Flexibilidad: YARN permite ejecutar diferentes tipos de aplicaciones más allá de MapReduce, como Spark, Tez y otras, proporcionando un entorno más flexible y general.

- Utilización Eficiente de Recursos: La separación permite una mejor utilización de los recursos del clúster, adaptándose dinámicamente a las necesidades de las aplicaciones.

# YARN (Hadoop 2.0)

https://aws.amazon.com/es/blogs/big-data/configure-hadoop-yarn-capacityscheduler-on-amazon-emr-on-amazon-ec2-for-multi-tenant-heterogeneous-workloads/




1. **ResourceManager:**

- Responsabilidad: Gestiona los recursos del clúster de manera global.
- Componentes Internos:
    - Application Manager: Recibe las solicitudes de aplicaciones y coordina la asignación inicial de recursos.
    - Scheduler: Asigna recursos a las aplicaciones basándose en políticas de planificación y disponibilidad.

2. **NodeManager:**

- Responsabilidad: Gestiona los recursos y la ejecución de contenedores en cada nodo específico.
- Funcionalidades:

  Inicia y supervisa los contenedores.

  Informa periódicamente al ResourceManager sobre el estado de los recursos en el nodo.

## YARN (Hadoop 2.0)

3. **ApplicationMaster:**

- Responsabilidad: Gestiona la ejecución de una aplicación específica.
- Funcionalidades:

  Solicita recursos al ResourceManager.
  Coordina la ejecución de las tareas dentro de los contenedores asignados.
  Maneja la lógica específica de la aplicación para reintentos, fallos y finalización.

4. **Contenedores:**

- Responsabilidad: Aíslan y gestionan los recursos asignados para ejecutar una tarea específica.
- Funcionalidades:

  Proporcionan un entorno controlado y limitado para la ejecución de procesos.

  Cada contenedor tiene recursos específicos como CPU, memoria y disco.

**Flujo de Trabajo Detallado**

1. Envío de Trabajo (Job Submission)

   Cliente 1 envía un trabajo pequeño (Job 1) al ResourceManager.
   Cliente 2 envía un trabajo más grande (Job 2) al ResourceManager.

2. Gestión de Recursos

   El ResourceManager recibe las solicitudes de trabajos y pasa la responsabilidad de gestionar cada trabajo al Application Manager.
   El Scheduler dentro del ResourceManager asigna recursos a las aplicaciones basándose en políticas de planificación.

3. Creación del ApplicationMaster

   Para cada trabajo (Job 1 y Job 2), el ResourceManager asigna el primer contenedor que ejecutará el ApplicationMaster para ese trabajo específico.

4. Ejecución de Tareas

   El ApplicationMaster solicita recursos adicionales al ResourceManager para ejecutar las tareas del trabajo.
   El ResourceManager asigna contenedores en varios nodos del clúster, gestionados por los NodeManagers.

# YARN (Hadoop 2.0)

5. Monitorización y Comunicación

   Los NodeManagers ejecutan los contenedores asignados y supervisan su ejecución.
   Los NodeManagers envían informes de estado periódicos al ResourceManager sobre la disponibilidad y el uso de recursos.
   El ApplicationMaster gestiona la ejecución de las tareas dentro de los contenedores y maneja los fallos, los reintentos y la finalización del trabajo.

**Interacción de Componentes en el Diagrama**

1. ResourceManager: Centraliza la gestión de recursos y coordina la asignación de contenedores a través del Scheduler.
2. NodeManagers: Informan al ResourceManager sobre el estado de los recursos y gestionan los contenedores en cada nodo.
3. ApplicationMasters: Gestionan la ejecución de trabajos individuales, solicitando recursos y coordinando la ejecución de tareas en los contenedores asignados.

Esta separación mejora la escalabilidad, flexibilidad y eficiencia del clúster, permitiendo que Hadoop soporte una variedad más amplia de modelos de procesamiento de datos más allá de MapReduce.

# MapReduce

MapReduce es un modelo de programación desarrollado por Google para el procesamiento y generación de grandes conjuntos de datos distribuidos en clústeres de computadoras. Es especialmente útil para tareas que pueden dividirse en sub-tareas independientes y ejecutarse en paralelo en diferentes nodos del clúster. Hadoop, un marco de trabajo de código abierto de Apache, implementa el modelo MapReduce, permitiendo el procesamiento distribuido de datos a gran escala.

**Conceptos Clave**

**Clave-Valor en MapReduce**

- Definición: El modelo MapReduce se basa en el procesamiento de datos en forma de pares clave-valor. Una clave es un identificador único que representa un fragmento de datos, y un valor es el dato asociado con esa clave.
- Función del Par Clave-Valor: Permite organizar y agrupar datos de manera que puedan procesarse de forma distribuida. Las claves se utilizan para agrupar los datos intermedios, facilitando el trabajo de los reducers.

Ejemplo de Par Clave-Valor

Clave: subject (por ejemplo, "Math")
Valor: grade (por ejemplo, 85)

En un proceso de contar palabras:

Clave: word (por ejemplo, "Hadoop")
Valor: count (por ejemplo, 1)

**Mapper**

- Definición: El mapper es la primera fase en el modelo MapReduce. Su función principal es procesar los datos de entrada y transformarlos en pares clave-valor intermedios.
- Funcionamiento: Cada nodo del clúster ejecuta una instancia del mapper en sus particiones de datos. El mapper lee los datos de entrada, los transforma y emite pares clave-valor intermedios.

Ejemplo: En un problema de contar palabras, el mapper leería líneas de texto, dividiría las líneas en palabras y emitiría cada palabra con un valor 1.

# MapReduce

**Reducer**

- Definición: El reducer es la segunda fase en el modelo MapReduce. Su función es procesar los pares clave-valor intermedios generados por los mappers y producir los resultados finales.
- Funcionamiento: Los pares clave-valor intermedios se barajan y clasifican para que todos los valores asociados con la misma clave lleguen al mismo reducer. El reducer procesa estos valores, realiza cálculos agregados y emite los resultados finales.

Ejemplo: En un problema de contar palabras, el reducer recibiría todas las ocurrencias de cada palabra, sumaría los valores y emitiría el total de ocurrencias para cada palabra.

**Flujo de Ejecución de un Trabajo MapReduce**

1. División de Datos: Hadoop divide el archivo de entrada en fragmentos y asigna cada fragmento a una tarea de mapeo.
2. Fase de Mapeo: Las tareas de mapeo procesan los fragmentos de datos y generan pares clave-valor intermedios.
3. Barajado y Clasificación: Hadoop agrupa y clasifica los pares clave-valor intermedios, preparando los datos para la fase de reducción.
4. Fase de Reducción: Las tareas de reducción procesan los pares clave-valor agrupados, realizando cálculos agregados y produciendo los resultados finales.
5. Escritura de Resultados: Los resultados generados por la fase de reducción se escriben en el sistema de archivos HDFS.