

Desarrollo con Hadoop

Temario

Francisco Philip [francisco.philip@gmail.com]



Temario

- Qué es hadoop?.
- Configurando un entorno de desarrollo Hadoop.
- Inserción de datos en Hadoop con Java.
- Aplicaciones MapReduce distribuidas e invertidas.
- Configurar ,crear y utilizar combinadores y particionadores.
- TotalOrderPartitioner.
- Clasificación de datos con clave compuesta
- Definición de clases para formatos de entrada y salida.
- Compresión de datos.
- RawComparator.
- Join Map-side.
- Filtro Bloom.
- Prueba unitaria de un job MapReduce.
- Importación de datos en HBase.
- Creación de un job HBase MapReduce.
- Creación de funciones de usuario Pig y Hive.
- Definir un flujo Oozie.

Qué es Hadoop?

Según Apache Hadoop Site

What Is Apache Hadoop?

The Apache™ Hadoop® project develops open-source software for reliable, scalable, distributed computing.

The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. Rather than rely on hardware to deliver high-availability, the library itself is designed to detect and handle failures at the application layer, so delivering a highly-available service on top of a cluster of computers, each of which may be prone to failures.

Quien usa?

<https://wiki.apache.org/hadoop/PoweredBy>

Historia

Como podríamos imaginarnos los primeros en encontrarse con problemas de procesamiento, almacenamiento y alta disponibilidad de grandes bancos de información fueron los buscadores y las redes sociales. Con la implementación de sus algoritmos de búsquedas y con la indexación de los datos en poco tiempo se dieron cuenta de que debían hacer algo y ya.

Fue así como nació el sistema de archivos de Google (GFS), un sistema de archivos distribuido capaz de ser montado en clusters de cientos o incluso de miles de máquinas compuestas por hardware de bajo coste. Entre algunas de sus premisas contaba con que es más óptimo el tratamiento de pocos ficheros de gran tamaño que el tratamiento de muchos pequeños y que la mayor parte de las modificaciones sobre los ficheros suelen consistir en añadir datos nuevos al final en vez de reemplazar los existentes.

De la mano del GFS, la gente de Google creo Map-Reduce como modelo de programación cuya finalidad es paralelizar o distribuir el procesamiento sobre los datos a través de los clústeres, para luego publicarlos dando base al proyecto Hadoop.

The Google File System, Octubre 2003. Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung - <http://research.google.com/archive/gfs.html>
MapReduce: Simplified Data Processing on Large Clusters, Diciembre 2004. Jeffrey Dean and Sanjay Ghemawat - <http://research.google.com/archive/mapreduce.html>

Así pues de la mano Doug Cutting un trabajador de Yahoo! y del gran avance de Google fue como nació Hadoop .

Cabe mencionar a Facebook ya que es parte de la popularización y contribución al desarrollo de Hadoop, pues bien fue una de los primeros en implementar uno de los clústeres más grandes y se atribuye el desarrollo de <Hive> el cual permite realizar consultas similares al SQL sobre los datos en un ambiente distribuido.

Ventajas

- El framework Hadoop permite al usuario escribir rápidamente y probar sistemas distribuidos. Es eficiente y automático que distribuye los datos y trabajar a través de las máquinas y a su vez, utiliza el paralelismo de los núcleos de CPU.
- Hadoop no depende de hardware para proporcionar tolerancia a fallos y alta disponibilidad (FTHA), y Hadoop propia biblioteca ha sido diseñado para detectar y controlar errores en el nivel de aplicación.
- Los servidores se pueden añadir o quitar del clúster dinámicamente y Hadoop continúa funcionando sin interrupción.
- Otra gran ventaja de Hadoop es que aparte de ser open source, que es compatible en todas las plataformas ya que está basado en Java.

Composición de Hadoop

Hadoop Common:

Librerías de uso común que dan soporte a todas los demás módulos que conforman *Hadoop*.

Hadoop Distributed File System (HDFS™):

Filesystem distribuido que provee de alto rendimiento al acceso de datos.

Hadoop YARN:

Un framework para la programación de tareas y la gestión de recursos del *cluster*.

Hadoop MapReduce:

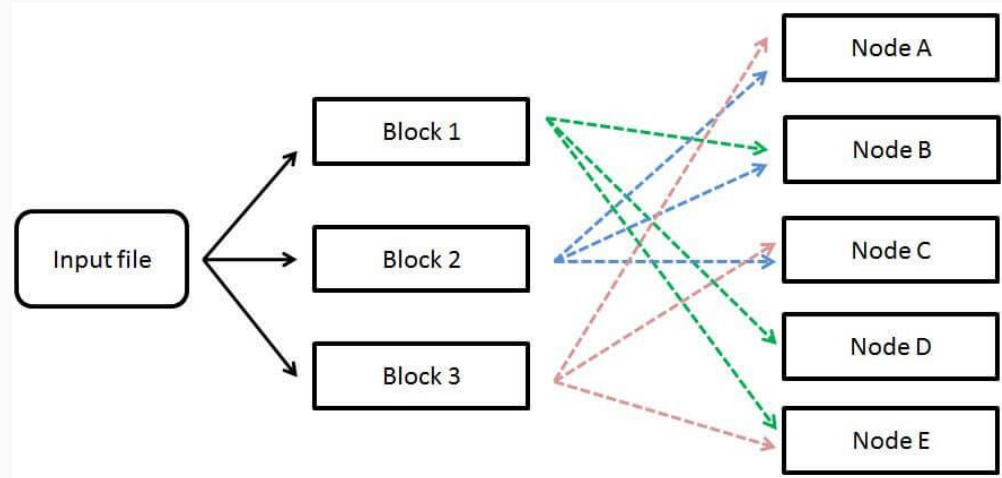
Sistema basado en YARN que nos permite de manera paralela el procesamiento de grandes conjuntos de datos.

Hadoop Distributed File System (HDFS™)

Es un sistema de archivos distribuidos el cual permite difundir los datos a través de cientos o miles de nodos para su procesamiento. Aquí es donde se proporciona redundancia (Los datos están repetidos o replicados en varios nodos) y tolerancia a fallos (Si falla algún nodo se reemplaza automáticamente).

En su funcionamiento el sistema de archivos HDFS divide los datos en bloques donde a su vez cada bloques se replica en distintos nodos de manera que la caída de un nodo no implique la pérdida de los datos que éste contiene.

De esta manera se facilita el uso de modelos de programación como MapReduce, ya que se puede acceder a varios bloques de un mismo fichero en forma paralela.



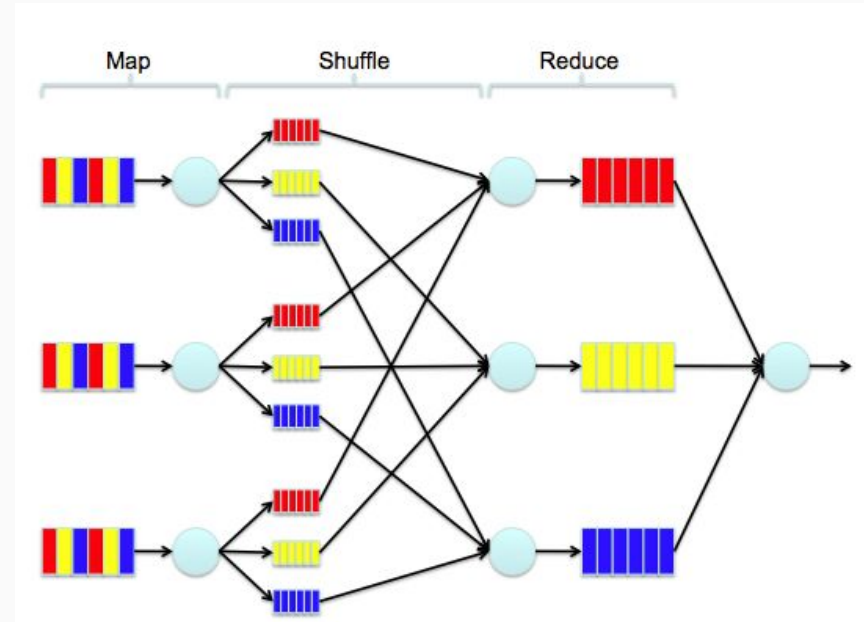
Hadoop MapReduce

Es el corazón de Hadoop el cual permite el fácil desarrollo de aplicaciones y algoritmos bajo el lenguaje Java para el procesamiento distribuido de grandes cantidades de datos. Dentro del ecosistema las aplicaciones desarrolladas para el framework MapReduce se conocen como Jobs, éstos se componen de las siguientes funciones:

Map (Mapeo): Encargada de la división de las unidades de procesamiento a ejecutar en cada nodo y de su distribución para su ejecución en paralelo. Aquí a cada llamada se le asignará una lista de pares key/value.

Shuffle and sort (Combinación y Orden): Aquí se mezclan los resultados de la etapa anterior con todas las parejas clave/valor para combinarlos en una lista y a su vez se ordenan por clave.

Reduce: Aquí se reciben todas las claves y listas de valores haciendo si es necesaria la agregación de las mismas.



Tipos de instalación

- Stand Alone (un solo nodo)
- Pseudo-distribuida
- Totalmente Distribuida

Standalone

La instalación Stand Alone nos permite configurar un despliegue de Hadoop donde todos los servicios tanto de maestro como de esclavo se ejecutan en un solo nodo al mismo tiempo que solo trabaja con un thread.

Una de las grandes ventajas es que nos puede servir para probar aplicaciones sin tener que preocuparnos de concurrencias de ejecución, además es una buena manera de entrar en el mundo de Hadoop pues experimentarás aspectos a tener en cuenta de instalación y configuración los cuales seguramente necesitaras en un futuro.

Pseudo-distribuida

Esta instalación al igual que la anterior también se monta sobre un solo nodo, todos sus servicios corren sobre éste pero la diferencia es que permite ejecutar múltiples threads.

Gracias a que ejecutamos aplicaciones multi-threads podemos experimentar la ejecución de aplicaciones sin necesidad de disponer de múltiples computadoras pues aquí empezamos a aprovechar mucho mejor los distintos núcleos de nuestro procesador.

Totalmente Distribuida

Tal como su nombre lo indica es aquí donde instalamos y configuramos un ambiente completamente distribuido donde dispondremos de un master y varios slaves aprovechando completamente la paralelización.

Este es el tipo de instalación para un ambiente de producción donde consideraremos grandes racks y nodos bien dotados de procesador y memoria para divisar los verdaderos beneficios de Hadoop.

Ecosistema de Hadoop

Debido a la creciente comunidad Open Source existen distintos proyectos y herramientas que ofrecen funcionalidades adicionales las cuales son consideradas parte de un gran ecosistema pensado en apoyar las distintas etapas de un proyecto Big Data.

A continuación enumeramos parte de estas.

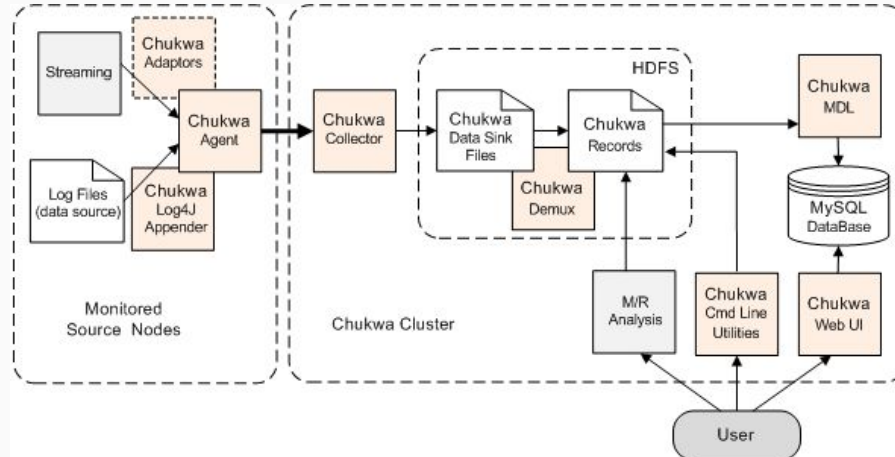
Captura y Manipulación De Datos

Chukwa

<http://chukwa.apache.org/>



Es un proyecto construido para capturar y analizar grandes volúmenes de datos principalmente logs. Debido a que está construido sobre Hadoop hereda escalabilidad y robustez con el uso de HDFS y Map-Reduce al mismo tiempo que provee adicionalmente un grupo de herramientas flexibles y potentes para visualizar, controlar y analizar los datos.



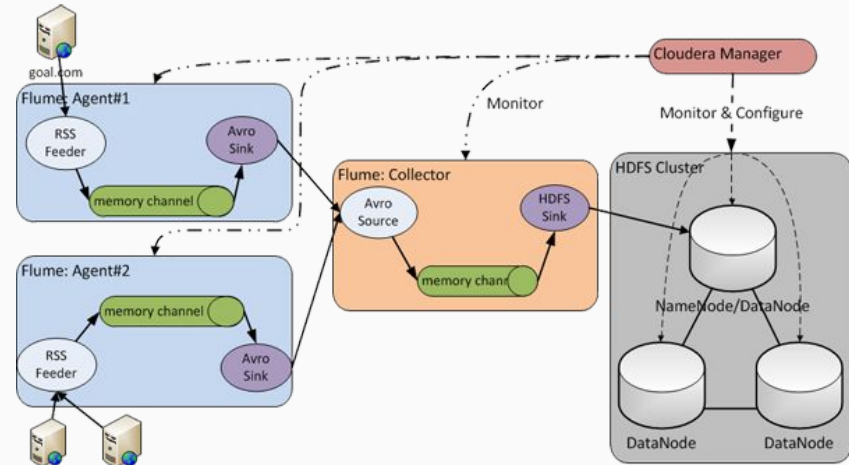
Captura y Manipulación De Datos

Flume

<https://flume.apache.org/>



Es una herramienta distribuida para la recolección, agregación y transmisión de grandes volúmenes de datos de diferentes orígenes altamente configurable. Ofrece una arquitectura basada en la transmisión de datos por streaming altamente flexible y configurable pero a la vez simple de manera que se adapta a distintas situaciones tales como monitorización logs (control de calidad y mejora de la producción), obtención de datos desde las redes sociales (Sentiment Analysis y medición de reputación) o mensajes de correo electrónico.



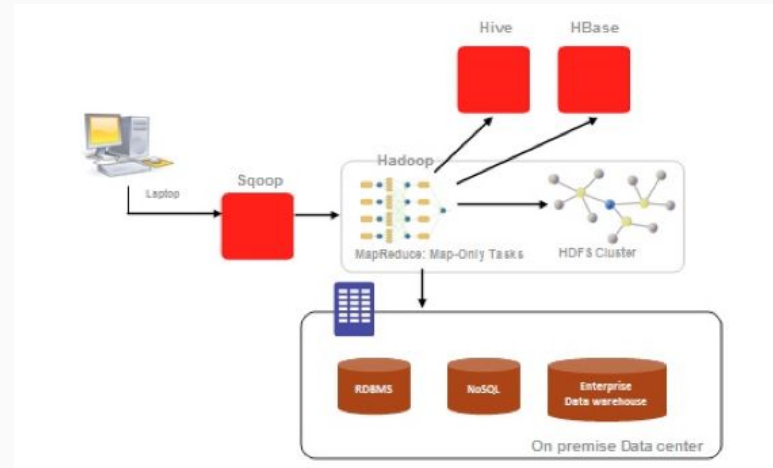
Captura y Manipulación De Datos

Sqoop

<http://sqoop.apache.org/>



Es una de las herramientas que deberías tener en cuenta si deseas potenciar tu sistema BI pues su funcionalidad permite mover grandes cantidades de datos entre Hadoop y bases de datos relacionales al mismo tiempo que ofrece integración con otros sistemas basados en Hadoop tales como Hive, HBase y Oozie . Utilizando el framework Map-Reduce transfiere los datos del DW en paralelo hacia los distintos Clústeres de manera que una vez ahí puede realizar análisis más potentes que el análisis tradicional.



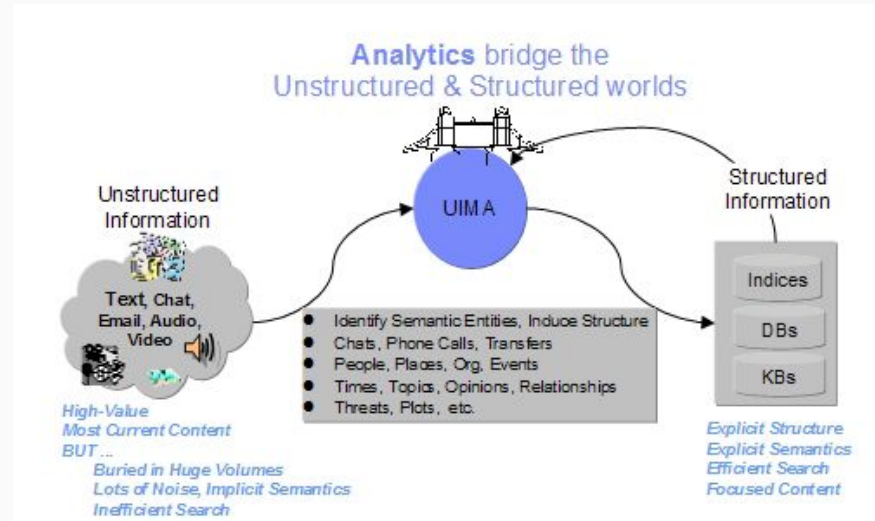
Captura y Manipulación De Datos

Uima

<https://uima.apache.org/>



Otra aplicación interesante con la que podremos analizar grandes volúmenes de datos no estructurados tales como texto, video, datos de audio, imágenes, etc... y obtener conocimiento que sea relevante para el usuario final. Por ejemplo a partir de un fichero plano, poder descubrir qué entidades son personas, lugares, organizaciones, etc...



Lucene

<https://lucene.apache.org/core/>



Un gran proyecto, una librería escrita en Java diseñada como un motor de búsqueda de textos., la cual adecuada para casi cualquier aplicación que requiera la búsqueda de texto completo. Lucene permite indexar cualquier texto o palabra (el texto puede contener letras, enteros, reales, fechas y combinaciones) permitiéndonos después encontrarlos basados en criterios de búsquedas como palabra clave, términos, frases, comodines y muchas más.

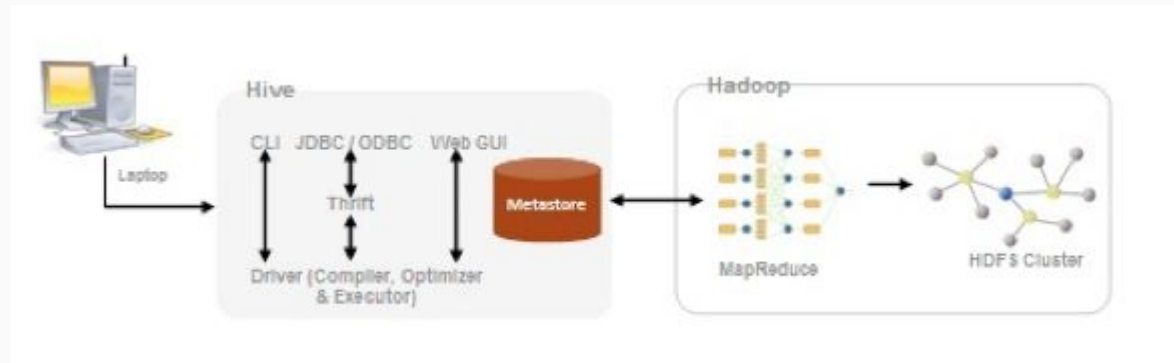
Almacenamiento



Hive

<http://hive.apache.org/>

Es una herramienta data warehousing que facilita la creación, consulta y administración de grandes volúmenes de datos almacenados en Hadoop. Cuenta con su propio lenguaje derivado del SQL, conocido como Hive QL, el cual permite realizar las consultas sobre los datos utilizando MapReduce para poder paralelizar las tareas. Por esta misma razón, se dice que Hive lleva las bases de datos relacionales a Hadoop. Otra gran ventaja es nuestro camino a la evolución del BI es que posee drivers de conexión tales como JDBC/ODBC por lo que facilita notablemente la integración con nuestros sistemas proporcionandonos extensión en análisis y procesamiento sin cargar el proceso diario..



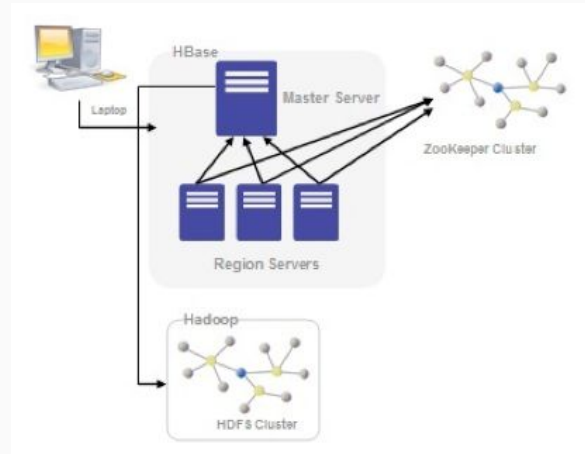
Almacenamiento

HBase

<http://hbase.apache.org/>



Es la base de datos de Hadoop distribuida y escalable. Su principal uso se encuentra cuando se requieren escrituras/lecturas en tiempo real y acceso aleatorio para grandes conjuntos de datos. Debido a que su base es Hadoop adquiere las sus capacidades y funciona sobre HDFS. Puedes almacenar en un ambiente distribuido tablas sumamente grandes incluso hablando de billones de registros por millones de columnas, la manera de soportar esta cantidad de datos es debido a que es una base NoSQL de tipo Columnar por lo cual no es posible realizar consultas SQL.



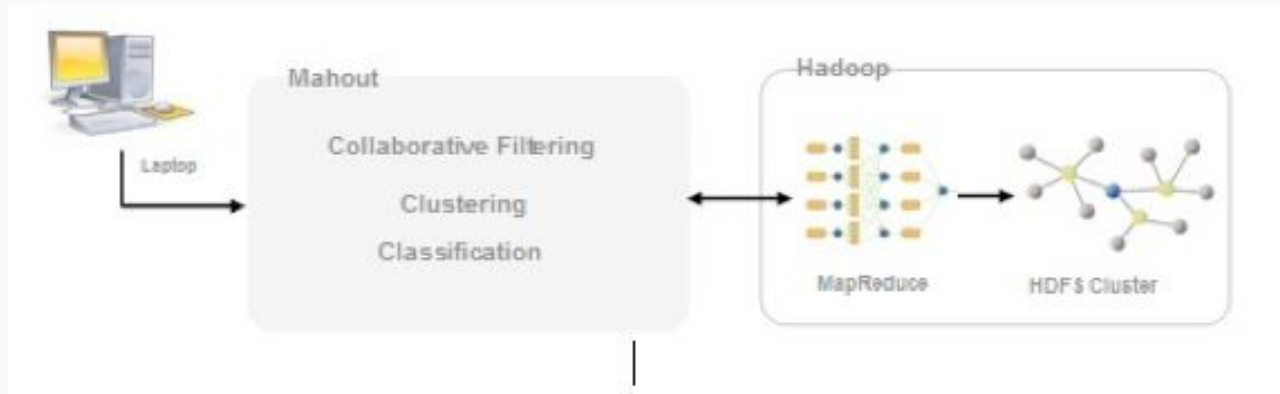
Tratamiento de datos

Mahout

<http://mahout.apache.org/>



Este proyecto nos permite desarrollar algoritmos escalables de Machine Learning y Data Mining sobre Hadoop. Soporta algoritmos como recomendación, clustering, clasificación y filtrado colaborativo, también si es el caso podremos crear algoritmos para encontrar patrones, que aprendan sobre los datos y que los clasifique una vez termine su fase de aprendizaje.



Tratamiento de datos

Jaql <https://www.ibm.com/developerworks/jaql>

Es un lenguaje de consulta funcional y declarativa que permite la manipulación y procesamiento de datos en formato JSON e incluso semi-estructurados. Fue creado y liberado por IBM bajo Apache License 2.0. Además de ser pensado para trabajar con formato JSON también permite realizar consultas sobre XML, CSV, Archivos Planos y RDBMS. Ya que es compatible con Hadoop puede ejecutar consultas de datos sobre HDFS generando automáticamente Jobs Map-Reduce solo cuando sea necesario y soportando procesamiento en paralelo sobre el Clúster.



Pig <http://pig.apache.org/>

Este proyecto nos permite analizar grandes volúmenes de datos mediante el uso de su propio lenguaje de alto nivel llamado PigLatin. Sus inicios fueron en Yahoo donde sus desarrolladores pensaban que el Map-Reduce era de muy bajo nivel y muy rígido por lo cual podías tardar mucho tiempo en la elaboración y manutención. Así pues nace Pig con su propio lenguaje y trabaja sobre Hadoop traduciendo las consultas del usuario a Map-Reduce sin que éste siquiera lo note. De esta manera provee un entorno fácil de programación convirtiendo las paralelizaciones en dataflows, un concepto mucho más sencillo para el usuario del negocio.

Pig tiene dos componentes: su lenguaje PigLatin y su entorno de ejecución.

Pig provee un enfoque más analítico que a la construcción.

Debido a su fácil y potente uso es usado en procesos de ETL y en la manipulación y análisis de datos crudos.

Tratamiento de datos

Oozie

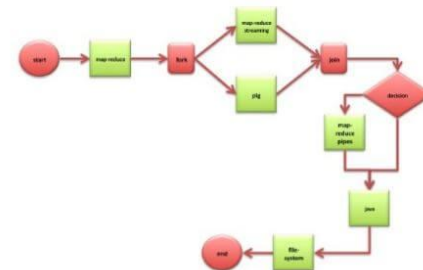
<http://oozie.apache.org/>



Proyecto el cual nos permite planificar workflows para soluciones que realizan procesos o tareas Hadoop. Al igual que Pig, está orientado al usuario no experto por lo cual le permite definir fácilmente flujos de trabajo complejos sobre los datos.

Oozie funciona como un motor de workflows a modo de servicio que permite lanzar, parar, suspender, retomar y volver a ejecutar una serie de trabajos Hadoop (tales como Java Map-Reduce, Streaming Map-Reduce, Pig, Hive, Sqooq...) basándose en ciertos criterios, como temporales o de disponibilidad de datos. Los flujos de trabajo Oozie son grafos no cíclicos directos -también conocidos como DAGs- donde cada nodo es un trabajo o acción con control de dependencia, es decir, que una acción no puede ejecutarse a menos que la anterior haya terminado.

Apache Oozie For Defining Workflows



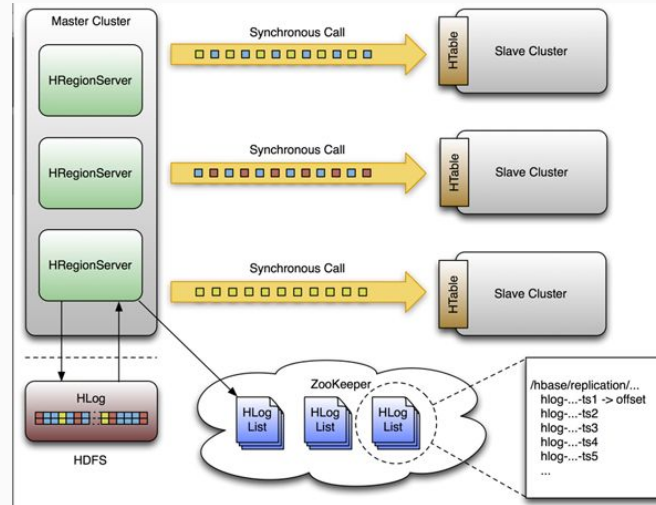
Administración

Zookeeper

<https://zookeeper.apache.org/>



Es un proyecto de Apache el cual brinda una infraestructura centralizada y servicios que permiten la sincronización del clúster. Zookeeper en pocas palabras se encarga de administrar y gestionar la coordinación entre los distintos procesos de los sistemas distribuidos.



Hue

<http://gethue.com/>



Hue es una herramienta enfocada en los administradores de las distribuciones Hadoop proporcionando una interfaz web para poder trabajar y administrar las distintas herramientas instaladas. Desde aquí puedes cargar o visualizar datos, programar y ejecutar consultas Pig o SQL, realizar búsquedas e incluso programar en pocos pasos un flujo de datos.

Una funcionalidad que independientemente de las herramientas que tengas instaladas en tu proyecto Big Data con Hadoop no puede faltar.

Avro

<https://avro.apache.org/>



Avro, es un sistema de serialización de datos creado por Doug Cutting, el padre de Hadoop. Debido a que podemos encontrar distintos formatos de datos dentro de Hadoop, Avro se ocupa de que dichos formatos puedan ser procesados por distintos lenguajes de programación por ejemplo Java, C, C++, Python, Ruby y C#. El formato que utiliza para serializar el JSON gracias a su portabilidad y fácil lectura..

HDFS

Introducción a HDFS

HDFS es el sistema de almacenamiento, es un sistema de ficheros distribuido, creado a partir del Google File System (GFS). HDFS se encuentra optimizado para grandes flujos y trabajar con ficheros grandes en sus lecturas y escrituras. Su diseño reduce la E/S en la red. La escalabilidad y disponibilidad son otras de sus claves, gracias a la replicación de los datos y tolerancia a los fallos.

Los elementos importantes del cluster son :

NameNode: Sólo hay uno en el cluster. Regula el acceso a los ficheros por parte de los clientes. Mantiene en memoria la metadata del sistema de ficheros y control de los bloques de fichero que tiene cada DataNode.

DataNode: Son los responsables de leer y escribir las peticiones de los clientes. Los ficheros están formados por bloques, estos se encuentran replicados en diferentes nodos.

HDFS :: Características

- El sistema de archivos HDFS está escrito en Java, basado en Google GFS.
- Permite tener como sistemas de archivos nativos a Ext3, xfs...
- Permite usar almacenamiento para cantidades de datos masivos, usando computadoras baratas y de baja gama.
- HDFS trabaja mejor con archivos de poco tamaño, ya que a menudo vamos a tener millones o miles de millones de archivos. Cada archivo pesa sobre los 100Mb o más.
- Los archivos en HDFS están escritos solo una vez, y no se permiten la escritura aleatoria en archivos.
- Por otra parte HDFS está optimizado para grandes lecturas de archivos Streaming, a menudo son lecturas aleatorias.

HDFS :: Características

- Es adecuado para el almacenamiento y procesamiento distribuido.
- Hadoop proporciona una interfaz de comandos para interactuar con HDFS.
- Los servidores de namenode datanode y ayudan a los usuarios a comprobar fácilmente el estado del clúster.
- Streaming el acceso a los datos del sistema de ficheros.
- HDFS proporciona permisos de archivo y la autenticación.

HDFS :: ¿Cómo son almacenados los archivos?

Los archivos se dividen en Bloques. Los Datos son distribuidos a través de muchas máquinas cuando son cargados. Un mismo archivo puede tener Bloques diferentes almacenados en distintas computadoras y esto se debe a que nos proporciona un procesamiento más eficiente para la operación MapReduce.

Los Bloques son replicados a través del mayor número de computadoras, conocidas como DataNodes. Por defecto, se hace una réplica con factor igual a 3. Por ejemplo un mismo Bloque puede estar en tres máquinas distintas.

El Nodo maestro se llamará NameNode y mantendrá la información de qué Bloques hacen un archivo, además de donde están localizados. A esto se le conoce como Metadata.

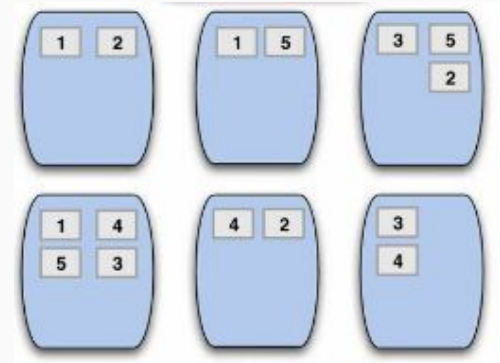
HDFS :: ¿Cómo son almacenados los archivos?

NameNode: almacena solamente los Metadatos de los archivos



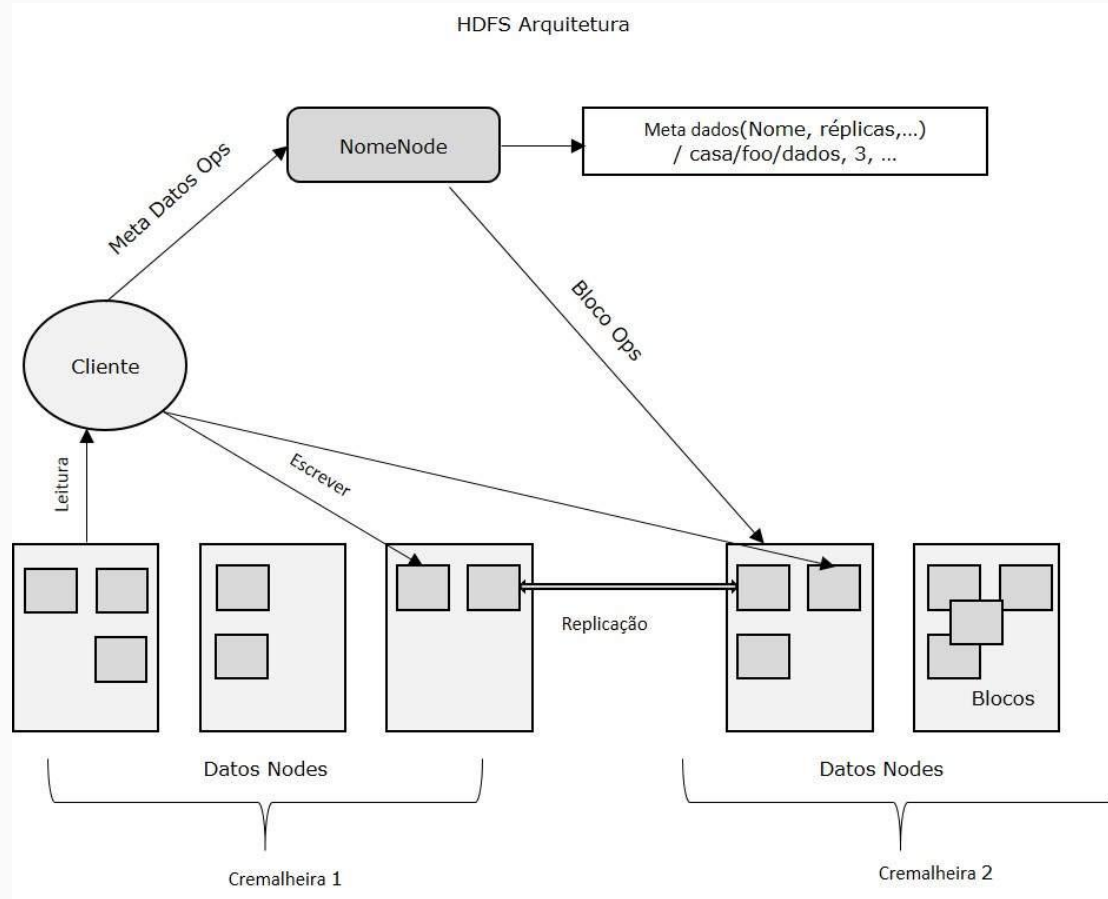
DataNode: almacena los Bloques actuales.

Cada bloque se replica 3 veces en el cluster.



Cliente lee un archivo: se comunica con el NameNode para determinar cuáles de los bloques hacen un archivo, y en cuáles de los DataNodes se almacenan. Entonces se comunica directamente con el DataNode para leer los datos.

HDFS :: Arquitectura



HDFS Datanode

- DataNode es responsable de almacenar los datos reales en HDFS.
- También se conoce como el esclavo
- NameNode y DataNode están en constante comunicación.
- Cuando un DataNode se inicia, se anuncia al NameNode junto con la lista de bloques de los que es responsable.
- Cuando un DataNode está inactivo, no afecta la disponibilidad de datos o el clúster. NameNode organizará la replicación para los bloques administrados por el DataNode que no está disponible.
- Generalmente se configura con una gran cantidad de espacio en el disco duro. Porque los datos reales se almacenan en el nodo de datos.

HDFS :: Namenode

- Es la pieza central de HDFS.
- También se conoce como el maestro
- Solo almacena los metadatos de HDFS, el árbol de directorios de todos los archivos en el sistema de archivos y rastrea los archivos en el clúster.
- No almacena los datos reales o el conjunto de datos. Los datos en sí mismos se almacenan en los DataNodes.
- Conoce la lista de bloques y su ubicación para cualquier archivo dado en HDFS. Con esta información, NameNode sabe cómo construir el archivo desde bloques.
- Es tan crítico para HDFS y cuando NameNode está inactivo, el clúster HDFS / Hadoop es inaccesible y se considera inactivo.
- Es un *punto único de falla* en el clúster de Hadoop.
- Generalmente se configura con mucha memoria (RAM). Porque las ubicaciones de los bloques son de ayuda en la memoria principal.

HDFS :: Bloque

En general los datos de usuario se almacenan en los archivos de HDFS. El archivo en un sistema de archivos se divide en uno o más segmentos y/o almacenados en los nodos de datos. Estos segmentos se denominan como bloques.

En otras palabras, la cantidad mínima de datos que HDFS puede leer o escribir se llama un bloque.

El tamaño de bloque por defecto es de 64 MB, pero puede ser aumentado por la necesidad de cambiar de configuración HDFS.

- **Detección de fallos y recuperación** : Desde los HDFS incluye un gran número de componentes de hardware, fallos de componentes es frecuente. HDFS Por lo tanto debe contar con mecanismos para una rápida y automática detección de fallos y recuperación.
- **Ingentes conjuntos** : HDFS debe tener cientos de nodos por clúster para administrar las aplicaciones de grandes conjuntos de datos.
- **Hardware a una velocidad de transferencia de datos** : una tarea solicitada se puede hacer de una manera eficiente, cuando el cálculo se lleva a cabo cerca de los datos. Especialmente en los casos en que grandes conjuntos de datos se trata, reduce el tráfico de red y aumenta el rendimiento.

HDFS :: Comandos

User Commands

- classpath
- dfs
- fetchdt
- fsck
- getconf
- groups
- lsSnapshottableDir
- jmxget
- oev
- oiv
- oiv_legacy
- snapshotDiff
- version

Administration Commands

- balancer
- cacheadmin
- crypto
- datanode
- dfsadmin
- haadmin
- journalnode
- mover
- namenode
- nfs3
- portmap
- secondarynamenode
- storagepolicies
- zkfc

Hadoop YARN

(Yet Another Resource Negotiator)

Introducción a YARN

Apache Hadoop YARN es una tecnología de administración de clústeres, la misma es una de las características clave de la segunda generación de la versión Hadoop 2 del marco de procesamiento distribuido de código abierto de Apache Software Foundation.

Originalmente descrito por Apache como un gestor de recursos rediseñado, YARN se caracteriza ahora como un sistema operativo distribuido, a gran escala, para aplicaciones de big data.

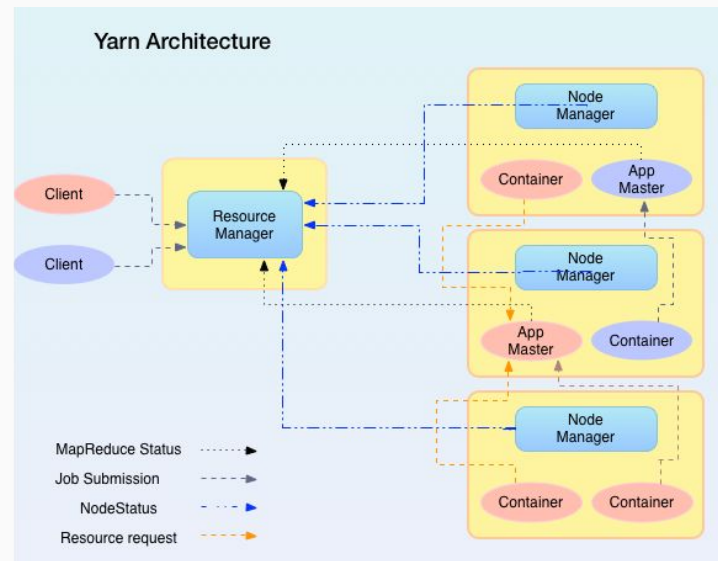
Introducción a YARN

En 2012, YARN se convirtió en un subproyecto de Apache Hadoop. A veces llamado MapReduce 2.0 (MRv2), YARN es una reescritura de software que desacopla las capacidades de gestión de recursos y planificación de MapReduce del componente de procesamiento de datos, permitiendo a Hadoop soportar enfoques más variados de procesamiento, y una gama más amplia de aplicaciones.

Por ejemplo, los clusters Hadoop ahora pueden ejecutar consultas interactivas y transmisiones de aplicaciones de datos de forma simultánea con los trabajos por lotes de MapReduce. La encarnación original de Hadoop empareja de cerca al sistema de archivos distribuidos Hadoop (HDFS) con el marco de programación MapReduce orientado a lotes, que se ocupa de la gestión de recursos y la planificación de tareas en los sistemas Hadoop, y soporta el análisis y la condensación de conjuntos de datos en paralelo.

YARN :: Arquitectura

- **Resource Manager** provee un endpoint a los clientes para hacer la solicitud de trabajos. Tiene ApplicationsManager incorporados que gestionan los trabajos en el cluster.
- **Node Manager** hay uno por nodo esclavo, es el responsable de la monitorización y gestión de los recursos. Recoge las directrices del ResourceManager y crea contenedores basado en los requerimientos de la tarea.
- **Application master** se despliega junto al **NodeManager**. Es creado por Job y controla la monitorización y la ejecución de las tareas usando el contenedor. Negocio los requerimientos de los recursos para el Job con el ResourceManager y tiene la responsabilidad de completar las tareas. proporciona la tolerancia a fallos a nivel de tarea.



YARN :: Arquitectura

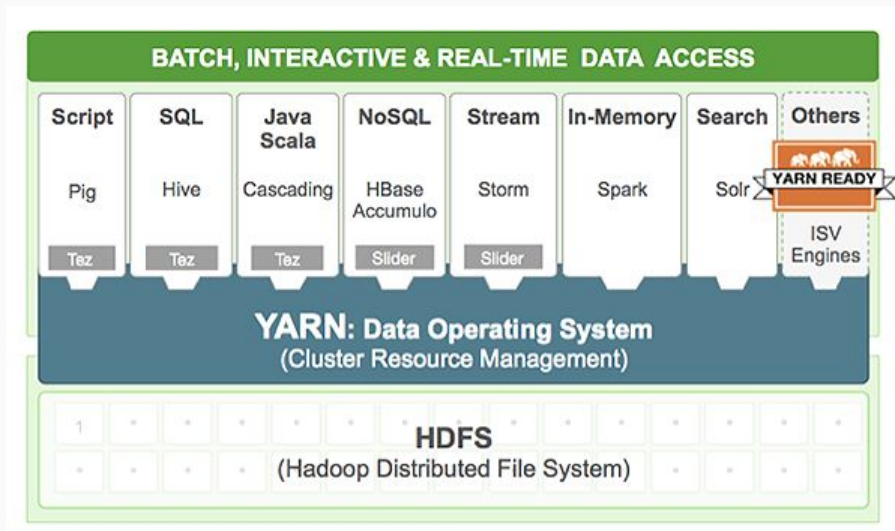
- Container es la unidad básica de la asignación en lugar de un Map o Reduce slot en Hadoop 1.x. El contenedor se define con atributos como la memoria, CPU, disco etc,... aplicaciones como procesamiento gráfico y MPI.
- History Server mantiene la historia de todos los Jobs

YARN :: Arquitectura

El hecho de separar la gestión de los recursos de la gestión de las “aplicaciones” hace posible que puedan coexistir varios modelos de computación distribuida en un mismo cluster. Esto hace que se reutilicen los cluster, reduciendo su número y facilitando su gestión (y por ende, su coste).

Esto hace que YARN se haya convertido en una especie de Sistema Operativo para Big Data de manera que varios modelos de procesamiento distribuido de datos puedan coexistir, entre ellos tenemos:

- MapReduce, Hive/Pig
- Spark
- Apache Hama
- Apache Giraph
- Solr
- Generic Co-Processors for Apache Hbase



YARN

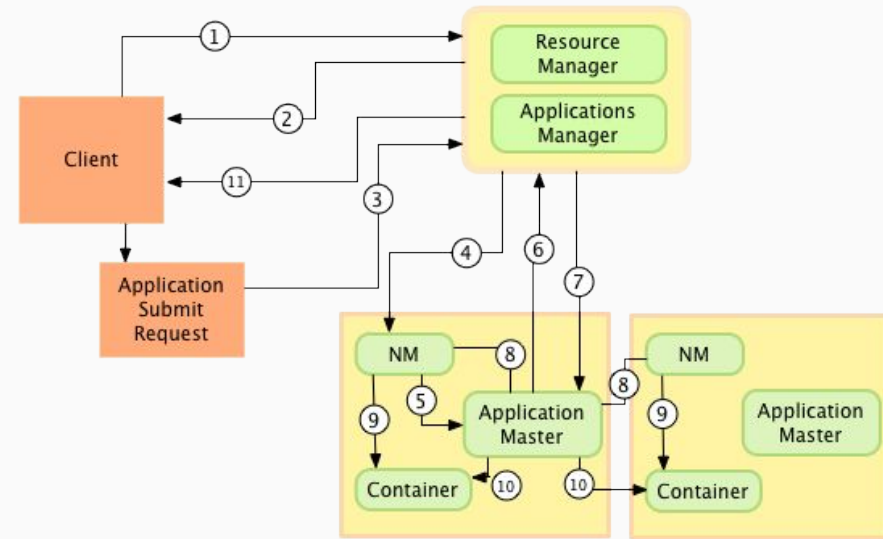
[1] Cliente comunica con el Resource Manager con una nueva Application Request

[2] Resource Manager responde con Application Id

[3] Cliente construye la petición de Application Submission request con detalles de requerimientos como memoria, cpu's, prioridad, etc... La Application Request puede tener el context para el Job como los jars de la aplicación

[4] Applications Manager una vez recibida la petición desde el cliente hace la petición al Node Manager para crear un Application Master por Job

[5] Node Manager crea el Application Master



1 - Application Request

2 - Response - Application Id

3 - Submit Application Launch Request
with application parameters

4 - Start Application Master

5 - Launch Application Master

6 - Allocate Containers

7 - Response - List of containers

8 - Request Launch Container

9 - Create Container

10 - Application Master manages job
execution

11 - Request Application status

YARN

[6] Application Master crea la petición para la asignación de recursos al Resource Manager. Application Master es responsable para la ejecución del Job hasta que se completa.

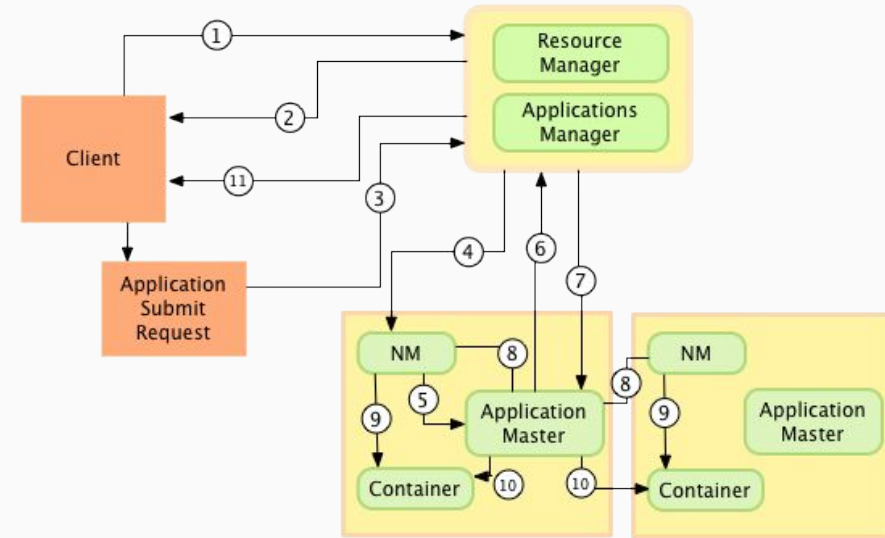
[7] El ResourceManager devuelve una lista de contenedores.

[8] Application Master pide al Node Manager lanzar los contenedores para ese Job concreto.

[9] Node Manager crea el contenedor. Contenedor ejecuta el código cliente específico en el contenedor.

[10] Application Master gestiona la ejecución de Job hasta que se complete el Job.

[11] Client pide el reporte de status de la aplicación



- | | |
|--|--|
| 1 - Application Request | |
| 2- Response - Applciation Id | |
| 3 - Submit Application Launch Request
with application parameters | |
| 4 - Start Application Master | 8- Request Launch Container |
| 5 - Launch Application Master | 9 - Create Container |
| 6 - Allocate Containers | 10 - Application Master manages job
execution |
| 7- Response - List of containers | 11 - Request Application status |

Configurando un entorno de desarrollo Hadoop

Modos de configuración

Un único nodo en local (single node), utilizado para hacer pruebas de concepto corriendo Hadoop en una misma máquina

Un cluster pseudo-distribuido para simular un cluster de varios nodos pero corriendo en una misma máquina.

Montar un cluster entre distintas máquinas (multi node) totalmente distribuido que sería el modo que utilizamos para montar un sistema Big Data en producción.

Configurando un entorno

A continuación detallaremos la instalación de una instancia del tipo single para los posteriores ejemplos.

El mismo será mediante vagrant que nos permitirá replicar en cualquier ordenador sin problemas de versiones, dependencias y

Configurando un entorno

Verificar que Java esté instalado:

```
java -version
```

```
openjdk version "1.8.0_141"
```

```
OpenJDK Runtime Environment (build 1.8.0_141-8u141-b15-3~14.04-b15)
```

```
OpenJDK 64-Bit Server VM (build 25.141-b15, mixed mode)
```

Descargamos hadoop:

```
curl -O http://apache.javapipe.com/hadoop/common/hadoop-2.8.1/hadoop-2.8.1.tar.gz
```

```
tar -xvzf hadoop-2.8.1.tar.gz
```

```
sudo mv hadoop-2.8.1 /usr/local/
```

```
sudo mv /usr/local/hadoop-2.8.1 /usr/local/hadoop
```

```
rm hadoop-2.8.1.tar.gz
```

Creación del usuario Hadoop

El usuario Hadoop se utiliza para administrar los servicios del HDFS.

Ejecutar el siguiente comando para crear el usuario y darle permisos administrativos:

```
sudo useradd -d /home/hadoop -m hadoop
echo -e "H4d00p\nH4d00p\n" | sudo passwd hadoop
sudo usermod -aG sudo hadoop
sudo usermod -s /bin/bash hadoop
mkdir /tmp/hadoop-namenode
mkdir /tmp/hadoop-logs
mkdir /tmp/hadoop-datanode
sudo chown -Rf hadoop:hadoop /usr/local/hadoop /tmp/hadoop-*
```

Luego se ingresa al sistema con este usuario:

```
su - hadoop
```

Creación del usuario Hadoop

Agregar al final del archivo `$HOME/.bashrc` del usuario donde se instala Hadoop las siguientes líneas:

```
cat <<EOT >> $HOME/.bashrc
export HADOOP_HOME=/usr/local/hadoop
export PATH=\$PATH:\$HADOOP_HOME/bin:\$HADOOP_HOME/sbin
export HADOOP_MAPRED_HOME=\${HADOOP_HOME}
export HADOOP_COMMON_HOME=\${HADOOP_HOME}
export HADOOP_HDFS_HOME=\${HADOOP_HOME}
export YARN_HOME=\${HADOOP_HOME}
EOT
```

Estas líneas son las variables de entorno del HDFS, le especifican al sistema dónde encontrar los archivos que necesita para la ejecución y configuración del HDFS.

Configuración de red y SSH

Los nodos de Hadoop deberán poder conectarse entre sí mediante una conexión ssh (sin contraseña o con una misma contraseña para todos). Para esto, desde el usuario donde se está instalando hadoop se crea una clave pública ssh que se compartirá con los demás nodos.

```
echo -e 'y\n'|ssh-keygen -q -t rsa -N "" -f ~/.ssh/id_rsa  
cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

Configuración de red y SSH

Le permisos de lectura al archivo de autorización de conexión por ssh:

```
chmod go-w $HOME $HOME/.ssh  
chmod 600 $HOME/.ssh/authorized_keys  
chown `whoami` $HOME/.ssh/authorized_keys
```

Para probar la configuración, se realiza una conexión por ssh con el localhost:

```
ssh localhost -p 2222
```

Si todo sale bien, estaremos en una sesión por SSH sin haber ingresado una contraseña. Para salir de la sesión SSH ejecutar:

```
exit
```


Configuración de red y SSH

Ahora deshabilitamos ipv6 desde los archivos de configuración ya que Hadoop advierte en su documentación oficial que no tiene compatibilidad con el uso de ipv6. Para esto agregaremos lo siguiente al archivo `/etc/sysctl.conf`

```
su -  
cat <<EOT >> /etc/sysctl.conf  
net.ipv6.conf.all.disable_ipv6 = 1  
net.ipv6.conf.default.disable_ipv6 = 1  
net.ipv6.conf.lo.disable_ipv6 = 1  
EOT
```

Configuración de ficheros Hadoop

Los archivos principales de configuración del HDFS se encuentran en el directorio `/usr/local/hadoop/etc/hadoop`. Allí se modificarán varios archivos.

Los archivos más relevantes en la configuración son:

- `core-site.xml`
- `hdfs-site.xml`
- `hadoop-env.sh`
- `mapred-site.xml`
- `yarn-site.xml`

Archivo core-site.xml

El fichero `core-site.xml` informa al *daemon Hadoop* donde se ejecuta el *NameNode* dentro de un *cluster*. Contiene la configuración que ajusta el *Hadoop Core* así como la del I/O que es común para HDFS y MapReduce.

Primero editaremos el archivo `core-site.xml` con lo siguiente:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
  <property>
    <name>fs.default.name</name>
    <value>hdfs://localhost:8020</value>
    <description>Nombre del filesystem por defecto.</description>
  </property>
</configuration>
```

Luego en el archivo `hdfs-site.xml` especificaremos que se utilizará un factor de replicación igual a 1. Nos cercioramos de que quede escrito lo siguiente:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
  <property>
    <name>dfs.namenode.name.dir</name>
    <value>file:/home/hadoop/workspace/dfs/name</value>
    <description>Path del filesystem donde el namenode almacenará los metadatos.</description>
  </property>
  <property>
    <name>dfs.datanode.data.dir</name>
    <value>file:/home/hadoop/workspace/dfs/data</value>
    <description>Path del filesystem donde el datanode almacenará los bloques.</description>
  </property>
  <property>
    <name>dfs.replication</name>
    <value>1</value>
    <description>Factor de replicación. Lo ponemos a 1 porque sólo tenemos 1 máquina.</description>
  </property>
</configuration>
```

Creación de carpetas HDFS

Creamos los directorios de trabajo

- `/home/hadoop/workspace/dfs/name`
- `/home/hadoop/workspace/dfs/data`

```
mkdir -p /home/hadoop/workspace/dfs/name
```

```
mkdir -p /home/hadoop/workspace/dfs/data
```

(verifique el owner y los permisos de los directorios que sean accesibles para el user 'hadoop')

Cambiamos el valor de JAVA_HOME en el archivo hadoop-env.sh

```
export JAVA_HOME=${JAVA_HOME}
```

por

```
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
```

```
(sudo update-alternatives --config javac)
```

Archivo mapred-site.xml

Hacemos una copia del archivo por defecto `mapred-site.xml.template` hacia `mapred-site.xml` :

```
cp mapred-site.xml.template mapred-site.xml
```

Y colocamos la siguiente configuración.

```
<configuration>
  <property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
  </property>
</configuration>
```

Luego creamos los directorios `/home/hadoop/workspace/mapred/system` y `/home/hadoop/workspace/mapred/local`

```
mkdir -p /home/hadoop/workspace/mapred/system  
mkdir -p /home/hadoop/workspace/mapred/local
```

(verifique el owner y los permisos de los directorios que sean accesibles para el user 'hadoop')

Entre las etiquetas `<configuration></configuration>` dentro del archivo `yarn-site.xml` se agrega lo siguiente:

```
<property>
  <name>yarn.nodemanager.aux-services</name>
  <value>mapreduce_shuffle</value>
</property>
<property>
  <name>yarn.nodemanager.aux-services.mapreduce_shuffle.class</name>
  <value>org.apache.hadoop.mapred.ShuffleHandler</value>
</property>
```

Formatear el NameNode

Formatea el sistema de ficheros HDFS.

```
hdfs namenode -format
```

(verifique mediante 'env' las variables de entorno están correctas, sino vuelva a realizar su -hadoop)

Lanzando Hadoop

Se ejecuta los siguientes comandos uno después del otro.

```
start-dfs.sh
```

```
start-yarn.sh
```

luego mediante `jps` podremos verificar si se encuentran en ejecución

```
22642 Jps  
22323 ResourceManager  
22423 NodeManager  
22014 DataNode  
21887 NameNode  
22175 SecondaryNameNode
```



Logged in as: dr.who

All Applications

Cluster

[About](#)
[Nodes](#)
[Node Labels](#)
[Applications](#)
NEW
NEW SAVING
SUBMITTED
ACCEPTED
RUNNING
FINISHED
FAILED
KILLED

[Scheduler](#)

Tools

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved
0	0	0	0	0	0 B	8 GB	0 B	0	8	0

Cluster Nodes Metrics

Active Nodes	Decommissioning Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes	Shutdown Nodes
1	0	0	0	0	0	0

Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation	Maximum Cluster Application Priority
Capacity Scheduler	[MEMORY]	<memory:1024, vCores:1>	<memory:8192, vCores:4>	0

Show 20 entries

Search:

ID	User	Name	Application Type	Queue	Application Priority	StartTime	FinishTime	State	FinalStatus	Running Containers	Allocated CPU VCores	Allocated Memory MB	% of Queue	% of Cluster	Progress	Tracking UI	Blacklisted Nodes
▼	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕

No data available in table

Showing 0 to 0 of 0 entries

First Previous Next Last

http://localhost:8042/

Hadoop Overview Datanodes Datanode Volume Failures Snapshot Startup Progress Utilities

Overview 'localhost:8020' (active)

Started:	Wed Nov 22 04:14:43 +0100 2017
Version:	2.8.1, r20fe5304904fc2f5a18053c389e43cd26f7a70fe
Compiled:	Fri Jun 02 08:14:00 +0200 2017 by vinodkv from branch-2.8.1-private
Cluster ID:	CID-1cf26905-9a6c-4438-9993-7f692e7beb4c
Block Pool ID:	BP-258052823-127.0.0.1-1511320109138

Summary

Security is off.

Safemode is off.

1 files and directories, 0 blocks = 1 total filesystem object(s).

Heap Memory used 35.98 MB of 51.76 MB Heap Memory. Max Heap Memory is 966.69 MB.

Non Heap Memory used 41.55 MB of 42.31 MB Committed Non Heap Memory. Max Non Heap Memory is <unbounded>.

Configured Capacity:	37.45 GB
DFS Used:	4 KB (0%)
Non DFS Used:	3.43 GB
DFS Remaining:	34.02 GB (90.85%)

<http://localhost:50070>

Interfaces Web



Logged in as: dr.who

▸ ResourceManager

▾ NodeManager

[Node
Information
List of
Applications
List of
Containers](#)

▸ Tools

NodeManager information

Total Vmem allocated for Containers	16.80 GB
Vmem enforcement enabled	true
Total Pmem allocated for Container	8 GB
Pmem enforcement enabled	true
Total VCores allocated for Containers	8
NodeHealthyStatus	true
LastNodeHealthTime	Wed Nov 22 14:09:25 UTC 2017
NodeHealthReport	
NodeManager started on	Wed Nov 22 03:15:17 UTC 2017
NodeManager Version:	2.8.1 from 20fe5304904fc2f5a18053c389e43cd26f7a70fe by vinodkv source checksum 2517569efbba43f05f7e51f978fe1fac on 2017-06-02T06:19Z
Hadoop Version:	2.8.1 from 20fe5304904fc2f5a18053c389e43cd26f7a70fe by vinodkv source checksum 60125541c2b3e266cbf3becc5bda666 on 2017-06-02T06:14Z

Hadoop Command Line

Usage: `hadoop [--config confdir] [--loglevel loglevel] [COMMAND] [GENERIC_OPTIONS] [COMMAND_OPTIONS]`

User Commands

- `archive`
- `checknative`
- `classpath`
- `credential`
- `distcp`
- `fs`
- `jar`
- `key`
- `trace`
- `version`
- `CLASSNAME`

Administration Commands

- `daemonlog`

Testing Hadoop Instalación

A continuación testeamos la instalación mediante el uso del los comando

```
hadoop fs
```


Creamos un una carpeta en HDFS

Crear un directorio o carpeta dentro del sistema de archivos HDFS de manera análoga al comando `mkdir` de UNIX.

```
hadoop fs -mkdir /carpeta
```

Damos permisos un una carpeta

Dar permisos a un archivo (O carpeta) de manera análoga al comando `chmod` de UNIX.

```
hadoop fs -chmod permisos archivo
```

Donde archivo será la ruta al archivo que se le aplicará el cambio.

Dar permisos de lectura y escritura en `/user/example`.

```
hadoop fs -chmod +rw /user/example
```

Listar contenido un una carpeta

Listar los archivos y subdirectorios de un directorio de forma análoga al comando ls de UNIX.

```
hadoop fs -ls archivo
```

Donde archivo es la ruta al directorio que se desea listar.

Ejemplo

```
hadoop fs -ls /user/example
```

Copiar ficheros en HDFS

Para enviar un archivo local hacia el sistema de archivos HDFS se utiliza el siguiente comando.

```
hadoop fs -copyFromLocal myFile destinyFile
```

Donde “myFile” es la ruta del archivo que se desea subir y “destinyFile” es la ruta donde se desea almacenar el archivo en el HDFS.

Ejemplo

```
hadoop fs -copyFromLocal archivo.txt /user/example
```

Inserción de datos en Hadoop con Java

Funcionamiento del HDFS Java API

La clase `org.apache.hadoop.fs.FileSystem` es una clase genérica para acceder y administrar archivos / directorios HDFS ubicados en un entorno distribuido.

El contenido del archivo almacenado dentro del Datanode con múltiples bloques de tamaños iguales (por ejemplo, 64 MB) y Namenode conservan la información de esos bloques y la información Meta.

`FileSystem` lee y transmite accediendo a los bloques en orden de secuencia, obteniendo primero información de bloques de `NameNode` y luego abre, lee y cierra uno por uno. Abre los primeros bloques una vez que se completa, luego cierra y abre el siguiente bloque.

HDFS Java API

HDFS replica el bloque para brindar mayor confiabilidad y escalabilidad, y si el cliente es uno de los nodos de datos, entonces intenta acceder al bloque localmente si falla y luego pasa a otro nodo de datos del clúster.

`FileSystem` utiliza `FSDataOutputStream` y `FSDataInputStream` para escribir y leer los contenidos en la transmisión. Hadoop ha proporcionado varias implementaciones de `FileSystem` como se describe a continuación:

`DistributedFileSystem`: para acceder al archivo HDFS en un entorno distribuido.

`LocalFileSystem`: para acceder al archivo HDFS en el sistema Local.

`FTPFileSystem`: para acceder al archivo HDFS cliente FTP.

`WebHdfsFileSystem`: para acceder al archivo HDFS a través de la web.

HDFS Java API :: URI y Path

URI:

El URI de Hadoop ubica la ubicación del archivo en HDFS. Utiliza `hdfs: // host: puerto / ubicación` para acceder al archivo a través de `FileSystem`. El URI de Hadoop ubica la ubicación del archivo en HDFS. Utiliza `hdfs://host: puerto/ubicación` para acceder al archivo a través de `FileSystem`.

```
hdfs://localhost:9000/user/joe/TestFile.txt  
URI uri=URI.create (“hdfs://host: port/path”);
```

Path:

La clase `Path` consta de `URI` y resuelve la dependencia del sistema operativo en el `URI`, p. Windows usa `\\ path` mientras que linux usa `//`. También sirve para resolver la dependencia de los padres y los hijos.

HDFS Java API :: Configuration

La clase de Configuration pasa la información de configuración de Hadoop a FileSystem. Carga los ficheros core-site y core-default.xml mediante el cargador de clases y mantiene la información de configuración de Hadoop como fs.defaultFS, fs.default.name, etc.

```
Configuration conf = new Configuration ();
```

La misma puede ser explícita

```
conf.set("fs.default.name", "hdfs://localhost:9000");
```

HDFS Java API :: Uso de FileSystem

Mediante estos métodos podremos crear una instancia .

```
public static FileSystem get(Configuration conf)
public static FileSystem get(URI uri, Configuration conf)
public static FileSystem get(URI uri, Configuration conf, String user)
```

FileSystem usa NameNode para localizar el DataNode y luego acceder directamente al bloque en orden de secuencia para leer el archivo. FileSystem utiliza la interfaz Java IO FileSystem principalmente DataInputStream y DataOutputStream para la operación IO.

Si está buscando obtener un sistema de archivos local, podemos usar directamente el método getLocal como se menciona a continuación.

```
public static LocalFileSystem getLocal(Configuration conf)
```

HDFS Java API :: FSDataInputStream

FSDataInputStream ajusta el DataInputStream e implementa las interfaces Seekable, PositionedReadable que proporcionan un método como getPos(), seek() para proporcionar acceso aleatorio en un archivo HDFS.

FileSystem tiene el método open() que devuelve FSDataInputStream

```
URI uri = URI.create ("hdfs://host: port/file path");
Configuration conf = new Configuration ();
FileSystem file = FileSystem.get (uri, conf);
FSDataInputStream in = file.open(new Path(uri));
byte[] btbuffer = new byte[5];
in.seek(5); //Seekable
Assert.assertEquals(5, in.getPos()); //Seekable
in.read(btbuffer, 0, 5); //PositionedReadable
```

HDFS Java API :: FSDataOutputStream

El método `create()` de `FileSystem` devuelve `FSDataOutputStream`, que se utiliza para crear un nuevo archivo HDFS o escribir el contenido en el EOF. No proporciona búsqueda debido a la limitación de HDFS para escribir solo en EOF. Envuelve el `DataOutputStream` de Java IO y agrega un método como `getPos()` para obtener la posición del archivo y `write()` para escribir el contenido en la última posición..

```
public FSDataOutputStream create(Path f) create empty file.  
public FSDataOutputStream append(Path f) will append existing file
```

A su vez podemos crear un `FSDataOutputStream` capaz de trazar el status durante la creación;

```
public FSDataOutputStream create(Path f, Progressable progress)
```

HDFS Java API :: FileStatus

Como se describe a continuación, el método `getFileStatus()` de `FileSystem` proporciona la metainformación del archivo HDFS

```
URI uri=URI.create(strURI);
FileSystem fileSystem=FileSystem.get(uri,conf);
FileStatus fileStatus=fileSystem.getFileStatus(new Path(uri));
System.out.println("AccessTime:"+fileStatus.getAccessTime());
System.out.println("AccessTime:"+fileStatus.getLen());
System.out.println("AccessTime:"+fileStatus.getModificationTime());
System.out.println("AccessTime:"+fileStatus.getPath());
```

Y si el `Path` es un directorio podremos obtener la lista mediante

```
public FileStatus[] listStatus(Path f)
```

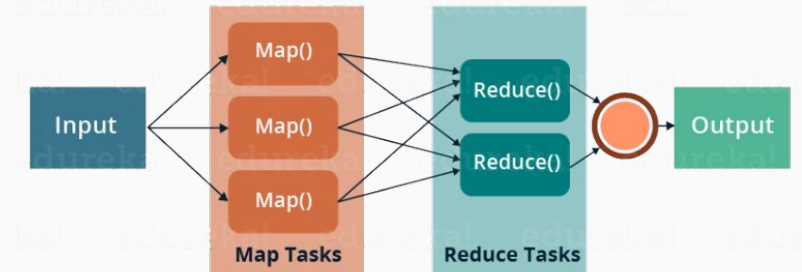
Aplicaciones MapReduce distribuidas e invertidas

Que es MapReduce

MapReduce es un marco de programación que nos permite realizar un procesamiento distribuido y paralelo en grandes conjuntos de datos en un entorno distribuido.

MapReduce consta de dos tareas distintas:

Mapear (Map) y Reducir (Reduce).



MapReduce

Como sugiere el nombre MapReduce, la fase de reducción tiene lugar una vez que se ha completado la fase de mapeo.

Entonces, el primero es el trabajo de mapa, donde un bloque de datos es leído y procesado para producir pares clave-valor como salidas intermedias.

La salida de un Mapper o trabajo de mapeo (pares clave-valor) se ingresa al Reducer.

El reductor recibe el par clave-valor de múltiples trabajos de mapa.

Luego, el reductor agrega esas tuplas de datos intermedios (par clave-valor intermedio) en un conjunto más pequeño de tuplas o pares clave-valor que es el resultado final.

MapReduce

Antes de proceder con el análisis de datos mediante JAVA MapReduce es necesario conocer los aspectos principales de la API para Hadoop.

Para ello, veremos la estructura de un programa JAVA para Hadoop MapReduce.

Como se ha explicado ampliamente, el paradigma MapReduce está constituido por dos tareas fundamentales (Map y Reduce), por tanto, cada una de estas tareas debe ser implementada cuando se realiza el código JAVA. Además de estas dos tareas, existen otras etapas intermedias que no son imprescindibles pero se profundirán como los son los combiners y partitioners.

Ventajas

Posee 2 grandes ventajas

1. Procesamiento Paralelo
2. Localidad de datos

Procesamiento Paralelo

En MapReduce, estamos dividiendo el trabajo entre varios nodos y cada nodo trabaja simultáneamente con una parte del trabajo. Entonces, MapReduce se basa en el paradigma Divide and Conquer que nos ayuda a procesar los datos usando diferentes máquinas. Como los datos son procesados por máquinas múltiples en lugar de una sola máquina en paralelo, el tiempo necesario para procesar los datos se reduce en una cantidad tremenda.

Ventajas :: Procesamiento Paralelo

En MapReduce, estamos dividiendo el trabajo entre varios nodos y cada nodo trabaja simultáneamente con una parte del trabajo.

Entonces, MapReduce se basa en el paradigma Divide and Conquer que nos ayuda a procesar los datos usando diferentes máquinas.

Como los datos son procesados por máquinas múltiples en lugar de una sola máquina en paralelo, el tiempo necesario para procesar los datos se reduce en una cantidad tremenda.

Ventajas :: Localidad de datos

En el framework de MapReduce en lugar de mover datos a la unidad de procesamiento, estamos moviendo la unidad de procesamiento a los datos. En el sistema tradicional, solíamos traer datos a la unidad de procesamiento y procesarlos. Pero, a medida que los datos crecían y se volvían muy grandes, llevar esta gran cantidad de datos a la unidad de procesamiento plantea los siguientes problemas:

- Mover datos enormes al procesamiento es costoso y deteriora el rendimiento de la red.
- El procesamiento toma tiempo ya que los datos son procesados por una sola unidad que se convierte en el cuello de botella.
- El nodo maestro puede sobrecargarse y puede fallar.

Ahora, MapReduce nos permite superar los problemas anteriores al traer la unidad de procesamiento a los datos. Entonces, como puede ver en la imagen de arriba, los datos se distribuyen entre varios nodos donde cada nodo procesa la parte de los datos que residen en él. Esto nos permite tener las siguientes ventajas:

MapReduce en Java

MapReduce, es su implementación en Java se encuentra en el y paquete :

```
org.apache.hadoop.mapreduce
```

Encontramos allí tres clases principales del API que son las clases

- JobContext Interface
- Job Class
- Mapper Class
- Reducer Class

JobContext en Java

La interfaz JobContext es la **super interface** para todas las clases, que define diferentes trabajos en MapReduce. Le proporciona una vista de solo lectura del trabajo que se proporciona a las tareas mientras se ejecutan.

Las sub-interfaces de JobContext , `MapContext<KEYIN, VALUEIN, KEYOUT, VALUEOUT>` , definen el contexto que se les da al Mapper y `ReduceContext<KEYIN, VALUEIN, KEYOUT, VALUEOUT>` al Reducer.

Clase Job

La clase Job es la implementación principal de la interface JobContext, además de ser la clase más importante en la API de MapReduce.

Le permite al usuario configurar el trabajo, enviarlo, controlar su ejecución y consultar el estado. Los métodos establecidos solo funcionan hasta que se envíe el trabajo, luego arrojarán una `IllegalStateException`.

Normalmente, el usuario crea la aplicación, describe las diversas facetas del trabajo y luego envía el trabajo y supervisa su progreso.

Clase Job :: Métodos

El objeto Job permite invocar numerosos métodos relacionados con la configuración, los principales son los siguientes:

<code>void setInputFormatClass(..);</code>	<code>void setJobName(String name);</code>
<code>void setOutputFormatClass(..);</code>	<code>float mapProgress();</code>
<code>void setMapperClass(..);</code>	<code>float reduceProgress();</code>
<code>void setCombinerClass(..);</code>	<code>boolean isSuccessful();</code>
<code>void setReducerClass(...);</code>	<code>void killJob();</code>
<code>void setPartitionerClass(..);</code>	<code>void submit();</code>
<code>void setMapOutputKeyClass(..);</code>	<code>boolean waitForCompletion(..);</code>
<code>void setMapOutputValueClass(..);</code>	
<code>void setOutputKeyClass(..);</code>	
<code>void setOutputValueClass(..);</code>	

Clase Map

La clase Mapper define el trabajo Map.

A las instancias de Map ingresan los pares clave-valor en un conjunto de pares clave-valor intermedios. Las instancias de Map son las tareas individuales que transforman los registros de entrada en registros intermedios.

Los registros intermedios transformados no necesitan ser del mismo tipo que los registros de entrada. Un par de entrada dado puede mapearse a cero o a muchos pares de salida.

El método map es el más destacado de la clase Mapper. La sintaxis se define a con la firma:

```
map(KEYIN key, VALUEIN value,Mapper.Context context)
```

Este método se llama una vez para cada par clave-valor en la división (split) de entrada.

Clase Reducer

La clase Reducer define el trabajo de reducción en MapReduce.

La misma reduce un conjunto de valores intermedios que comparten una clave para un conjunto de valores más pequeño.

Las implementaciones pueden acceder la configuración mediante el método `JobContext.getConfiguration()`.

Clase Reducer :: Fases

Un Reducer tiene 3 fases primarias: Shuffle, Sort, y Reduce.

Shuffle:

La instancia de clase Reducer copia la salida ordenada de cada asignador utilizando HTTP en toda la red.

Sort:

El framework fusiona las entradas del Reductor por claves (ya que diferentes Mappers pueden haber emitido la misma clave). Las fases de mezcla y ordenamiento ocurren simultáneamente, es decir, mientras se obtienen las salidas, se fusionan.

Reduce:

En esta fase el método `reduce(Object, Iterable, Context)` es llamado por cada `<key, (collection of values)>` en las entradas ordenadas.

Clase Reducer :: Métodos

Reduce es el método más destacado de la clase Reducer. La sintaxis se define a como:

```
reduce(KEYIN key, Iterable<VALUEIN> values, Reducer.Context context)
```

Este método se llama una vez para cada clave en la colección de pares clave-valor.

Clase principal o de configuración

En esta clase se establece toda la configuración necesaria para la correcta ejecución del programa. Además, se crea un objeto job que representa la ejecución del código. La configuración esencial se muestra seguidamente:

```
public class MyMapReduce {  
    public static void main(String[] args) {  
        Configuration conf = new Configuration();  
        Job job = JobContext.getConfiguration(conf, "MyMapReduce");  
        job.setJarByClass(MyMapReduce.class);  
        job.setOutputKeyClass(Text.class);  
        job.setOutputValueClass(IntWritable.class);  
        job.setMapperClass(MyMapReduce.Map.class);  
        job.setReducerClass(MyMapReduce.Reduce.class);  
        FileInputFormat.addInputPath(job, new Path(args[0]));  
        FileOutputFormat.setOutputPath(job, new Path(args[1]));  
        job.waitForCompletion(true);  
    }  
}
```

Clase principal o de configuración

Como se puede observar se establece toda la configuración imprescindible para la correcta ejecución del programa.

Es necesario crear un objeto `Job` e indicar el formato de salida de las parejas clave/valor. Asimismo, es necesario indicar los nombres de las clases `Map` y `Reduce`.

Por último, se indica la procedencia de los datos de entrada y donde se escriben los datos de salida mediante `addInputPath` y `setOutputPath`.