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 n añadir datos nuevos al final en vez de reemplazar los existentes.

Historia

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 frameworkHadoop 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*.

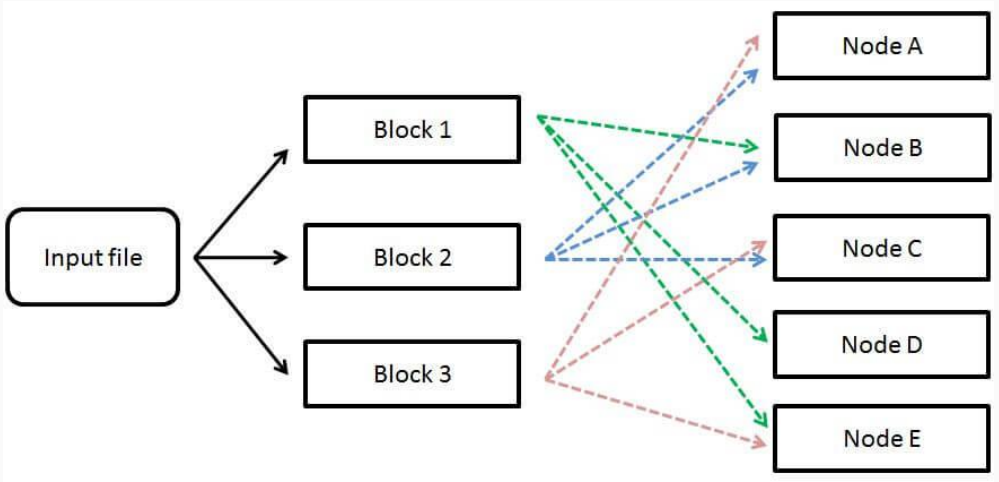
Hadoop MapReduce:
Sistema basado en YARN que nos permite de manera paralela el procesamiento de grandes conjuntos de datos.

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.



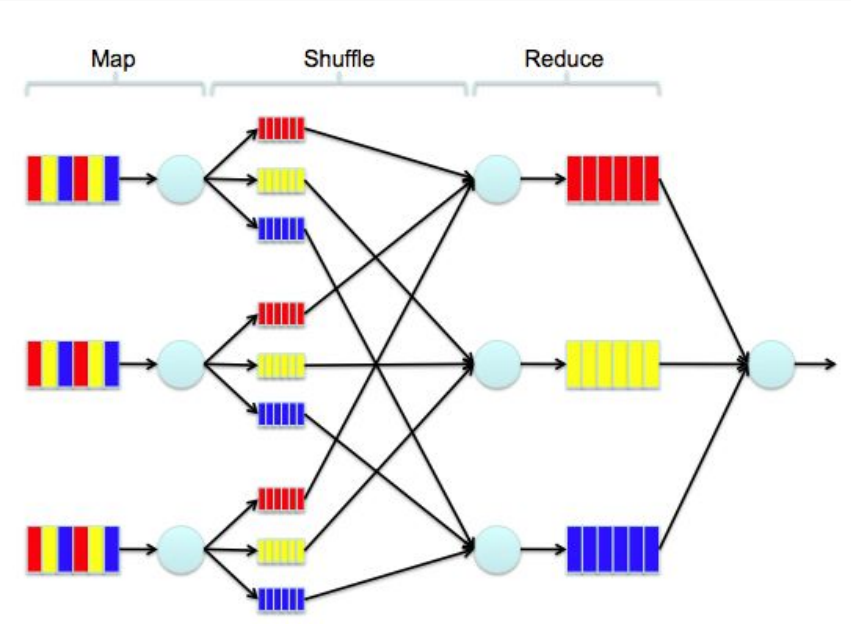
Hadoop MapReduce

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 experimentaras 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 paralización.

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

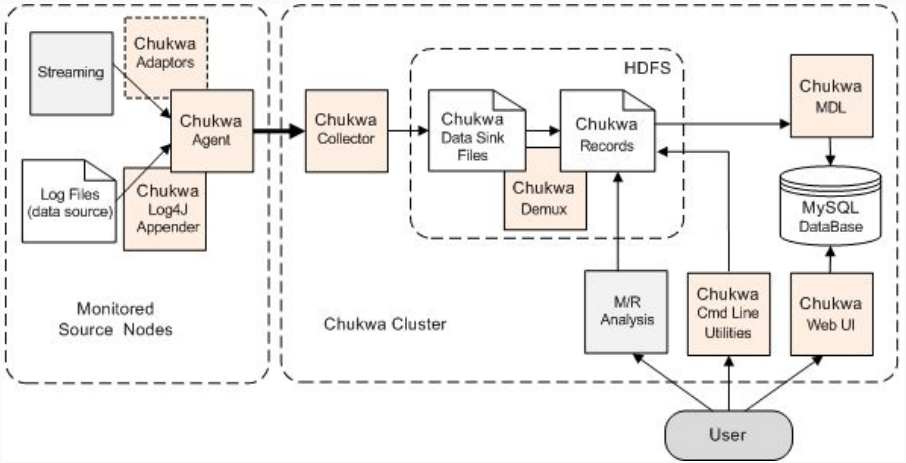
Debido a la creciente comunidad Open Source existen distintos proyectos y herramientas que ofrecen funcionalidades adicionales las cuales son consideradas parte de un gran ecosistema pensado en apoyar las distintas etapas de un proyecto Big Data.

A continuación enumeramos parte de estas.

Captura y Manipulación De Datos

Chukwa http://chukwa.apache.org/

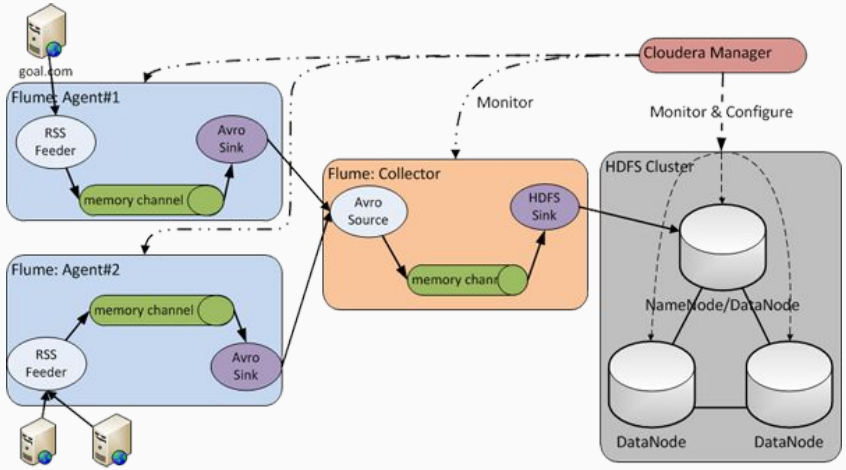
Es un proyecto construido para capturar y analizar grandes volúmenes de datos principalmente logs. Debido a que está construido sobre Hadoop hereda escalabilidad y robustez con el uso de HDFS y Map-Reduce al mismo tiempo que provee adicionalmente un grupo de herramientas flexibles y potentes para visualizar,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



Captura y Manipulación De Datos

Flume https://flume.apache.org/

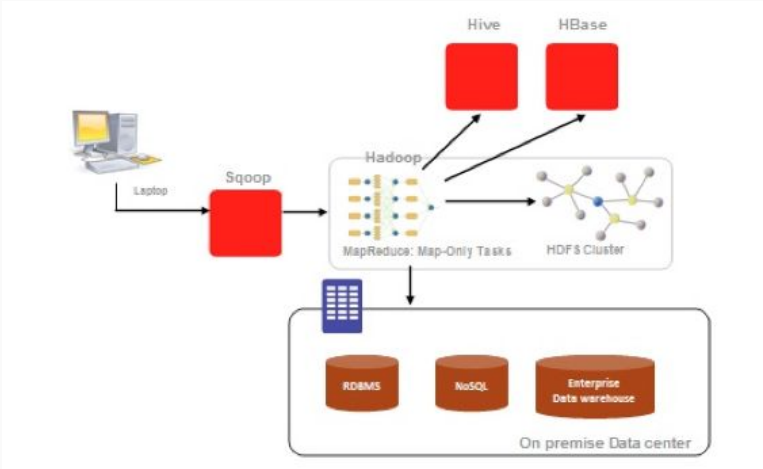
Es una herramienta distribuida para la recolección, agregación y transmisión de grandes volúmenes de datos de diferentes orígenes altamente configurable. Ofrece una arquitectura basada en la transmisión de datos por streaming altamente flexible y configurable pero a la vez simple de manera que se adapta 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.

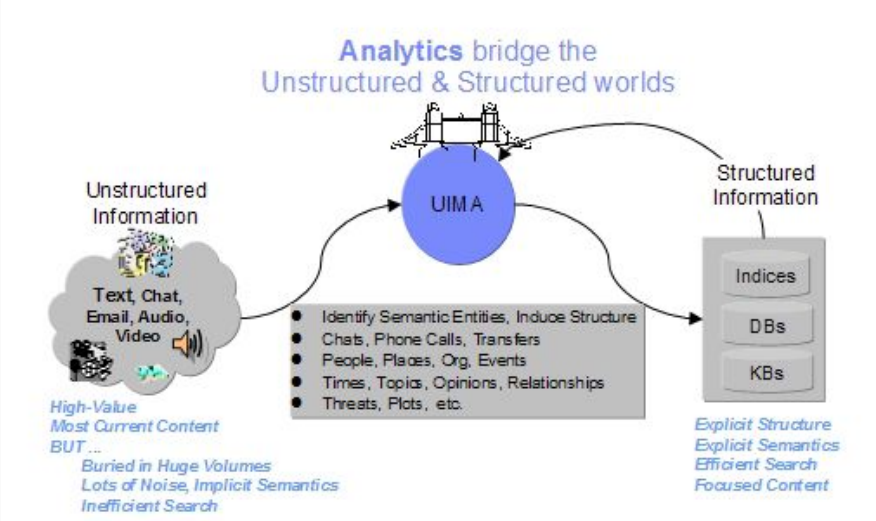


Captura y Manipulación De Datos

Unstructured Information Management Architecture
An Apache Project

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…



Captura y Manipulación De Datos

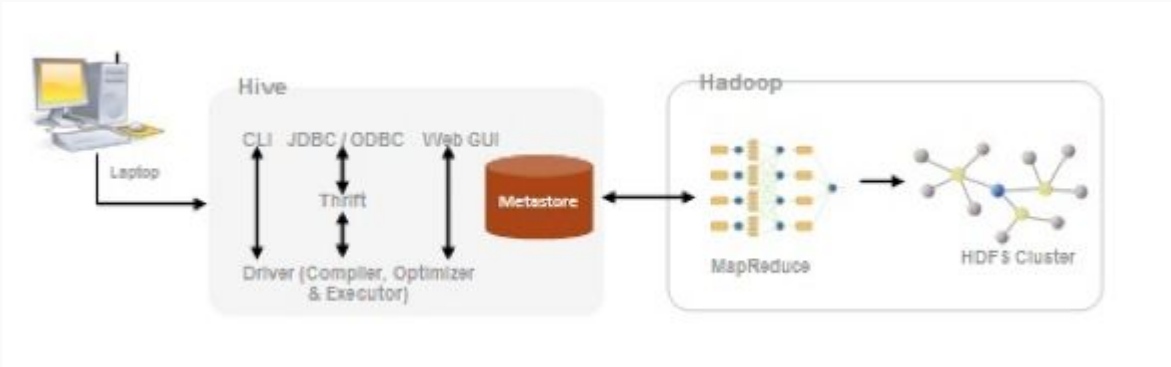
Lucene https://lucene.apache.org/core/

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/

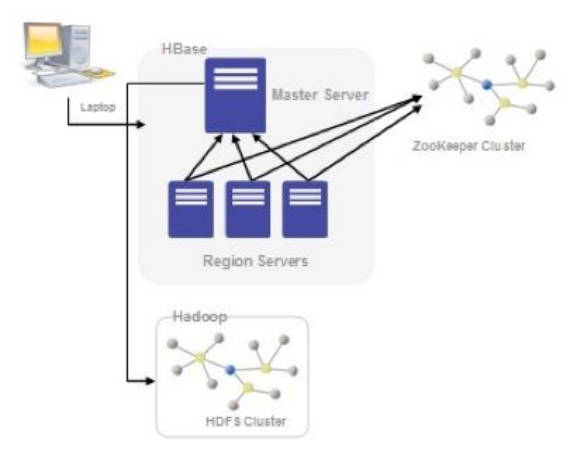
Es una herramienta data warehousing que facilita la creación, consulta y administración de grandes volúmenes de datos almacenados en Hadoop. Cuenta con su propio lenguaje derivado del SQL, conocido como Hive QL, el cual permite realizar las consultas sobre los datos utilizando MapReduce para poder paralelizar 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.

Tratamiento de datos

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.

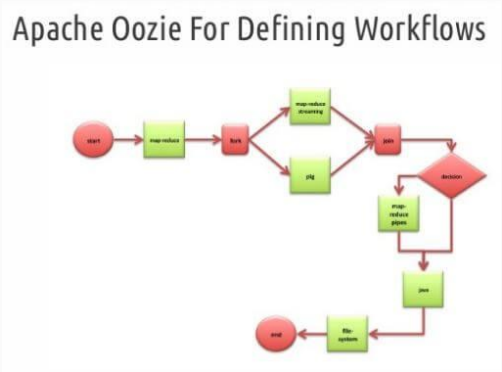
Debido a su fácil y potente uso es usado en procesos de ETL y en la manipulación y análisis de datos crudos.

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.

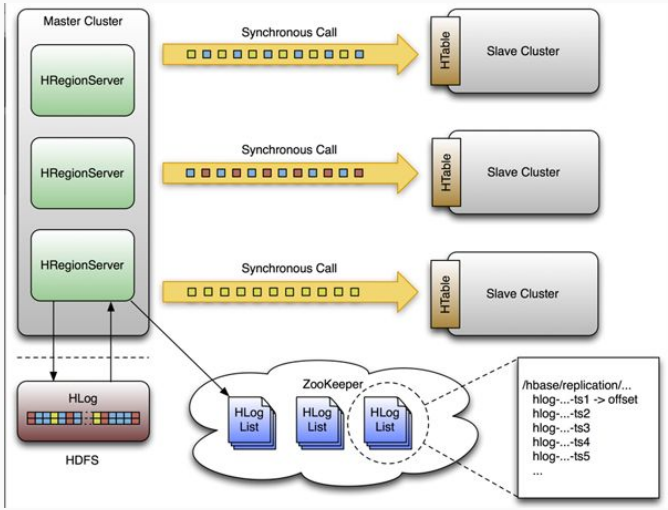
Oozie funciona como un motor de workflows a modo de servicio que permite lanzar, parar, suspender, retomar y volver a ejecutar una serie de trabajos Hadoop (tales como Java Map-Reduce, Streaming Map-Reduce, Pig, Hive, Sqooq…) basándose en ciertos criterios, como temporales o de disponibilidad de datos. 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.



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.



Administración

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 ayudar 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.

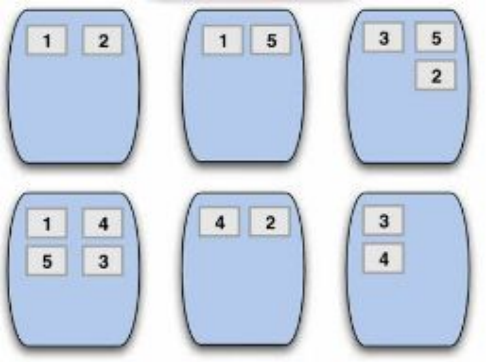
El Nodo maestro se llamará `NameNode` y mantendrá la información de qué Bloques hacen un archivo, además de donde están localizados. A esto se le conoce como `Metadata`.

HDFS :: ¿Cómo son almacenados los archivos?

`NameNode`: almacena solamente los Metadatos de los archivos

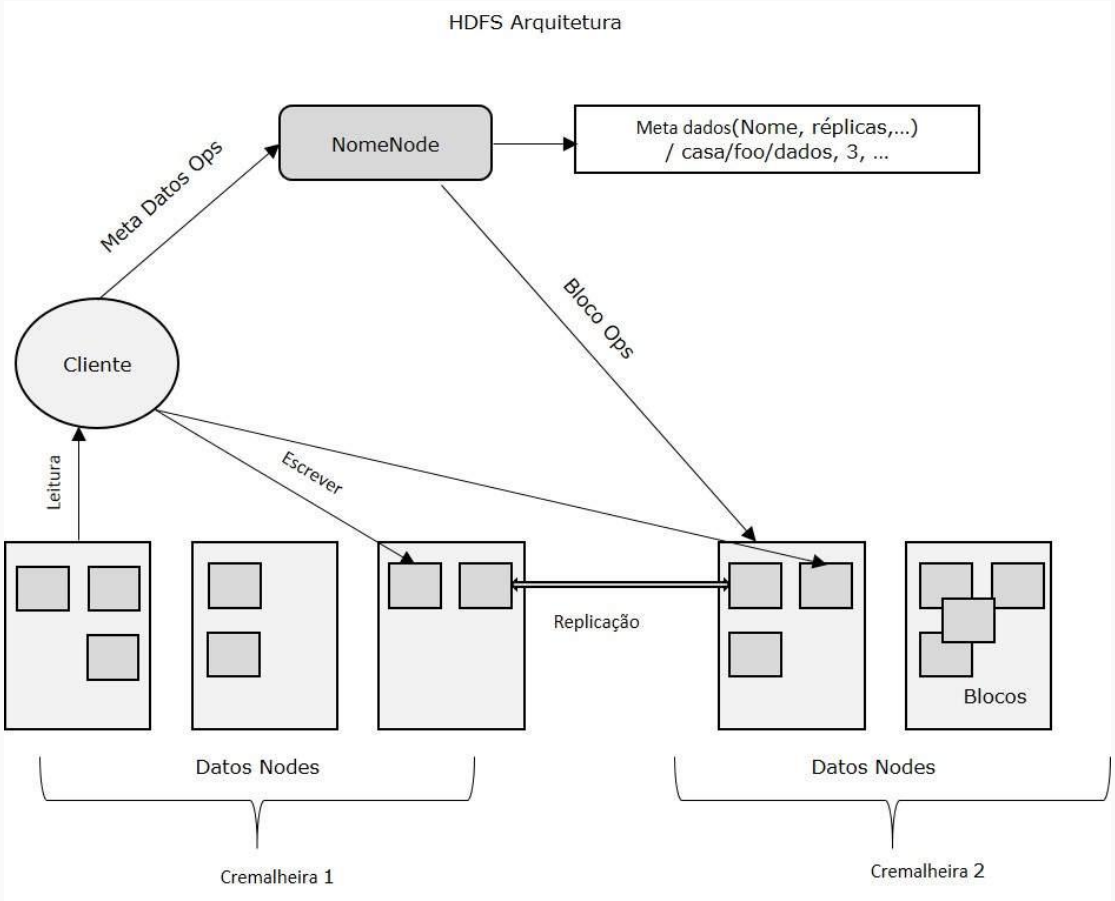
`DataNode`: almacena los Bloques actuales.

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 archive, 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.

HDFS :: Objetivos

- **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

https://hadoop.apache.org/docs/r2.8.2/hadoop-project-dist/hadoop-hdfs/HDFSCommands.html

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.

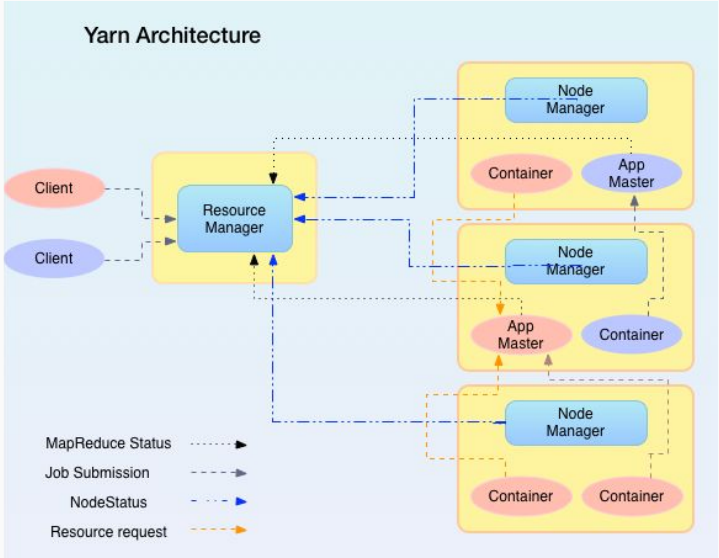
Introducción a YARN

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

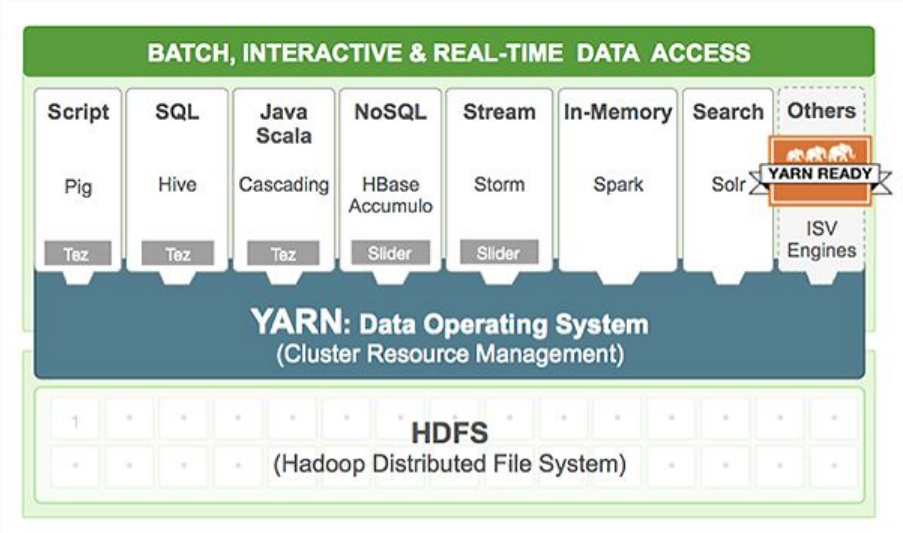
- `Container` es la unidad básica de la asignación en lugar de un Map o Reduce slot en Hadoop 1.x. El contenedor se define con atributos como la memoria, CPU, disco etc,… aplicaciones como procesamiento gráfico y MPI.
- `History Server` mantiene la historia de todos los Jobs

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

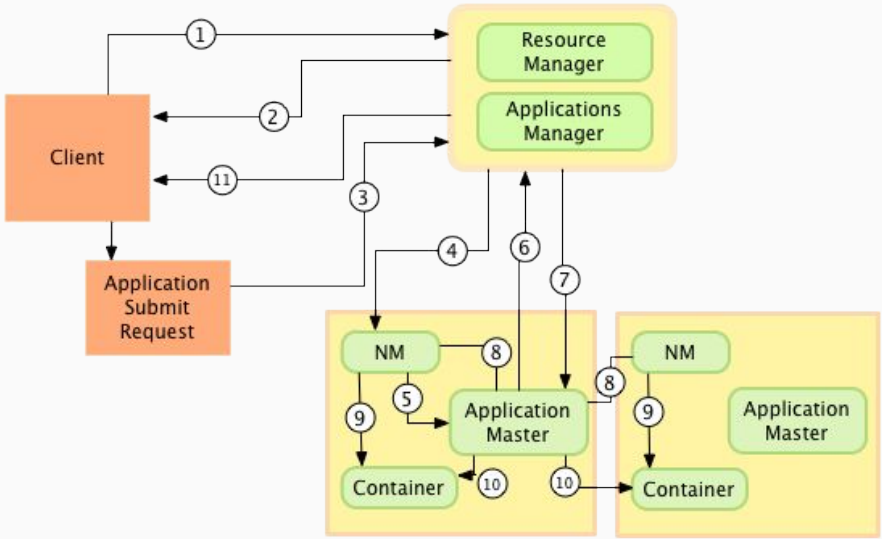
[1] Cliente comunica con el Resource Manager con una nueva Application Request

[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 - Applciation 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.

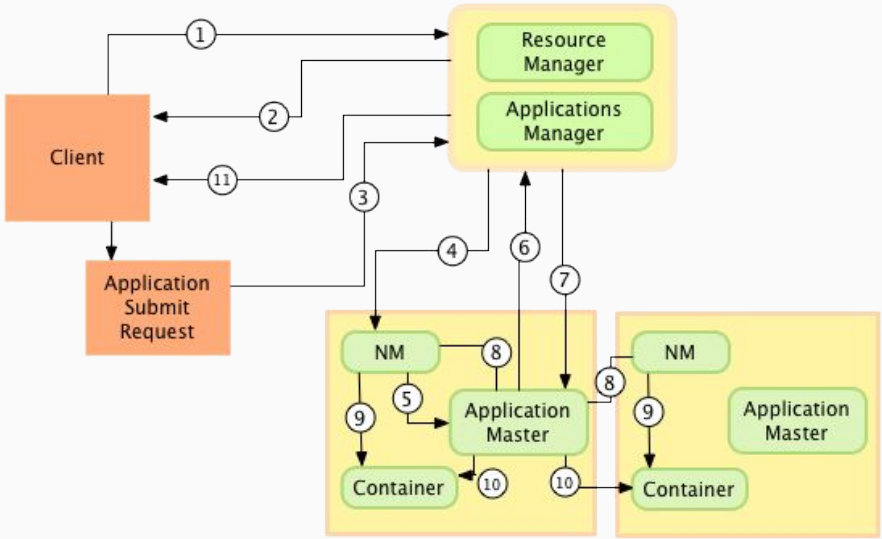
[7] El ResourceManager devuelve una lista de contenedores.

[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

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

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 *deamon 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> <!-- CUIDADO usar  0.0.0.0:8080 en caso probelmas -->
          <description>Nombre del filesystem por defecto.</description>
     </property>
</configuration>
```

Archivo hdfs-site.xml

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’)

Archivo hadoop-env.sh

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>
```

Archivo mapred-site.xml

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’)

Archivo yarn-site.xml

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.

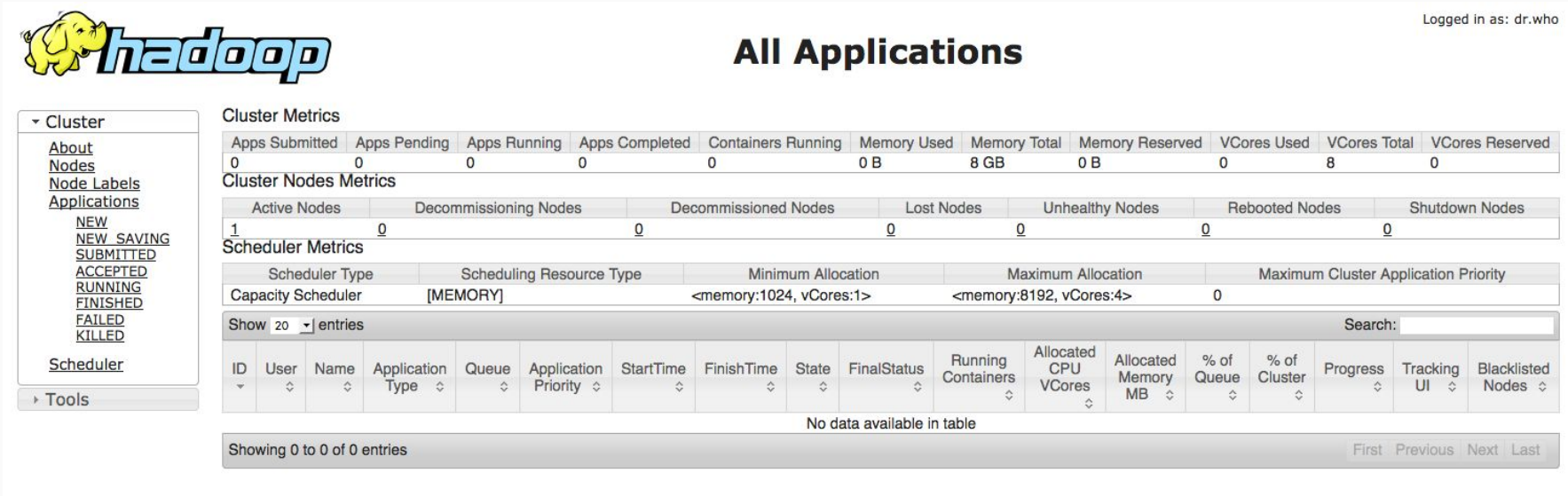
```
start-dfs.sh

start-yarn.sh
```

luego mediante jps podremos verificar si se encuentran en ejecución

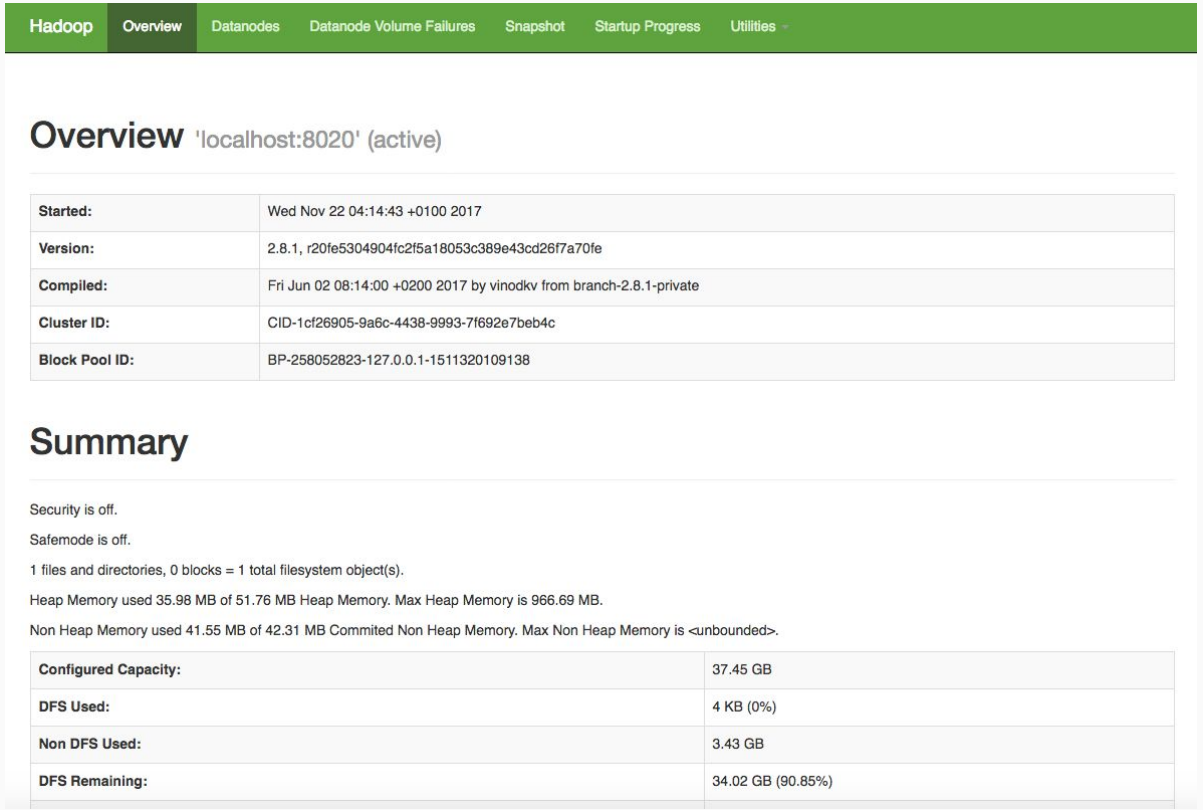
```
22642 Jps
22323 ResourceManager
22423 NodeManager
22014 DataNode
21887 NameNode
22175 SecondaryNameNode
```

Interfaces Web



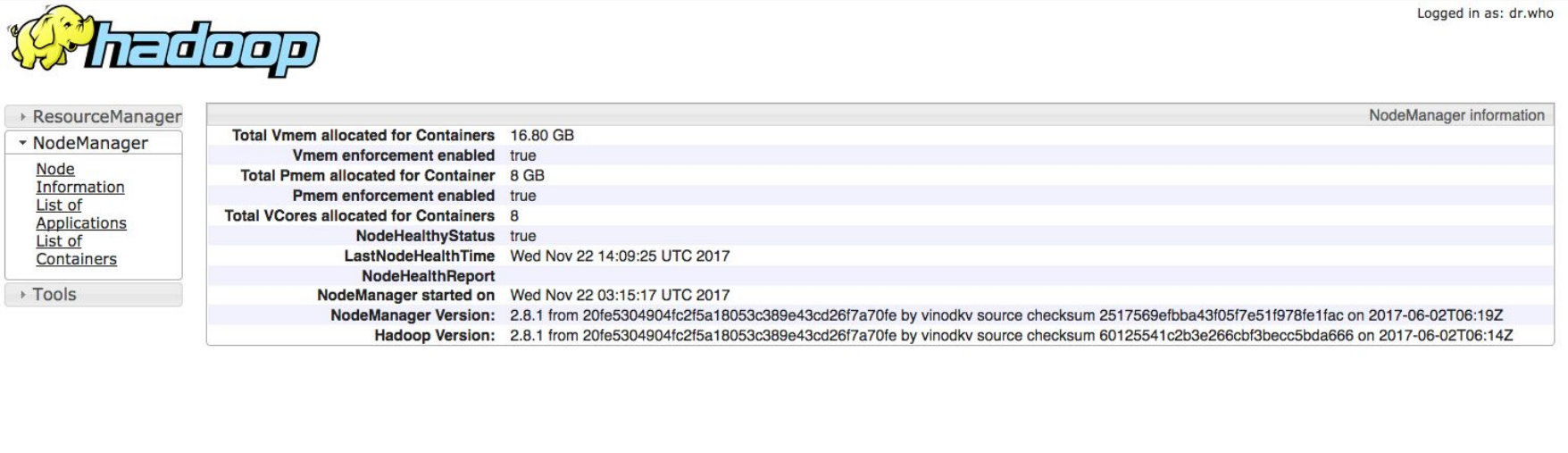
http://localhost:8042/

Interfaces Web



http://localhost:50070

Interfaces Web



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("Len:"+fileStatus.getLen());
System.out.println("ModificationTime:"+fileStatus.getModificationTime());
System.out.println("Path:"+fileStatus.getPath());
```

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

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

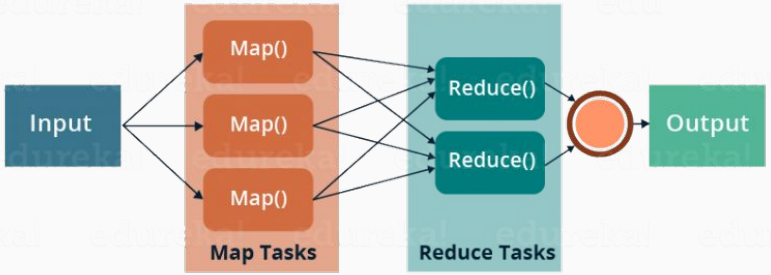
Aplicaciones MapReduce distribuidas e invertidas

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 profundirzán 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:

Shuffling y Sorting

En Hadoop, el proceso por el cual la salida intermedia de los mapeadores se transfiere al reductor se denomina Shuffling. Reducer obtiene 1 o más claves y valores asociados sobre la base de reductores. El valor clave intermedio generado por el asignador se clasifica automáticamente por clave.

Shuffling en MapReduce

El proceso de transferencia de datos desde los mapeadores a los reductores se conoce como reorganización, es decir, el proceso por el cual el sistema realiza el ordenamiento y transfiere la salida del mapa al reductor como entrada.

Por lo tanto, la fase de mezcla MapReduce es necesaria para los reductores, de lo contrario, no tendrían ninguna entrada (o entrada de cada asignador). Como la mezcla puede comenzar incluso antes de que la fase del mapa haya finalizado, ahorra tiempo y completa las tareas en menor tiempo.

Sorting en MapReduce

Las claves generadas por el asignador se ordenan automáticamente por MapReduce Framework, es decir, antes de iniciarse con reductor, todos los pares clave-valor intermedios en MapReduce generados por el mapeador se clasifican por clave y no por valor. Los valores pasados a cada reductor no están ordenados; pueden estar en cualquier orden.

La clasificación en Hadoop ayuda al Reducer a distinguir fácilmente cuándo debe comenzar una nueva tarea de reducción. Esto ahorra tiempo para el reductor. Reducer inicia una nueva tarea de reducción cuando la siguiente clave en los datos de entrada ordenados es diferente a la anterior. Cada tarea de reducción toma pares clave-valor como entrada y genera un par clave-valor como salida.

Sorting en MapReduce

Tenga en cuenta que la reorganización y ordenación en MapReduce no se realiza en absoluto si especifica cero reductores (setNumReduceTasks (0)). Luego, el trabajo MapReduce se detiene en la fase del mapa, y la fase del mapa no incluye ningún tipo de clasificación (por lo que incluso la fase del mapa es más rápida).

Secondary Sorting in MapReduce

Si queremos ordenar los valores del reductor, entonces se usa la técnica de clasificación secundaria ya que nos permite ordenar los valores (en orden ascendente o descendente) pasados a cada reductor.

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, ademas 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:

```
void setInputFormatClass(..);
void setOutputFormatClass(..);
void setMapperClass(..);
void setCombinerClass(..);
void setReducerClass(...);
void setPartitionerClass(..);
void setMapOutputKeyClass(..);
void setMapOutputValueClass(..);
void setOutputKeyClass(..);
void setOutputValueClass(..);
```

```
void setJobName(String name);
float mapProgress();
float reduceProgress();
boolean isSuccessful();
void killJob();
void submit();
boolean waitForCompletion(..);
```

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.

Clase Map

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 llamadopor 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`.

Counters

Los contadores de Hadoop proporcionan una forma de medir el progreso o la cantidad de operaciones que se producen dentro del trabajo de MapReduce.

Hadoop posee algunos contadores incorporados para cada trabajo y estos informan varias métricas, como, hay contadores para el número de bytes y registros, que nos permiten confirmar que la cantidad esperada de entrada se consume y se produce la cantidad esperada de salida.

Los contadores son un canal útil para reunir estadísticas sobre el trabajo de MapReduce:

- Para control de calidad o para nivel de aplicación.
- También son útiles para el diagnóstico de problemas.

Counters

Los contadores representan los contadores globales de Hadoop, definidos por el framework o las aplicaciones de MapReduce.

Cada contador es nombrado por un `Enum` y tiene un `long` para el valor. Los contadores se agrupan en grupos, cada uno compuesto de contadores de una clase particular de Enum.

Los contadores validan que:

- Se leyó y se escribió la cantidad correcta de bytes.
- Se lanzó la cantidad correcta de tareas y se ejecutó correctamente.
- La cantidad de CPU y memoria consumida es apropiada para nuestro trabajo y nodos del clúster.

Counters :: Tipos

Básicamente hay 2 tipos de contadores MapReduce:

- Contadores incorporados en MapReduce.
- Contadores definidos por el usuario / Contadores personalizados en MapReduce.

Los contadores se dividen en grupos y hay varios grupos para los contadores integrados. Cada grupo contiene contadores de tareas (que se actualizan como progreso de la tarea) o contador de trabajos (que se actualizan como un progreso del trabajo).

Counters :: Integrados

Contador de Jobs MapReduce :

El contador de tareas recopila información específica (como el número de registros leídos y escritos) sobre tareas durante su tiempo de ejecución. Por ejemplo, el contador MAP_INPUT_RECORDS es el contador de tareas que cuenta los registros de entrada leídos por cada tarea de mapa.

Los contadores de tareas de Hadoop se mantienen con cada intento y se envían periódicamente al maestro de aplicaciones para que puedan agregarse globalmente.

Contadores FileOutputFormat : FileOutputFormat recopila información de una cantidad de bytes escritos por tareas de Map (para trabajos de solo mapa) o Reduce tareas a través de FileOutputFormat.

Counters :: Tipos

Contadores de FileSystem : Reúnen información como una cantidad de bytes leídos y escritos por el sistema de archivos. A continuación se encuentran el nombre y la descripción de los contadores del sistema de archivos:

- Número de bytes leídos por el sistema de archivos por mapa y reducir tareas.
- Cantidad de bytes escritos en el sistema de archivos por mapa y reducir tareas.

Contadores FileInputFormat en Hadoop: Los contadores FileInputFormat recopilan información de una cantidad de bytes leídos por tareas de mapa a través de FileInputFormat.

Contadores FileOutputFormat en MapReduce : FileOutputFormat recopila información de una cantidad de bytes escritos por tareas de mapa (para trabajos de solo mapa) o reduce tareas a través de FileOutputFormat.

Counters :: Tipos

Contadores de Jobs en MapReduce : El contador de Jobs mide las estadísticas de nivel de trabajo. Por ejemplo, TOTAL_LAUNCHED_MAPS, que cuenta el número de tareas de Map que se iniciaron durante el curso de un Job (incluidas las tareas que fallaron).

El maestro de aplicaciones mantiene los contadores de trabajo, por lo que no es necesario enviarlos a través de la red, a diferencia de todos los demás contadores, incluidos los definidos por el usuario.

Counters :: Tipos

Contadores dinámicos en Hadoop: Los campos de Java Enum se definen en tiempo de compilación, por lo que no podemos crear contadores nuevos en tiempo de ejecución mediante enumeraciones.

Para hacerlo, usamos contadores dinámicos, uno que no está definido en tiempo de compilación utilizando java enum.

Counters :: Conclusión

En conclusión, los contadores verifican si se lee o escribe el número correcto de bytes, si se inicia el número correcto de tareas y se ejecuta correctamente.

Por lo tanto, Hadoop mantiene contadores incorporados y contadores definidos por el usuario para medir el progreso que ocurre en el trabajo de MapReduce.

InputFormat en Mapreduce

El InputFormat define cómo se dividen y leen los archivos de entrada en Hadoop. El InputFormat es el primer componente en Map-Reduce, es responsable de crear las divisiones de entrada y dividirlas en registros.

Inicialmente, los datos para una tarea MapReduce se almacenan en archivos de entrada, y los archivos de entrada normalmente residen en HDFS. Aunque el formato de estos archivos es arbitrario, se pueden usar archivos de registro basados en líneas y formatos binarios. Usando InputFormat definimos cómo estos archivos de entrada se dividen y leen.

InputFormat :: Definición

La clase InputFormat es una de las clases fundamentales en el marco Hadoop MapReduce que proporciona la siguiente funcionalidad:

Los archivos u otros objetos que deberían usarse para la entrada son seleccionados por InputFormat.

InputFormat define las divisiones de Datos, que define tanto el tamaño de las tareas de Mapa individuales como su posible servidor de ejecución.

InputFormat define el RecordReader, que es responsable de leer los registros reales de los archivos de entrada.

InputFormat :: Tipos

FileInputFormat:
es la clase base para todos los InputFormats basados en archivos. Hadoop FileInputFormat especifica el directorio de entrada donde se encuentran los archivos de datos. Cuando comenzamos un trabajo de Hadoop, a FileInputFormat se le proporciona una ruta que contiene archivos para leer. FileInputFormat leerá todos los archivos y los dividirá en uno o más InputSplits.

TextInputFormat:
es el InputFormat predeterminado de MapReduce. TextInputFormat trata cada línea de cada archivo de entrada como un registro separado y no realiza ningún análisis sintáctico. Esto es útil para datos no formateados o registros basados en línea como archivos de registro.

Clave: es el desplazamiento del byte del comienzo de la línea dentro del archivo (no todo el archivo solo una división), por lo que será único si se combina con el nombre del archivo.

Valor: es el contenido de la línea, excluyendo los terminadores de línea.

InputFormat :: Tipos

KeyValueTextInputFormat:
es similar a TextInputFormat ya que también trata cada línea de entrada como un registro separado. Mientras que TextInputFormat trata a toda la línea como el valor, pero KeyValueTextInputFormat divide la línea en clave y valor por un carácter de tabulación ('/ t'). Aquí la clave es todo hasta el carácter de tabulación mientras que el valor es la parte restante de la línea después del carácter de tabulación.

SequenceFileInputFormat:
es un InputFormat que lee archivos de secuencia. Los archivos de secuencia son archivos binarios que almacenan secuencias de pares clave-valor binarios. Los archivos de secuencia están comprimidos en bloque y proporcionan serialización directa y deserialización de varios tipos de datos arbitrarios (no solo texto). Aquí Key & Value ambos están definidos por el usuario.

InputFormat :: Tipos

SequenceFileAsTextInputFormat:
es otra forma de SequenceFileInputFormat que convierte los valores de clave del archivo de secuencia a objetos de texto. Al llamar a 'tostring ()' la conversión se realiza en las claves y valores. Este InputFormat hace que los archivos de secuencia sean una entrada adecuada para la transmisión.

SequenceFileAsBinaryInputFormat:
es un SequenceFileInputFormat con el que podemos extraer las claves y los valores del archivo de secuencia como un objeto binario opaco.

NLineInputFormat:
es otra forma de TextInputFormat donde las claves son offset de bytes de la línea y los valores son contenidos de la línea. Cada asignador recibe un número variable de líneas de entrada con TextInputFormat y KeyValueTextInputFormat y el número depende del tamaño de la división y la longitud de las líneas. Y si queremos que nuestro asignador reciba un número fijo de líneas de entrada, entonces usamos NLineInputFormat.

InputFormat :: Tipos

NLineInputFormat (cont):
En donde N es la cantidad de líneas de entrada que recibe cada asignador. Por defecto (N = 1), cada asignador recibe exactamente una línea de entrada. Si N = 2, cada división contiene dos líneas. Un mapeador recibirá los dos primeros pares de clave-valor y otro mapeador recibirá los dos pares de clave-valor.

DBInputFormat:
Es un InputFormat que lee datos de una base de datos relacional, utilizando JDBC. Como no tiene capacidades de división en porciones, debemos tener cuidado de no saturar la base de datos desde la cual estamos leyendo demasiados mapeadores. Por lo tanto, es mejor para cargar conjuntos de datos relativamente pequeños, tal vez para unirse a grandes conjuntos de datos de HDFS utilizando MultipleInputs. Aquí Key es LongWritables mientras que Value es DBWritables

OutputFormat en Mapreduce

Hadoop `OutputFormat` comprueba la especificación de salida del trabajo. Determina cómo se utiliza la implementación de `RecordWriter` para escribir resultados en archivos de salida.

Entre ellas `TextOutputFormat`, `SequenceFileOutputFormat`, `MapFileOutputFormat`, `SequenceFileAsBinaryOutputFormat`, `DBOutputFormat`, `LazyOutputForma` y `MultipleOutputs`.

OutputFormat :: RecordWriter

Antes de comenzar con Hadoop `OutputFormat` en `MapReduce`, primero veamos qué es un `RecordWriter` en MapReduce y cuál es su función en `MapReduce`.

RecordWriter:
El Reducer toma como entrada un conjunto de un par clave-valor intermedio producido por el mapeador y ejecuta una función reductora sobre ellos para generar un resultado que es nuevamente cero o más pares clave-valor.

RecordWriter escribe estos pares clave-valor de salida desde la fase `Reducer` a los archivos de salida.

OutputFormat :: Definición

El OutputFormat determina la forma en que RecordWriter graba estos pares clave-valor de salida en los archivos de salida. Las funciones OutputFormat y InputFormat son parecidas. Las instancias de OutputFormat proporcionadas por Hadoop se utilizan para escribir en archivos en HDFS o en el disco local. OutputFormat describe la especificación de salida para un trabajo Map-Reduce. Sobre la base de la especificación de salida;

- El trabajo de MapReduce comprueba que el directorio de salida aún no existe.
- OutputFormat proporciona la implementación de RecordWriter que se utilizará para escribir los archivos de salida del trabajo. Los archivos de salida se almacenan en un FileSystem.

El método FileOutputFormat.setOutputPath () se usa para establecer el directorio de salida. Cada Reducer escribe un archivo separado en un directorio de salida común..

OutputFormat :: Tipos

TextOutputFormat :

`MapReduce` por defecto `OutputFormat` es `TextOutputFormat`, que escribe pares (clave, valor) en líneas individuales de archivos de texto y sus claves y valores pueden ser de cualquier tipo ya que `TextOutputFormat` los convierte en cadenas llamando a `toString()` en ellos. Cada par clave-valor está separado por un carácter de tabulación, que se puede cambiar utilizando la propiedad *MapReduce.output.textoutputformat.separator.KeyValueTextOutputFormat* se utiliza para leer estos archivos de texto de salida, ya que divide las líneas en pares clave-valor basados en un separador configurable.

Tipos de OutputFormat

SequenceFileOutputFormat:
Es un `OutputFormat` que escribe los archivos de secuencias para su salida y es un uso de formato intermedio entre los trabajos de MapReduce, que serializan rápidamente los tipos de datos arbitrarios al archivo; y el `SequenceFileInputFormat` correspondiente deserializará el archivo en los mismos tipos y presentará los datos al siguiente mapeador de la misma manera en que fue emitido por el reductor anterior, ya que estos son compactos y fácilmente comprimibles. La compresión está controlada por los métodos estáticos en `SequenceFileOutputFormat`.

SequenceFileAsBinaryOutputFormat:
Es otra forma de SequenceFileInputFormat que escribe claves y valores para secuenciar el archivo en formato binario.

Tipos de OutputFormat

MapFileOutputFormat :
Es otra forma de FileOutputFormat en Hadoop, que se utiliza para escribir resultados como archivos de mapas. La clave en un MapFile debe agregarse en orden, por lo que debemos asegurarnos de que el reductor emita las claves en orden ordenado.

MultipleOutputs :
Permite escribir datos en archivos cuyos nombres se derivan de las claves y valores de salida o, de hecho, de una cadena arbitraria.

Tipos de OutputFormat

LazyOutputFormat:
A veces, FileOutputFormat creará archivos de salida, incluso si están vacíos. LazyOutputFormat es un wrapper OutputFormat que asegura que el archivo de salida se creará solo cuando el registro se emita para una partición determinada.

DBOutputFormat :
DBOutputFormat en Hadoop es un OutputFormat para escribir en bases de datos relacionales y HBase. Envía la salida de reducción a una tabla SQL. Acepta pares clave-valor, donde la clave tiene un tipo que se extiende DBwritable. Returned RecordWriter solo escribe la clave en la base de datos con una consulta SQL por lotes.

Qué es Partitioner personalizado

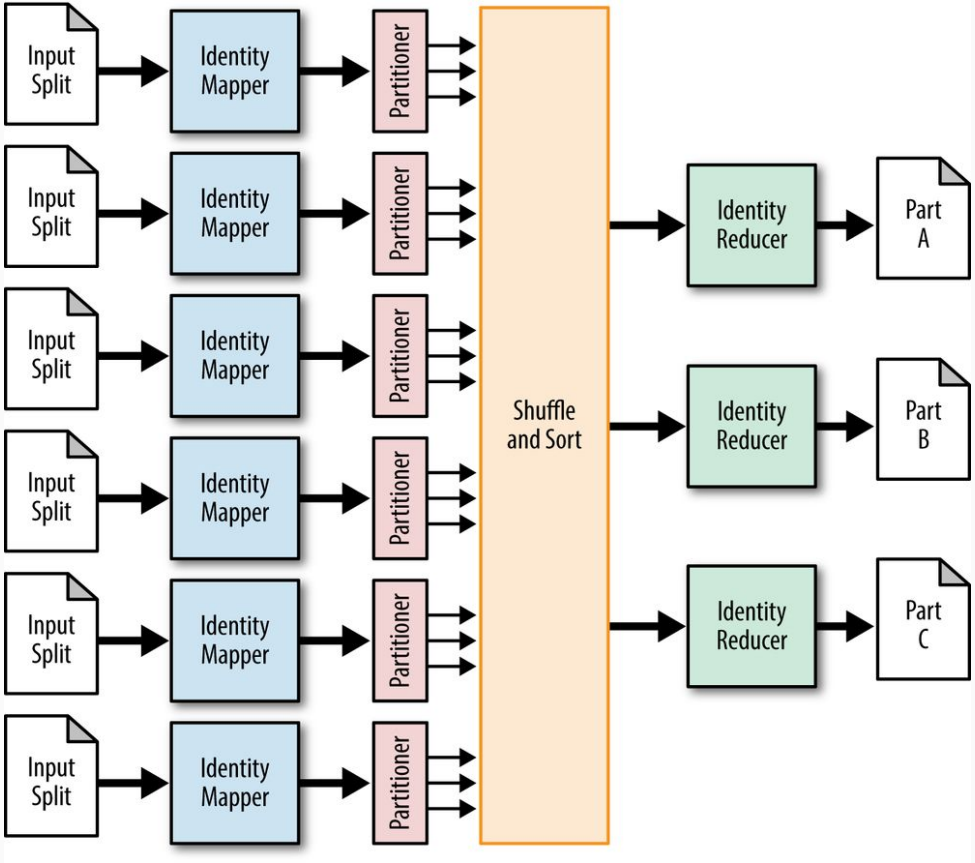
Partitioner personalizado es un proceso que le permite almacenar los resultados en diferentes Reducers, según la condición del usuario.

Al configurar un partitioner para que particione mediante la clave, podemos garantizar que los registros de la misma clave irán al mismo Reducer.

Un partitioner asegura que solo un reductor reciba todos los registros para esa clave en particular.

Partitioner

Dentro del flujo de ejecución podemos interferir el mismo dando una optimización de los mismos.



Hadoop Default Partitioner

HashPartitioner es el Partitioner predeterminado en Hadoop, que crea una tarea de reducción para cada "clave" única. Todos los valores con la misma clave van a la misma instancia de su reductor, en una sola llamada a la función de reducción.

Si el usuario está interesado en almacenar un grupo particular de resultados en diferentes reductores, entonces el usuario puede escribir su propia implementación del particionador. Puede ser de uso general o personalizado para los tipos de datos o valores específicos que espera usar en la aplicación del usuario.

Cantidad de Partitioners

El número total de `Partitioners` que se ejecutan en `Hadoop` es igual al número de reductores, es decir, el `Partitioner` dividirá los datos de acuerdo con el número de reductores establecido por el método `JobConf.setNumReduceTasks()`.

Por lo tanto, los datos del único particionador se procesan con un solo `Reducers`. Y el particionador se crea solo cuando hay múltiples reductores.

Multiples outputs

Hadoop admite una clase llamada `MultipleOutputs`, que nos permite escribir desde un programa de Reduce a múltiples archivos. Podemos escribir salida en diferentes archivos, tipos de archivos y diferentes ubicaciones con él. También puede elegir el nombre del archivo con esta API. Para usar esto, vamos a echar un vistazo a un programa de conteo de palabras simple y escribir de este programa a múltiples archivos de salida .

En el método `setup`, necesitamos iniciar el objeto de la clase `MultipleOutputs`. En el método de reducción, tenemos que agregar el método de escritura, que escribiría los valores clave deseados en dos archivos, uno del tipo de texto y otro del tipo de archivo de secuencia.

Multiples outputs

Además, también puede especificar la ruta completa aparte de la ruta de salida predeterminada en el código del reductor donde desea ver su archivo:

```
mos.write (clave, this.result, "/ my_new_output_path_up / file");
```

Es muy importante cerrar la secuencia `MultipleOutputs` en el método de cleanup.

`MulitpleOutputs` es de gran ayuda en dos condiciones: cuando desea ver múltiples archivos de salida en diferentes formatos de archivo y cuando desea escribir la salida de su mapa, reduzca el programa para usar los archivos especificados.

`MultipleOutputs` admite contadores, pero están deshabilitados de forma predeterminada. Una vez especificado, `MultipleOutputs` primero creará un archivo con un nombre específico, y luego comenzará a escribir cualquier dato si está disponible. Si no se generan datos, seguirá creando archivos de tamaño cero. Para evitar esto, podemos usar `LazyOutputFormat`.

Que es un Combiner

En un gran conjunto de datos cuando ejecutamos el trabajo MapReduce, el Mapper genera grandes cantidades de datos intermedios y estos datos intermedios se transfieren al Reducer para su posterior procesamiento, lo que genera una enorme congestión de red. MapReduce framework proporciona una función conocida como Combiner que desempeña un papel clave en la reducción de la congestión de la red.

El combinador en MapReduce también se conoce como 'Mini-reductor'. El trabajo principal de Combiner es procesar los datos de salida del asignador, antes de pasarlo a Reducer. Se ejecuta después del asignador y antes del Reducer y su uso es opcional.

Combiner juega un papel clave en la reducción de la congestión de la red. Mejora el rendimiento general del reductor al resumir la salida de Mapper.

Ventajas y desventajas del Combiner en MapReduce

Como hemos discutido en detalle sobre Hadoop MapReduce Combiner, ahora discutiremos algunas de las ventajas de MapReduce Combiner.

- Combiner reduce el tiempo de transferencia de datos entre mapper y reductor.
- Disminuye la cantidad de datos que el reductor debe procesar.
- El combinador mejora el rendimiento general del reductor.

También hay algunas desventajas de Hadoop Combiner.

- Los trabajos de MapReduce no pueden depender de la ejecución del combinador Hadoop porque no hay garantía en su ejecución.
- En el sistema de archivos local, los pares clave-valor se almacenan en Hadoop y ejecutan el combinador más tarde, lo que causará costosas IO de disco.

OutputFormat en Mapreduce

Hadoop OutputFormat comprueba la especificación de salida del trabajo. La misma determina cómo se utiliza la implementación de RecordWriter para escribir resultados en archivos de salida.

Entre los OutputFormat en Hadoop como `TextOutputFormat`, `SequenceFileOutputFormat`, `MapFileOutputFormat`, `DBOutputFormat`, `LazyOutputForma`, `SequenceFileAsBinaryOutputFormaty` `MultipleOutputs`.

Tipos OutputFormat

TextOutputFormat: Es el OutputFormat por defecto, que escribe pares (clave, valor) en líneas individuales de archivos de texto y sus claves y valores pueden ser de cualquier tipo ya que TextOutputFormat los convierte en cadenas llamando a `toString()` en ellos. Cada par clave-valor está separado por un carácter de tabulación, que se puede cambiar utilizando la propiedad `MapReduce.output.textoutputformat.separator.KeyValueTextOutputFormat` se utiliza para leer estos archivos de texto de salida, ya que divide las líneas en pares clave-valor basados en un separador configurable.

SequenceFileOutputFormat: es un OutputFormat que escribe los archivos de secuencias para su salida y es un formato de uso intermedio entre los trabajos de MapReduce, que serializan rápidamente los tipos de datos arbitrarios al archivo; y el `SequenceFileInputFormat` correspondiente deserializará el archivo en los mismos tipos y presentará los datos al siguiente mapeador de la misma manera en que fue emitido por el reductor anterior, ya que estos son compactos y fácilmente comprimibles. La compresión está controlada por los métodos estáticos en `SequenceFileOutputFormat`.

Tipos OutputFormat

SequenceFileAsBinaryOutputFormat: es otra forma de SequenceFileInputFormat que escribe claves y valores para secuenciar el archivo en formato binario.

MapFileOutputFormat: es otra forma de FileOutputFormat en Hadoop, que se utiliza para escribir resultados como archivos de mapas. La clave en un MapFile debe agregarse en orden, por lo que debemos asegurarnos de que el reductor emita las claves en orden.

MultipleOutputs: Permite escribir datos en archivos cuyos nombres se derivan de las claves y valores de salida o, de hecho, de una cadena arbitraria.

Tipos OutputFormat

LazyOutputFormat: a veces FileOutputFormat creará archivos de salida, incluso si están vacíos. LazyOutputFormat es un wrapper OutputFormat que asegura que el archivo de salida se creará solo cuando el registro se emita para una partición determinada.

DBOutputFormat: OutputFormat para escribir en bases de datos relacionales y HBase. Envía la salida de reducción a una tabla SQL. Acepta pares clave-valor, donde la clave tiene un tipo que se extiende DBwritable. Returned RecordWriter solo escribe la clave en la base de datos con una consulta SQL por lotes..

RawComparator

RawComparator

El objetivo del `RawComparator`, es minimizar el coste de ordenación entre las fases de `Map` y de `Reduce`.

Hadoop MR deserializa las claves de salida del mapa para ordenarlas entre el mapa y reducir las fases `deserialize(keyBytes1).compare(deserialize(keyBytes2))`

Si no usamos `RawComparator`, las claves intermedias tendrían que deserializarse por completo para realizar una comparación pudiendo utilizar `compare(keyBytes1, keyBytes2)`

RawComparator

La implementación de la interface `org.apache.hadoop.io.RawComparator` nos permitirá ayudar a acelerar los trabajos de `Map / Reduce` (MR).

- (K1,V1) –> Map –> (K2,V2)
- (K2,List[V2]) –> Reduce –> (K3,V3)

Los pares clave-valor (K2, V2) se denominan pares intermedios. Se pasan del `Map` al `Reduce`. Antes de que estos pares de valores clave intermedios lleguen al reductor, se realiza un paso de mezcla y ordenación.

La reorganización es la asignación de las claves intermedias (K2) a los `reducers` y el orden es la clasificación de estas claves. Al implementar `RawComparator` para comparar las claves intermedias, este esfuerzo extra mejorará en gran medida la clasificación. La ordenación se mejora porque `RawComparator` comparará las claves por byte.

RawComparator

Desafortunadamente esto requiere la escritura de un comparador custom.

Y se asume que los datos son sencillos de comparar de manera serializada.

```
public interface RawComparator<T> {
     public int compare(byte[] b1, int s1, int l1, byte[] b2, int s2, int l2);
}
```

Joins

¿Qué es una Join?

Este tipo de unión se usa para combinar dos o más tablas de base de datos basadas en claves externas.

En general, los datos se mantienen tablas separadas y, muchas veces se necesitan generar informes analíticos usando los datos presentes en estas tablas separadas.

Por lo tanto, medin unión en estas tablas separadas utilizando una columna común (clave externa), como id de cliente, etc., para generar una tabla combinada. Luego, analizan esta tabla combinada para obtener los informes analíticos deseados.

Tipos de join

Al igual que SQL join, también podemos realizar operaciones de combinación en MapReduce en diferentes conjuntos de datos. Hay dos tipos de operaciones de unión en MapReduce:

`Map Side Join`: como su nombre lo indica, la operación de unión se realiza en la fase del mapa. Por lo tanto, el asignador realiza la unión y es obligatorio que la entrada a cada mapa se particione y clasifique según las claves.

`Reduce Side Join`: como su nombre lo indica, en la unión lateral reducida, el reductor es responsable de realizar la operación de unión. Es comparativamente más simple y fácil de implementar que la unión lateral del mapa ya que la fase de ordenación y mezcla envía los valores que tienen claves idénticas al mismo reductor y, por lo tanto, de manera predeterminada, los datos están organizados para nosotros.

Join Map-side

¿Qué es una Map Side Join?

MapReduce procesa los conjuntos de datos grandes, y el procesamiento de grandes conjuntos de datos requiere la mayoría de las veces unir conjuntos de datos basados en claves comunes, como casi siempre hacemos al jugar con cualquier base de datos RDBMS basada en el concepto de clave primaria / extranjera.

Cuando?

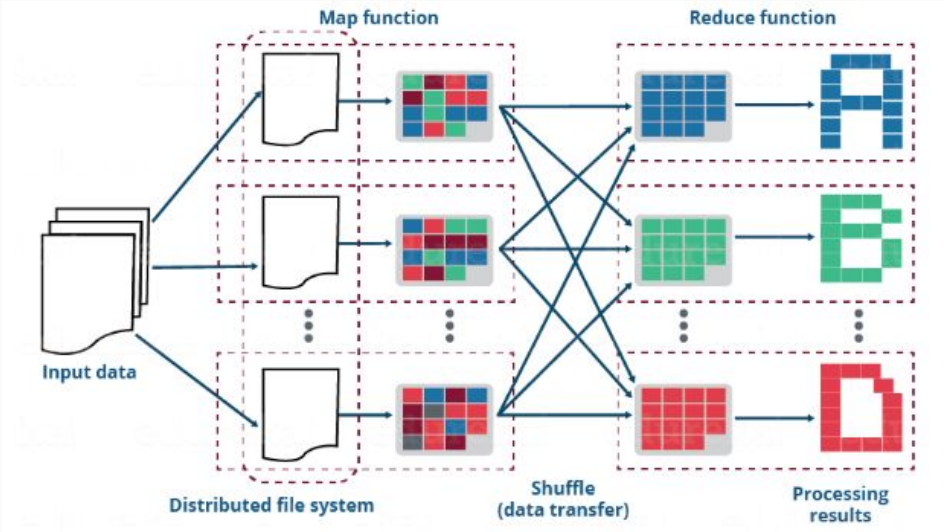
Puede usar la Map Side Join utilizando dos formas diferentes en función de sus conjuntos de datos, y eso depende de las condiciones siguientes:

- Ambos conjuntos de datos se deben dividir en el mismo número de particiones, y ya deben estar ordenados por la misma clave.
- De los dos conjuntos de datos uno debe ser pequeño (algo así como maestro) y puede caber en la memoria de cada nodo. (cache)

Reduce-Side Join

¿Qué es una Reduce-Side Join?

Las `joins` de conjuntos de datos realizados en la fase de reducción se denominan `reduce-side join`. Las mismas son más fácil de implementar ya que son menos estrictas que las uniones del lado del mapa que requieren que los datos se clasifiquen y particionen de la misma manera. Son menos eficientes que las combinaciones de mapas porque los conjuntos de datos tienen que atravesar la fase de `sort` y `shuffle`.



¿Qué es Reduce Side Join?

Como se hemos visto, Reduce Side Join es un proceso donde la operación de unión se realiza en la fase de reducción.

- Mapper lee los datos de entrada que se combinarán según la columna común o la clave de combinación.
- El asignador procesa la entrada y agrega una etiqueta a la entrada para distinguir la entrada que pertenece de diferentes fuentes o conjuntos de datos o bases de datos.
- El asignador genera el par clave-valor intermedio donde la clave no es más que la clave de unión.
- Después de la fase de sorting y shuffling, se genera una clave y la lista de valores para el reducer.
- Ahora, el reducer se une a los valores presentes en la lista con la clave para dar la salida agregada final.

¿Qué es Reduce Side Join?

El procedimiento de reducción de unión lateral genera un gran tráfico de E / S de red en la fase de clasificación y reducción donde los valores de la misma clave se juntan. Por lo tanto, si tiene una gran cantidad de conjuntos de datos diferentes que tienen millones de valores, existe una alta probabilidad de que se encuentre con una excepción OutOfMemory, es decir, su RAM está llena y, por lo tanto, sobreexcitada. En mi opinión, las ventajas de usar `Reduce-Side Join` son:

- Es muy fácil de implementar, ya que estamos aprovechando el algoritmo de clasificación y mezcla incorporado en el marco MapReduce que combina valores de la misma clave y la envía al mismo reductor.
- En la combinación lateral de reducción, su entrada no requiere seguir ningún formato estricto y, por lo tanto, también puede realizar la operación de unión en datos no estructurados.

En general, las personas prefieren Apache Hive, que es parte del ecosistema de Hadoop, para realizar la operación de unión.

Prueba unitaria de un job MapReduce

MRUnit

Generalmente, las programaciones distribuidas como los programas MapReduce son muy difíciles de depurar después de completar todo el código, por lo que ese tipo de programas es mejor probarlo en etapas anteriores. Estos tests son muy útiles para encontrar los errores y corregir los errores en las etapas iniciales de la programación.

Para ello, tenemos MRUnit como herramienta para testing para validar nuestro MapReduce de Hadoop.

¿Como se configura?

Al igual que cualquier test del tipo JUnit debemos crear nuestro suite y los diferentes cases. Para ello MRUnit nos dará sus propios drivers para la preparación de los casos:

```
MapDriver mapDriver = MapDriver.newMapDriver(mapper);

ReduceDriver reduceDriver = ReduceDriver.newReduceDriver(reducer);

MapReduceDriver mapReduceDriver = MapReduceDriver.newMapReduceDriver(mapper, reducer);
```

para luego setear la entrada y asignar la salida para la correspondiente validación:

```
mapReduceDriver.withInput(new LongWritable(), new Text("hadoop is bigdata"));

mapReduceDriver.withInput(new LongWritable(), new Text("hadoop is emerging"));

mapReduceDriver.withOutput(new Text("bigdata"), new LongWritable(1));

mapReduceDriver.withOutput(new Text("emerging"), new LongWritable(1));

mapReduceDriver.withOutput(new Text("hadoop"), new LongWritable(2));

mapReduceDriver.withOutput(new Text("is"), new LongWritable(2));
```

Compresión de datos

Para que?

Los enormes volúmenes de datos en una implementación típica de Hadoop hacen que la compresión sea una necesidad. La compresión de datos definitivamente le ahorra una gran cantidad de espacio de almacenamiento y seguramente acelerará el movimiento de esos datos a través de su clúster.

No es de extrañar que haya varios esquemas de compresión disponibles, llamados códecs, que pueda considerar.

Que podemos comprimir (cont’)

Comprimir archivos de entrada

Si el archivo de entrada está comprimido, entonces los bytes leídos desde HDFS se reducen, lo que significa menos tiempo para leer datos. Esta vez la conservación es beneficiosa para el desempeño de la ejecución del trabajo.

Si los archivos de entrada están comprimidos, se descomprimirán automáticamente a medida que MapReduce los lea, utilizando la extensión de nombre de archivo para determinar qué códec usar. Por ejemplo, un archivo que termina en .gz puede identificarse como un archivo comprimido gzip y, por lo tanto, leerse con GzipCodec.

Que podemos comprimir (cont’)

Comprimir archivos de salida

A menudo necesitamos almacenar el resultado como archivos de historial. Si la cantidad de salida por día es extensa, y a menudo necesitamos almacenar los resultados del historial para uso futuro, estos resultados acumulados requerirán una gran cantidad de espacio HDFS. Sin embargo, estos archivos de historial no se pueden usar con mucha frecuencia, lo que genera un desperdicio de espacio HDFS. Por lo tanto, es necesario comprimir la salida antes de almacenarla en HDFS.

Que podemos comprimir (cont’)

Salida del mapa de compresión

Incluso si su aplicación MapReduce lee y escribe datos sin comprimir, puede beneficiarse de la compresión de la salida intermedia de la fase del mapa. Dado que la salida del mapa se escribe en el disco y se transfiere a través de la red a los nodos reductores, usando un compresor rápido como LZO o Snappy, puede obtener ganancias de rendimiento simplemente porque se reduce el volumen de datos a transferir.

Tipos de compresores (I)

gzip:

Es naturalmente compatible con Hadoop. gzip se basa en el algoritmo DEFLATE, que es una combinación de LZ77 y Huffman Coding.

bzip2:

Es un compresor de datos de alta calidad, libremente disponible, libre de patentes. Normalmente comprime los archivos hasta dentro del 10% al 15% de las mejores técnicas disponibles (la familia de compresores estadísticos PPM), mientras que es aproximadamente el doble de rápido en la compresión y seis veces más rápido en la descompresión.

Tipos de compresores (II)

LZO:

El formato de compresión LZO se compone de muchos bloques más pequeños (~ 256K) de datos comprimidos, lo que permite dividir los trabajos a lo largo de los límites del bloque. Además, fue diseñado pensando en la velocidad: se descomprime aproximadamente dos veces más rápido que gzip, lo que significa que es lo suficientemente rápido como para mantenerse al día con las velocidades de lectura del disco duro. No se comprime tan bien como gzip: se esperan archivos del orden de un 50% más grandes que su versión comprimida. Pero eso es todavía 20-50% del tamaño de los archivos sin ninguna compresión en absoluto, lo que significa que los trabajos vinculados a IO completan la fase del mapa aproximadamente cuatro veces más rápido.

Problemas

Entre los problemas sobre la compresión y el split de entrada se debe considerar cómo comprimir los datos que MapReduce procesará, y para eso importante comprender si el formato de compresión admite la división. Considere un archivo descomprimido almacenado en HDFS cuyo tamaño es de 1 GB. Con un tamaño de bloque HDFS de 64 MB, el archivo se almacenará como 16 bloques, y un trabajo de MapReduce utilizando este archivo como entrada creará 16 divisiones de entrada, cada una procesada de manera independiente como entrada a una tarea de mapa separada.

Problemas (cont’)

Imagine ahora que el archivo es un archivo comprimido gzip cuyo tamaño comprimido es de 1GB. Como antes, HDFS almacenará el archivo como 16 bloques. Sin embargo, crear una división para cada bloque no funcionará, ya que es imposible comenzar a leer en un punto arbitrario en la secuencia gzip y, por lo tanto, es imposible que una tarea de mapa lea su división independientemente de las demás. El formato gzip usa DEFLATE para almacenar los datos comprimidos, y DEFLATE almacena los datos como una serie de bloques comprimidos. El problema es que el inicio de cada bloque no se distingue de ninguna manera que permita que un lector posicionado en un punto arbitrario en la secuencia avance al principio del siguiente bloque, sincronizándose así mismo con la secuencia. Por esta razón, gzip no admite división.

Problemas (cont’)

En este caso, MapReduce hará lo correcto y no intentará dividir el archivo comprimido, ya que sabe que la entrada está comprimida gzip y que este no admite la división. Esto funcionará, pero a expensas de la localidad: un solo mapa procesará los 16 bloques HDFS, la mayoría de los cuales no serán locales para el mapa. Además, con menos mapas, el trabajo es menos granular, por lo que puede llevar más tiempo ejecutarlo.

Si el archivo en nuestro ejemplo hipotético fuera un archivo LZO, tendríamos el mismo problema ya que el formato de compresión subyacente no proporciona una manera para que un lector se sincronice con la transmisión. Sin embargo, es posible preprocesar archivos LZO utilizando una herramienta indexadora que viene con las bibliotecas Hadoop LZO. La herramienta crea un índice de puntos de división, que los hace divisibles cuando se utiliza el formato de entrada MapReduce apropiado.

Un archivo bzip2, por otro lado, proporciona un marcador de sincronización entre bloques (una aproximación de 48 bits de pi), por lo que admite la división.

Resumen

Razones para comprimir:

a) Los datos se almacenan principalmente y no se procesan con frecuencia. Es el escenario habitual de DWH. En este caso, el ahorro de espacio puede ser mucho más significativo que el procesamiento general
b) El factor de compresión es muy alto y de eso ahorramos mucho IO.
c) La descompresión es muy rápida (como Snappy) y, por lo tanto, tenemos un poco de ganancia con poco precio
d) Los datos ya llegaron comprimidos

Resumen (cont’)

Razones para no comprimir:

a) Los datos comprimidos no son divisibles. Debe tenerse en cuenta que muchos formatos modernos están construidos con compresión de nivel de bloque para permitir la división y otro procesamiento parcial de los archivos.

b) Los datos se crean en el clúster y la compresión lleva un tiempo considerable. Hay que señalar que la compresión generalmente requiere mucha más CPU que la descompresión.

c) Los datos tienen poca redundancia y la compresión proporciona poca ganancia.