Hadoop

HDFS Hadoop Distributed File System

Introducción

Conceptos generales

introducción

- Problema: Conjunto de datos que crece por encima de la capacidad de almacenamiento de una máquina física
- Solución: Sistema de archivos distribuido
- Aumenta complejidad por ser un sistema en red
- Reto: evitar perdida de información por fallo de nodos

Intro

- HDFS está diseñado para almacenar archivos muy grandes con patrones de acceso a datos en streaming

Intro

- Archivos muy grandes →Cientos de MB o GB
- Datos en streaming → patrón 1 escritura , varias lecturas
- Hardware Comun → económico, genérico, con posibilidad de errores

Intro

- Entorno desaconsejados para HDFS
  - Tiempos de latencia bajos <10 ms
  - Muchos archivos pequeños → los metadatos se almacenan en memoria
    - Cada entrada 150 bytes, almacena archivos, directorios, bloques
    - 1 Millón archivos de 1 bloque → 300 MB
    - 1 Billón archivos →
  - Multiples escrituras y lecturas → HDFS sólo se escribe por un proceso al final del archivo

HDFS Bloques

Conceptos básicos

HDFS Bloques

- Un disco tiene un tamaño mínimo que puede ser escrito o leído
- Los archivos se dividen en bloques múltiplos de los del disco
- Tamaño de los bloques del disco 512 byte
- Trasparente al usuario

HDFS Bloques

- El mismo concepto distinto tamaño
- Normalmente 128 MB
- Archivos con bloques que pueden estar en diferentes unidades
- Un archivo de tamaño menor que el bloque no ocupa todo el bloque
  - SI el archivo es de 1 MB ocupa 1MB no 128 MB

HDFS Bloques

- Tamaño 128 MB
- Busca minimizar el tiempo de búsqueda
- Si el bloque es grande el tiempo de copiar es muy superior al de buscar
- Permite copiar a la velocidad de transferencia

HDFS Bloques

- Para un tiempo de búsqueda de 10 ms y una velocidad de trasferencia de 100 MB/s si queremos que el tiempo de búsqueda sea el 1% de la trasferencia el bloque tiene que ser de 100 MB

HDFS Bloques

- HDFS trabaja con nivel de abstracción a la hora de manejar bloques
- Beneficios
  - Un archivo puede ser mayor que un disco, los bloques se pueden almacenar en cualquier disco del cluster
  - Tratar como unidad de abstracción el bloque en vez del archivo simplifica la gestión
    - Tienen un tamaño fijo para los cálculos
    - Los metadatos no se almacenan con los datos

HDFS Bloques - tolerancia

- Trabajar con bloques permite realizar replicación para mejorar tolerancia a fallos
- Se replican en máquinas físicas distintas (3 normalmente)
- Si uno falla, otro puede ser leído de forma trasparente
- Un bloque corrompido puede sustituirse para mantener el factor de replicación
- Herramienta para consultar bloques “fsck”

HDFS nameNode y datanode

La clave de hdfs

HDFS Bloques

- HDFS tiene dos tipos de nodos
- Trabajan con el patrón maestro-obrero
- Maestro → NameNode
- Obrero → DataNode

HDFS NameNode

- El cálculo de memoria para un NameNode es aproximado
- Tiene una referencia de cada bloque de archivos
- Depende del número de bloques por archivo, longitud, directorios, versión
- Una estimación 1000MB de memoria 1 Millón de bloques
- Hadoop-env.sh. Se puede configurar el tamaño en JVM (Java Virtual Machine)
  - Hadoop-Namenode_OPTS y Hadoop-Secondary_OPTS

HDFS Bloques – NameNode

- Mantiene el árbol del sistema de archivos
- Almacena de forma persistente en dos archivos
  - Namespace image
  - Edit Log
- Conoce la ubicación de los DataNode, pero no los guarda de forma persistente
- El código de usuario no necesita saber de estos elementos para funcionar

HDFS tolerancia

- Sin el NameNode no se puede recuperar el sistema de archivos
- Hadoop tiene dos métodos para tolerancia a fallos
  - Backup→ los archivos que hacen persistente el NameNode.
    - Hace varias copias de los archivos
    - Asincrónos y atómicos
    - Normalmente escribir en el disco local y en sistema de archivos remoto

HDFS Bloques - DATAnode

- Son el caballo de trabajo
- Almacena y recupera bloques cuando se lo indican los clientes o NameNode
- Reporta al NameNode periódicamente la lista de nodos que almacena

HDFS Tolerancia

- NameNode Secundario
  - No actua como NameNode
  - Establece puntos de control cruzando el Edit Log con el Namespace Image para que el log no crezca demasiado
  - Se suele ejecutar en otra máquina por las necesidades de hardware

hDFS caching

- Normalmente la información se le del disco
- Por frecuencia de la lectura se pueden cachear
- Se cachea en un DataNode en memoria
- Se mejora el rendimiento (MapReduce y Spark)
- Las aplicaciones de usuario especifican los archivos y el tiempo

HDFS FEderation

- NameNode tiene información de todo el sistema de archivos
- Al ser muy grande puede ser una limitación para el escalado del sistema
- HDFS Federation permite usar varios NameNode
- Cada uno gestiona una parte del sistema
- Por ejemplo uno para /user y otro para /share

HDFS Federation

- Cada NameNode gestiona un espacio de nombres que contiene todos los bloques de un espacio de nombres
- Estos espacios de nombres son independientes para que no se comuniquen
- La caída de un NameNode no afecta al otro

Alta Disponibilidad

Conceptos

HDFS alta Disponibilidad

- Replicación de metadatos del namenode (local, NFS) + name node secundario para checkpoints= protección perdida de datos
- NO OFRECE ALTA DISPONIBILIDAD
- NameNode sigue siendo clave (SPOF Single ponit of Faliure)
- Si falla cae escritura, lectura, listado de directorios
- Si falla mapeo Archivo-Bloque todos los clientes (incluido MapRedude) falla
- Hadoop sistema KO

HDFS Alta Disponibilidad

- Hasta que no se reconstruya el namenode Hadoop está fuera de juego
- Es un caso poco habitual
- Tiene un coste de tiempo muy alto
- Importante tenerlo en cuenta para preparar un plan de recuperación para estos casos.

HDFS Alta Disponibilidad

- A partir de Hadoop 2.x
- 2 Namenodes en configuración activo/reserva
- Fallo del activo → El que está en reserva sigue con las tareas
- El paso es automático
- Son necesarios cambios architectura

HDFS Alta disponibilidad

Cambios en la arquitectura

- Namenode Edit Log almacenar en almacenamientos compartidos.
  - El nodo de reserva lee el final del log para sincronizarse con el nodo activo y continua leyendo las nuevas entradas
- DataNodes → comunicar a los dos namenodes.
  - Tienen que enviar los informes de bloques ya que estos se almacenan en memoria y no en disco

HDFS Alta Disponibilidad

Cambios (continuación)

- Las aplicaciones cliente implementen mecanismos para el fallo de namenode trasparentes para el usuario
- El rol del namenode secundario es asumido por el namenode de reserva
  - Realiza puntos del control del espacio de nombres gestionado por el de principal

HDFS Alta Disponibilidad

- Hay dos opciones para el almacenamiento compartido en alta disponibilidad
- NFS (Network File Sistem)
- QJM (Quorum Journal Manager) → Implementación recomendada de HDFS para alta disponibilidad

HDFS Alta disponibilidad

- QJP es una implementación de HDFS diseñada para ofrecer alta disponibilidad para el Edit Log.
- Se ejecuta como un conjunto de archivos de diario y cada edición se tienen que escribir en la mayoría de los nodos
- Normalmente hay tres nodos, por lo que puede fallar uno y todavía tener alta disponbilidad
- Es similar a la forma de trabajar de ZooKeeper

HDFS Alta Disponibilidad

- Si el namenode falla el de reserva puede activarse muy rápido
  - Dispone del último estado de la memoria (acceso a las entradas del Edit Log y el mapa de bloques actualizado
- Aunque está disponible en decimas de segundo en la realidad puede tardar unos minutos → El sistema tiene que decidir si el namenode principal ha caído o no
- Si el namenode de reserva no se inicia el administrador puede iniciarlo desde un arranque en frío ya que Hadoop está configurado para ello

HDFS Alta Disponibilidad - balanceo

- La transición entre un namenode a otro lo gestiona el Failover Controller
- La implementación por defecto utiliza ZooKeeper
- Se asegura que sólo un namenode está activo
- Cada namenode ejecuta un proceso mínimo del control de conmutación
  - Un pulso que envía genera un failover si no responde

Alta Disponibilidad - Balanceo

- La conmutación se puede iniciar manualmente → mantenimiento
- En estos casos el namenode activo puede pensar que sigue estando activo ya que puede haber un retraso en las comunicaciones
- La implementación en Alta Disponibilidad de Hadoop se asegura que esto no pueda suceder con el método llamado Fencing
  - QJM sólo permite a un namenode escribir en el Edit Log, pero todavía puede leer el nodo activo anterior

Alta Disponibilidad - Fencing

- QJM recomendable escribir comando SSH para matar el proceso del namenode
- NFS, es necesario establecer un método más estricto ya que no se puede limitar a que escriba sólo un namenode
- Estos métodos pueden ser
  - Limitar el acceso del namenode al sistema compartido
  - Descativar el puerto de red vía comando remoto
  - STONISH (Shoot the other node in the head) → apagar la máquina

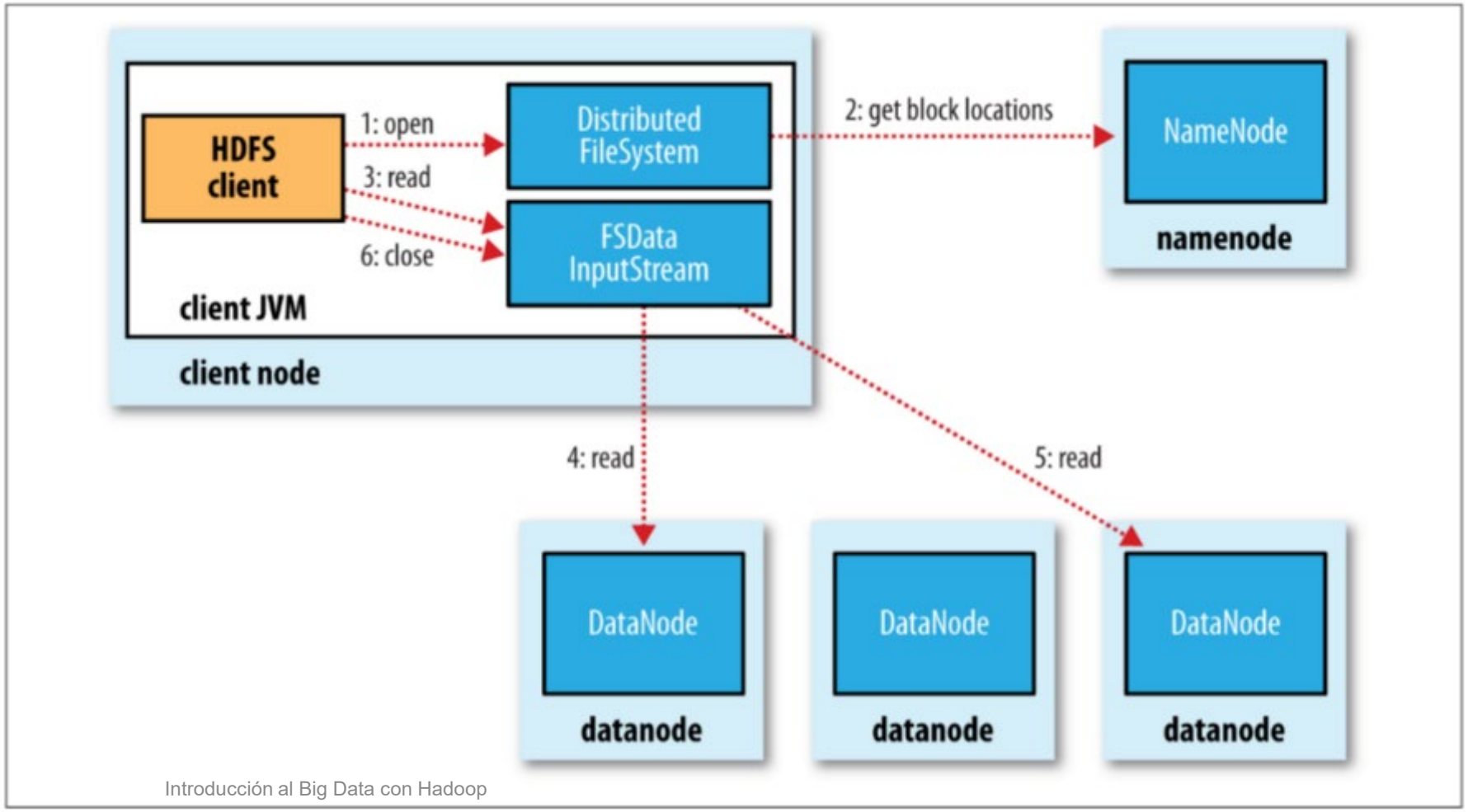
HDFS – JAVA Interface

Flujo de datos y ejemplos de lectura- escritura en java Interface

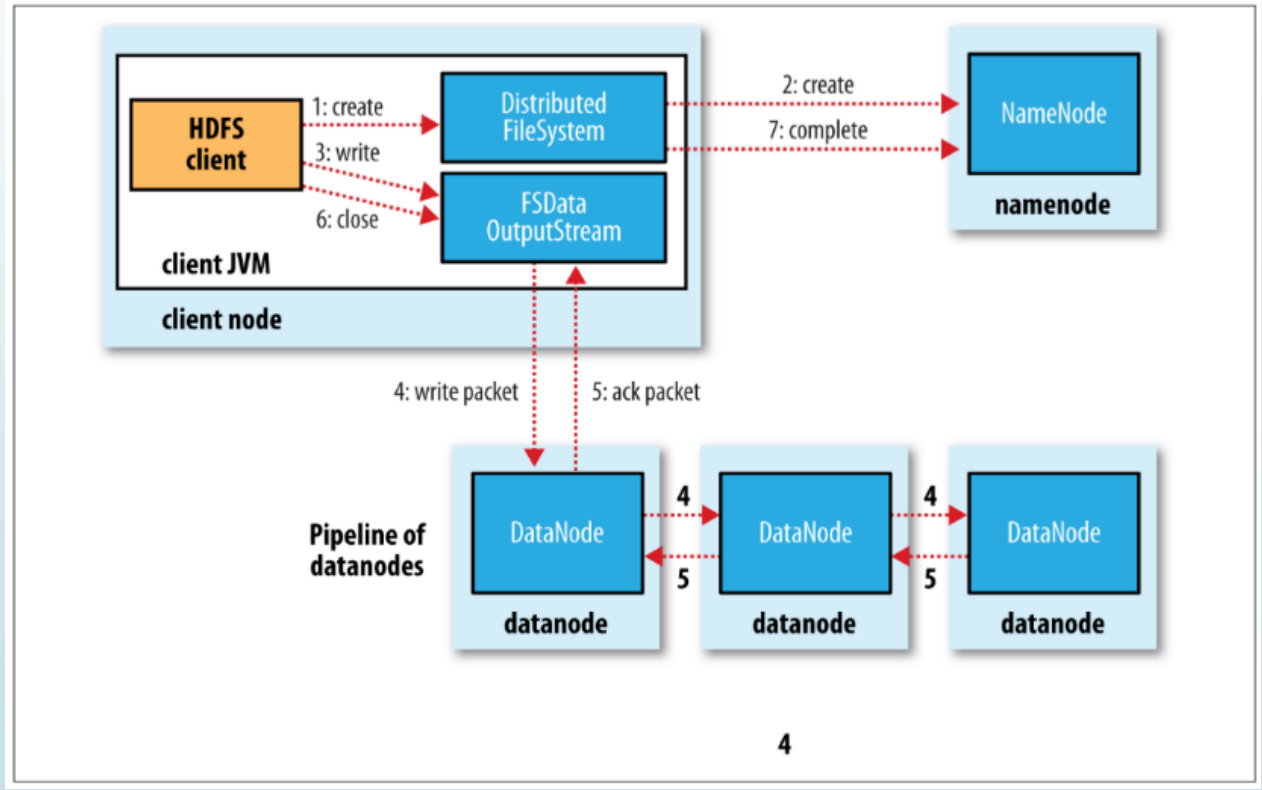
Java interface

- Hadoop tiene una visión abstracta del sistema de archivos
- HDFS es una implementación
- Interface está definida en org.apache.hadoop.fs.FileSystem
- Al estar escrito en JAVA el formato más común es utilizar la clase abstracta FileSystem
- DistribuitedFileSystem es la implementación de HDFS

HDFS Flujo de datos - Lectura



Flujo de datos - Escritura



Java Interface – leer datos

- Leer datos desde Hadoop URL con java.net.URL

```
InputStream in = null;
try {
  in = new URL("hdfs://host/path").openStream();
  // process in
} finally {
  IOUtils.closeStream(in);
}
```

Java Interface – Leer datos

- Mostrar información de un archivo utilizando URLStreamHandler

```
public class URLCat {

  static {
    URL.setURLStreamHandlerFactory(new FsUrlStreamHandlerFactory());
  }

  public static void main(String[] args) throws Exception {
    InputStream in = null;
    try {
      in = new URL(args[0]).openStream();
      IOUtils.copyBytes(in, System.out, 4096, false);
    } finally {
      IOUtils.closeStream(in);
    }
  }
}
```

JAVa interface Leer datos

- Si no es posible utilizar URLStreamHand lerFactory se puede utilizar API FyleSystem
- El objeto configuración contiene la información del cliente o servidor almacenada en archivos como *etc/hadoop/core-site.xml*

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

Java InterFace – Leer datos

- Una vez tenemos la instancia de FileSystem, invocamos el método Open
- Si no se especifica el tamaño del buffer se asigna por defecto 4 KB

```
public FSDataInputStream open(Path f) throws IOException
public abstract FSDataInputStream open(Path f, int bufferSize) throws IOException
```

Java InterFACE Leer datos

% hadoop FileSystemCat hdfs://localhost/user/tom/text.txt

```
public class FileSystemCat {

  public static void main(String[] args) throws Exception {
    String uri = args[0];
    Configuration conf = new Configuration();
    FileSystem fs = FileSystem.get(URI.create(uri), conf);
    InputStream in = null;
    try {
      in = fs.open(new Path(uri));
      IOUtils.copyBytes(in, System.out, 4096, false);
    } finally {
      IOUtils.closeStream(in);
    }
  }
}
```

JAVA InterFACE Leer datos

- El método Open de FileSystem Devuelve un FsDataInputStream - permite acceso aleatorio

```java
public class FileSystemDoubleCat {

  public static void main(String[] args) throws Exception {
    String uri = args[0];
    Configuration conf = new Configuration();
    FileSystem fs = FileSystem.get(URI.create(uri), conf);
    FSDataInputStream in = null;
    try {
      in = fs.open(new Path(uri));
      IOUtils.copyBytes(in, System.out, 4096, false);
      in.seek(0); // go back to the start of the file
      IOUtils.copyBytes(in, System.out, 4096, false);
    } finally {
      IOUtils.closeStream(in);
    }
  }
}
```

JavA Interface - Escribir

- FileSystem tiene varios métodos
- Path – crea el archivo y devuelve un stream para escribir
- Dispone de sobrecarga – Progressable para indicar la evolución de la escritura
- Se puede añadir información con el método Append

JaVA Inteface - Escritura

- Copiar archivo a Hadoop con indicador de progresión

```
public class FileCopyWithProgress {
  public static void main(String[] args) throws Exception {
    String localSrc = args[0];
    String dst = args[1];

    InputStream in = new BufferedInputStream(new FileInputStream(localSrc));

    Configuration conf = new Configuration();
    FileSystem fs = FileSystem.get(URI.create(dst), conf);
    OutputStream out = fs.create(new Path(dst), new Progressable() {
      public void progress() {
        System.out.print(".");
      }
    });

    IOUtils.copyBytes(in, out, 4096, true);
  }
}
```

Java interface - directorios

- Podemos crear un directorio con el método mkdirs() de java.io.File
- Normalmente no se utiliza ya que al utilizar create() se construye la estructura de directorios

```
public boolean mkdirs(Path f) throws IOException
```

JAvA Interface - Metadata

- La clase FileStatus encapsula todo los metadatos de archivos y directorios
  - Longitud, tamaño del bloque, replicación, hora de modificación, propietario e información de permisos
- El método GetFileStatus() de FileSystem obtiene esta información

Java Interface - Metadatos

- Ejemplo UnitTesting FileStatus()

```
public void fileStatusForFile() throws IOException {
  Path file = new Path("/dir/file");
  FileStatus stat = fs.getFileStatus(file);
  assertThat(stat.getPath().toUri().getPath(), is("/dir/file"));
  assertThat(stat.isDirectory(), is(false));
  assertThat(stat.getLen(), is(7L));
  assertThat(stat.getModificationTime(),
      is(lessThanOrEqualTo(System.currentTimeMillis())));
  assertThat(stat.getReplication(), is((short) 1));
  assertThat(stat.getBlockSize(), is(128 * 1024 * 1024L));
  assertThat(stat.getOwner(), is(System.getProperty("user.name")));
  assertThat(stat.getGroup(), is("supergroup"));
  assertThat(stat.getPermission().toString(), is("rw-r--r--"));
}
```

Java Interface - Directorios

- Método ListStatus de FileSystem devuelve lista de objetos FileStatus
  - Si es un archivo devuelve un objeto
  - Si es un directorio devuelve en un vectos con todos los archivos y subdirectorios
- Dispone de sobrecarga incluyendo PathFilter para seleccionar los archivos

JAVA Interface - Directorios

- Ejemplo de obtener una colecc de rutas

```
public class ListStatus {

  public static void main(String[] args) throws Exception {
    String uri = args[0];
    Configuration conf = new Configuration();
    FileSystem fs = FileSystem.get(URI.create(uri), conf);
    Path[] paths = new Path[args.length];
    for (int i = 0; i < paths.length; i++) {
      paths[i] = new Path(args[i]);
    }

    FileStatus[] status = fs.listStatus(paths);
    Path[] listedPaths = FileUtil.stat2Paths(status);
    for (Path p : listedPaths) {
      System.out.println(p);
    }
  }
}
```

JAVA Interface - Globbing

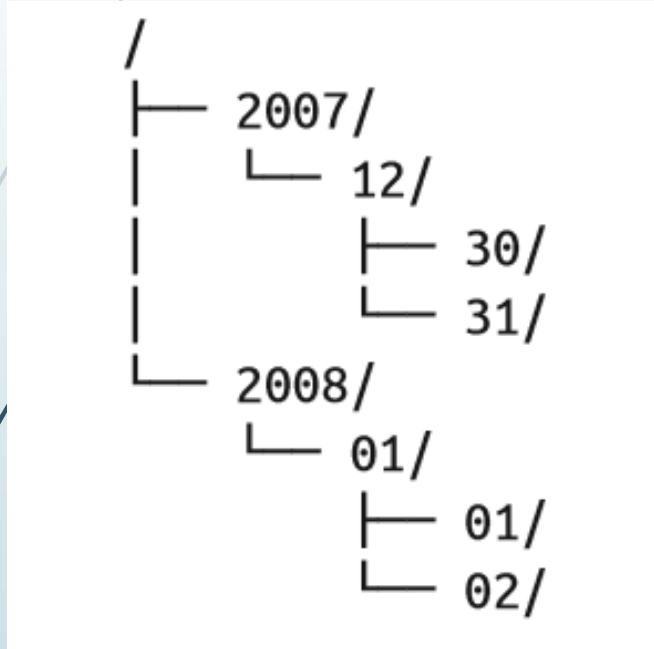
- Globbing es la utilizaciónd de caracteres especiales para identificar archivos o directorios en grupo en vez de uno en uno

```
public FileStatus[] globStatus(Path pathPattern) throws IOException
public FileStatus[] globStatus(Path pathPattern, PathFilter filter)
    throws IOException
```

JAVA INTERFACE

Glob	Name	Matches
*	*asterisk*	Matches zero or more characters
?	*question mark*	Matches a single character
[ab]	*character class*	Matches a single character in the set {a, b}
[^ab]	*negated character class*	Matches a single character that is not in the set {a, b}
[a-b]	*character range*	Matches a single character in the (closed) range [a, b], where a is lexicographically less than or equal to b
[^a-b]	*negated character range*	Matches a single character that is not in the (closed) range [a, b], where a is lexicographically less than or equal to b
{a,b}	*alternation*	Matches either expression a or b
\c	*escaped character*	Matches character c when it is a metacharacter

JAVA INTERFACE



/*	/2007 /2008
/*/*	/2007/12 /2008/01
/*/12/*	/2007/12/30 /2007/12/31
/200?	/2007 /2008
/200[78]	/2007 /2008
/200[7-8]	/2007 /2008
/200[^01234569]	/2007 /2008
/*/*/{31,01}	/2007/12/31 /2008/01/01
/*/*/3{0,1}	/2007/12/30 /2007/12/31
/*/{12/31,01/01}	/2007/12/31 /2008/01/01