Computación distribuida

Sergio Nesmachnow
(sergion@fing.edu.uy)

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Contenido

1. Computación distribuida y computación cloud
2. Procesamiento de grandes volúmenes de datos
3. El modelo de computación Map-Reduce
4. El framework Hadoop y su ecosistema
5. Almacenamiento: HDFS y HBase
6. Aplicaciones de Map Reduce sobre Hadoop: conteo, índice invertido, filtros
7. Procesamiento de datos en tiempo real: Apache Spark
8. Ejemplos de aplicaciones en Spark y el lenguaje Scala
9. Análisis de datos utilizando Spark y el lenguaje R.
10. Aplicaciones iterativas: Google Pregel y Apache Giraph

HDFS Y HBASE

UNIVERSIDAD
DE LA REPÚBLICA
URUGUAY

hadoop
HDFS

HDFS

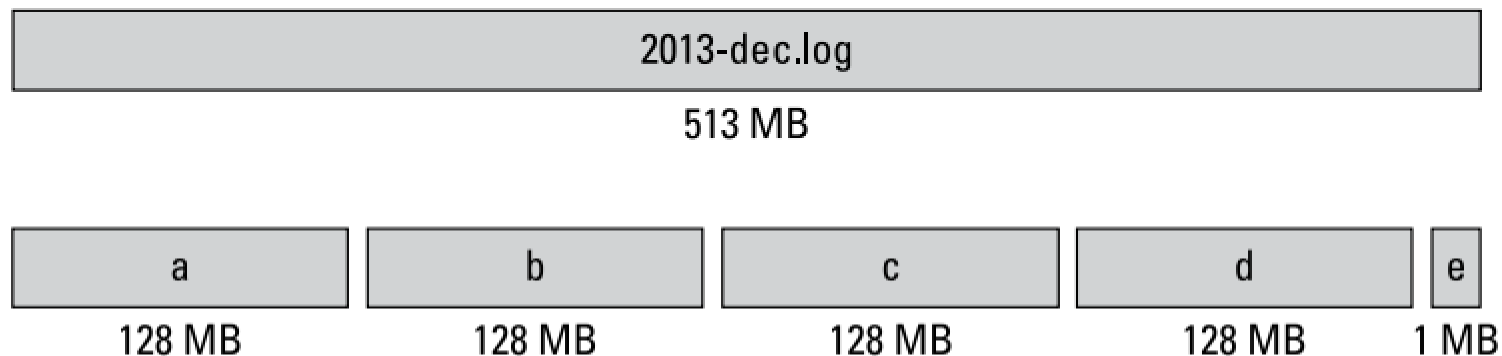
UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Hadoop Distributed File System (HDFS) es un sistema de archivos **distribuido, escalable y portátil**, escrito en Java, para Hadoop.
- Proporciona una arquitectura distribuida para el almacenamiento a gran escala, que puede ampliarse fácilmente escalando.
- Almacena archivos muy grandes (muchos petabytes de datos) a través de múltiples nodos (hosts): el tamaño de particionamiento (*split*) en Hadoop HDFS es configurable, por ejemplo, 64 o 128 MB (depende de la aplicación).
- HDFS se construyó con la idea de que el patrón de procesamiento de datos más eficiente es el de **escribir una vez y leer muchas veces**.
- Tolerancia a fallos mediante almacenamiento redundante configurable.

HDFS:Bloques

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Al almacenar un archivo en HDFS el sistema particiona el archivo en bloques individuales y almacena los bloques en varios nodos esclavos en el cluster de Hadoop.
- Tamaño configurable de forma general para la instancia de Hadoop y de forma individual para cada archivo.



HDFS: Bloques

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Tamaño de bloque grande (64MB o más):
  - Necesidad de tener que almacenar **grande volúmenes de datos**. Un tamaño de bloque chico implica que por cada archivo existan muchos bloques y eso sobrecarga al servidor que procesa la metadata que se usa para saber donde se guarda cada bloque de archivo.
  - HDFS está diseñado para tener un **alto *throughput*** (velocidad con la cual se transfiere un bloque de un nodo a otro) para que el procesamiento paralelo de estos grandes conjuntos de datos ocurra lo más rápido posible.
  - Minimizar el **tiempo de búsqueda en el sistema de archivo** de cada bloque. Con bloques grandes el tiempo de transferir los datos desde el disco es significativamente mayor al tiempo que se tarda en buscar cada comienzo de bloque en el disco.

HDFS: Bloques

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Beneficios de tener la abstracción de bloques:
  - El tamaño del archivo puede ser más grande que cualquiera de los discos en la red.
  - Encajan bien con la replicación para proporcionar tolerancia a errores y disponibilidad

HDFS: Namenodes y Datanodes

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Un clúster HDFS tiene dos tipos de nodos que operan en un patrón maestro-esclavo: un *namenode* (el maestro) y una serie de *datanodes* (trabajadores)
- ***Namenode:***
  - Maneja el espacio de nombres del sistema de archivos.
  - Mantiene el árbol del sistema de archivos y los metadatos para todos los archivos y directorios en el árbol. Almacena esa información en el disco local como dos archivos: La imagen del espacio de nombres (***.fsimage***) y el archivo de edición de logs (***.edits***).
  - Tiene la información de en cuales *datanodes* se encuentran todos los bloques para un archivo dado.
  - No almacena las ubicaciones de los bloques de forma persistente, porque esta información se reconstruye a partir de los *datanodes* cuando se inicia el sistema

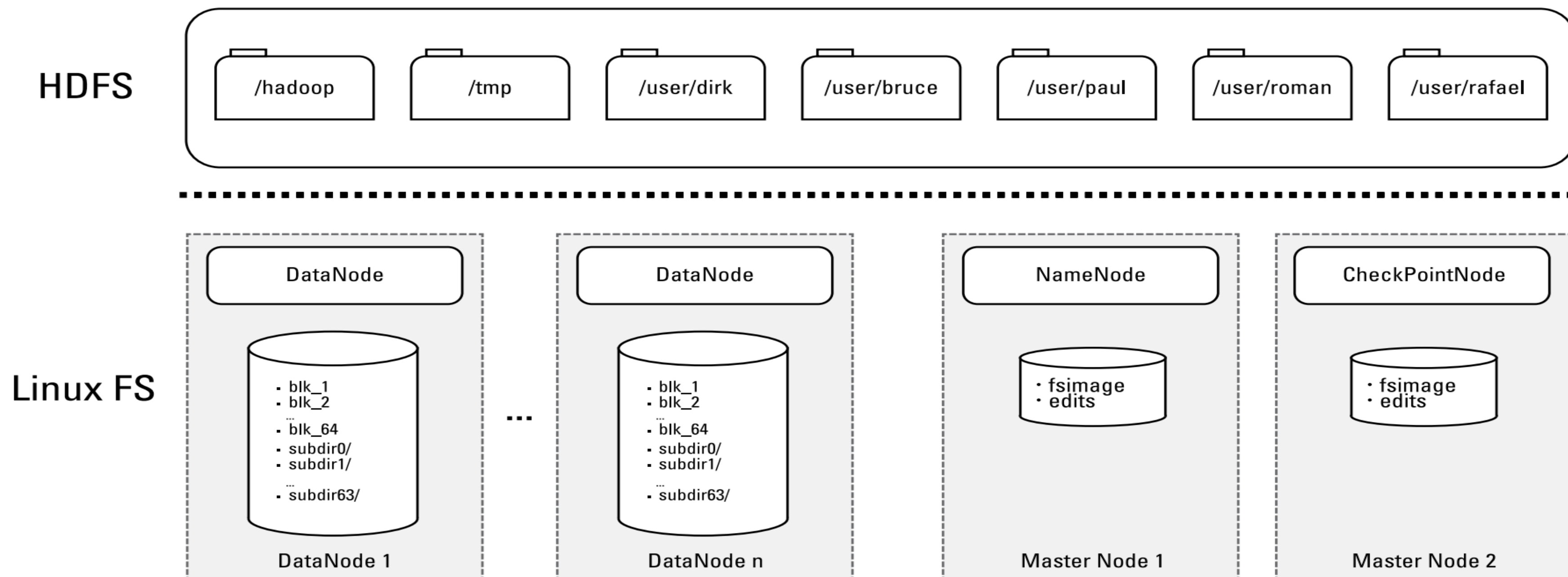
HDFS: Namenodes y Datanodes

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- *Datanode*:
  - Son los nodos donde se **almacenan y procesan los datos** (bloques de archivos DHSF guardados como archivos binarios).
  - Corre un proceso denominado *DataNode* el cual realiza un seguimiento de las porciones de datos que el sistema almacena.
  - El proceso *DataNode* se comunica de forma regular con el servidor *namenode* para informar sobre la salud y el estado de los datos almacenados localmente y la lista de bloques almacenados.
  - Desde los *datanodes* no se puede saber qué hay dentro de los bloques de datos que están almacenados.

HDFS: Namenodes y Datanodes

- Desde el punto de vista de un usuario de Hadoop, no se tiene idea de cuál de los nodos esclavos tiene las piezas de los archivos que se necesita procesar. Desde dentro de Hadoop, no se ven bloques de datos o cómo se distribuyen en el clúster: todo lo que se ve es una lista de archivos en HDFS.



HDFS: Namenodes y Datanodes

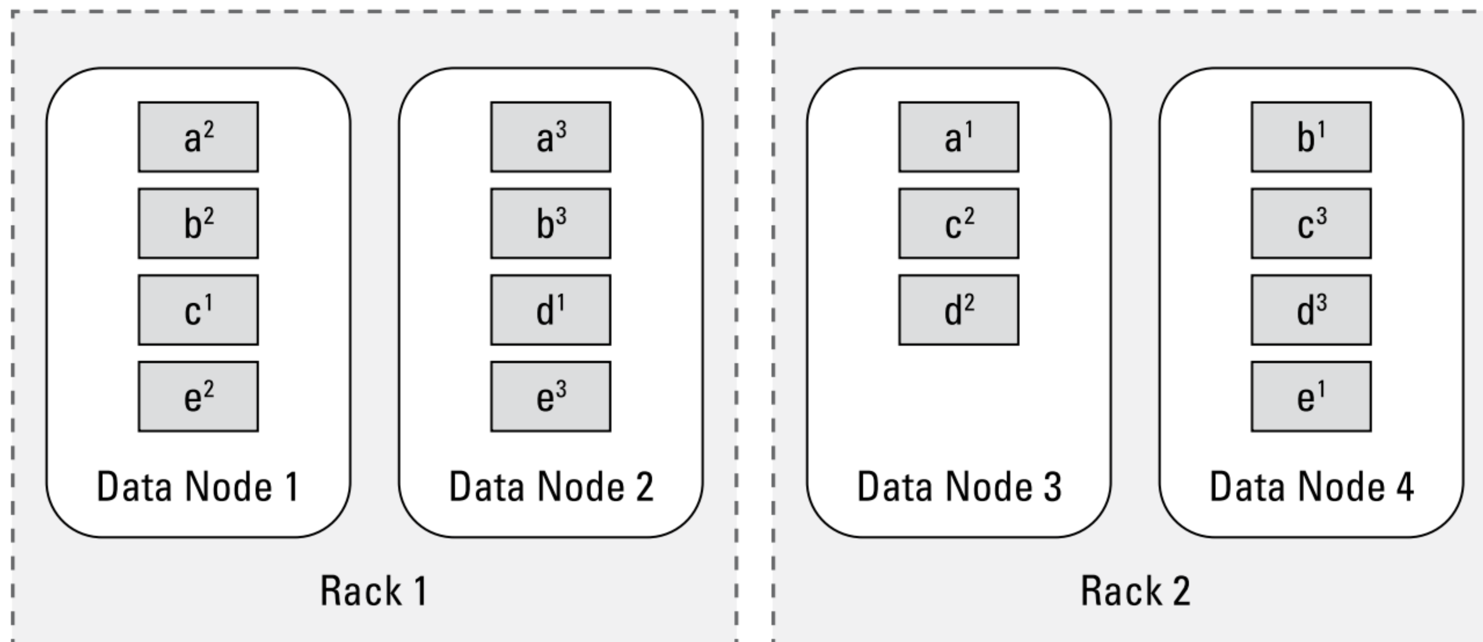
UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Si los archivos de la máquina que ejecuta el *namenode* fueran borrados, se perderían todos los archivos en el sistema ya que no habría forma de saber cómo reconstruir los archivos desde los bloques en los *datanodes*.
- Mecanismos de **resistencia a fallas del *namenode***:
  - Copia de seguridad de los archivos de metadatos. Almacenar el estado persistente en múltiples sistemas de archivos. Esta escritura de los archivos se hace en forma atómica y sincrónica. La configuración usual es escribir en el disco local y en un sistema NSF remoto.
  - Se ejecuta un segundo *namenode*. El curso de acción habitual en este caso es copiar los archivos de metadatos, que están respaldados en el sistema de archivos remoto NFS, al *namenode* secundario y ejecutarlo como *namenode* primario.

HDFS: Namenodes y Datanodes

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Cada bloque de datos tiene tres copias que están distribuidos uniformemente entre los nodos esclavos de modo que una copia del bloque seguirá estando disponible **independientemente de los fallos de disco, nodo o rack**. La figura muestra cinco bloques de datos a, b, c, d y e, distribuidos en el cluster de 2 raks con dos nodos cada uno. Las teres copias de cada bloque se distribuyen entre los distintos nodos.



HDFS: Operaciones básicas por línea de comandos

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Comandos similares a los de Linux.
- Sintaxis:

```
hadoop fs -comando_hdfs
```

- Para hacer referencia a un recurso en comandos shell de HDFS se utilizan URIs (uniform resource identifiers), tienen la siguiente sintaxis:

```
<scheme>://<authority>:<port>/<path>
```

  - ***scheme***: Es *hdfs* para archivos HDFS o *file* para el sistema de archivos local.
  - ***authority* y *port***: Nombre del host y nro de puerto asosiado al sistema de archivos.
  - ***path***: Ruta del el recurso dentro del sistema de archivos.
- Ejemplo:
  - `hdfs://localhost:9000/user/usuario_ejemplo/archivo.txt`

scheme ***authority*** ***port*** ***path***

HDFS: Operaciones básicas por línea de comandos

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- *Scheme*, *authority* y *port* son opcionales.
- Un archivo o directorio puede ser representado de dos maneras en HDFS.
- Ejemplos:
  - Usando URI abreviado de un directorio:
    `/user/usuario_ejemplo/`
  - Usando URI abreviado de un archivo:
    `/user/usuario_ejemplo/archivo.txt`
  - Usando URI completo de un directorio:
    `hdfs://localhost:9000/user/usuario_ejemplo/`
  - Usando URI completo de un archivo:
    `hdfs://localhost:9000/user/usuario_ejemplo/archivo.txt`

HDFS: Operaciones básicas por línea de comandos

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- En caso de no usar *scheme*, *authority* y *port* en los comandos HDFS se utilizan los valores por defecto configurados en el archivo core-site.xml que se encuentra en <HADOOP_INSTALATION_DIR>/etc/hadoop/core-site.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
<property>
  <name>fs.default.name</name>
      <value>hdfs://localhost:9000</value>
      </property>
</configuration>
```

scheme, authority y port usados por defecto

HDFS: Operaciones básicas por línea de comandos

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Por defecto en HDFS no se crean directorios para los usuarios por lo tanto se debe crear el directorio predeterminado de trabajo */user/$USER*, donde *$USER* es su nombre de usuario de inicio de sesión
- Ejemplo de creación de directorio para el usuario *bdata-user*:

```
hadoop fs -mkdir hdfs://localhost:9000/user/bdata-user
```

HDFS: Operaciones básicas por línea de comandos

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Por defecto en HDFS no se crean directorios para los usuarios por lo tanto se debe crear el directorio predeterminado de trabajo */user/$USER*, donde *$USER* es su nombre de usuario de inicio de sesión
- Ejemplo de creación de directorio para el usuario *bdata-user*:

```
hadoop fs -mkdir hdfs://localhost:9000/user/bdata-user
```

- También se puede usar la versión abreviada:

```
hadoop fs -mkdir /user/bdata-user
```

HDFS: Operaciones básicas por línea de comandos

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Por defecto en HDFS no se crean directorios para los usuarios por lo tanto se debe crear el directorio predeterminado de trabajo */user/$USER*, donde *$USER* es su nombre de usuario de inicio de sesión
- Ejemplo de creación de directorio para el usuario *bdata-user*:

```
hadoop fs -mkdir hdfs://localhost:9000/user/bdata-user
```

- También se puede usar la versión abreviada:

```
hadoop fs -mkdir /user/bdata-user
```

- Una vez creado el directorio predeterminado de trabajo para el usuario se puede omitir la parte de la ruta `/user/bdata-user` en los sucesivos comandos.
- Ejemplo crear un directorio en el directorio de trabajo del usuario:

```
hadoop fs -mkdir directorio_ejemplo
```

HDFS: Operaciones básicas por línea de comandos

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Ejemplo copiar un archivo desde el sistema de archivo local hacia HDFS:
```
hadoop fs -put data.txt /user/bdata-user/datacpy.txt
```
- Ejemplo copiar un archivo desde HDFS hacia el sistema de archivo local:
```
hadoop fs -get datacpy.txt data_from_hdfs.txt
```
- Ejemplo listar el contenido del directorio de trabajo del usuario:
```
hadoop fs –ls
```
- Comando similares a Linux, para obtener una lista completa de los comandos disponibles se puede recurrir al comando de ayuda:
```
hadoop fs -help
```

HDFS: Operaciones básicas por línea de comandos

- Ejemplo resultado de comando ls:

```
$ hadoop fs -ls
Found 2 items
-rw-r--r--   1 bdata-user supergroup  18 2018-04-01 22:23 /user/bdata-user/datacpy.txt
drwxr-xr-x   - bdata-user supergroup   0 2018-04-02 09:36 /user/bdata-user/directorio_ejemplo
```

HDFS: Operaciones básicas por línea de comandos

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Columna 1: permisos.

```
$ hadoop fs -ls
Found 2 items
-rw-r--r--   1 bdata-user supergroup   18 2018-04-01 22:23 /user/bdata-user/datacpy.txt
drwxr-xr-x   - bdata-user supergroup    0 2018-04-02 09:36 /user/bdata-user/directorio_ejemplo
```

- Comienza con “d” para directorio, “–” para archivo.
- Permisos lectura (r), escritura (w) y ejecución (x); para propietario, grupo y público, igual que en Linux. El permiso de ejecución se ignora para un archivo porque no se puede ejecutar un archivo en HDFS. Para un directorio este permiso es necesario para acceder a sus hijos

HDFS: Operaciones básicas por línea de comandos

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Columna 2: factor de replicación.

```
$ hadoop fs -ls
Found 2 items
-rw-r--r--   1 bdata-user supergroup  18 2018-04-01 22:23 /user/bdata-user/datacpy.txt
drwxr-xr-x   - bdata-user supergroup   0 2018-04-02 09:36 /user/bdata-user/directorio_ejemplo
```

- Para archivos representa la cantidad de veces que cada bloque del archivo se copia para implementar la replicación.
- Para los directorios no tiene sentido esta columna ya que son tratados como metadatos y almacenados por el namenode.

HDFS: Operaciones básicas por línea de comandos

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Columna 3: propietario.

```
$ hadoop fs -ls
Found 2 items
-rw-r--r--   1 bdata-user supergroup  18 2018-04-01 22:23 /user/bdata-user/datacpy.txt
drwxr-xr-x   - bdata-user supergroup   0 2018-04-02 09:36 /user/bdata-user/directorio_ejemplo
```

- Usuario propietario del archivo o directorio

HDFS: Operaciones básicas por línea de comandos

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Columna 4: superusuarios.

```
$ hadoop fs -ls
Found 2 items
-rw-r--r--   1 bdata-user supergroup  18 2018-04-01 22:23 /user/bdata-user/datacpy.txt
drwxr-xr-x   - bdata-user supergroup   0 2018-04-02 09:36 /user/bdata-user/directorio_ejemplo
```

- Nombre del grupo de superusuarios

HDFS: Operaciones básicas por línea de comandos

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Columna 5: tamaño.

```
$ hadoop fs -ls
Found 2 items
-rw-r--r--   1 bdata-user supergroup  18 2018-04-01 22:23 /user/bdata-user/datacpy.txt
drwxr-xr-x   - bdata-user supergroup   0 2018-04-02 09:36 /user/bdata-user/directorio_ejemplo
```

- Tamaño del archivo en bytes o 0 para directorios.

HDFS: Operaciones básicas por línea de comandos

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Columna 6: fecha y hora.

```
$ hadoop fs -ls
Found 2 items
-rw-r--r--   1 bdata-user supergroup  18 2018-04-01 22:23 /user/bdata-user/datacpy.txt
drwxr-xr-x   - bdata-user supergroup   0 2018-04-02 09:36 /user/bdata-user/directorio_ejemplo
```

- Fecha y hora de la ultima modificación del archivo o directorio.

HDFS: Operaciones básicas por línea de comandos

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Columna 7: nombre.

```
$ hadoop fs -ls
Found 2 items
-rw-r--r--   1 bdata-user supergroup  18 2018-04-01 22:23 /user/bdata-user/datacpy.txt
drwxr-xr-x   - bdata-user supergroup   0 2018-04-02 09:36 /user/bdata-user/directorio_ejemplo
```

- Nombre del archivo o directorio, incluyendo la ruta.

HDFS: Interfaz Java

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Ejemplo: Leer datos desde una URL de Hadoop.

```java
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);
    }
  }
}
```

HDFS: Interfaz Java

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Ejemplo: Leer datos desde una URL de Hadoop.

Se le indica a Java que se usará una URL de un esquema hdfs de Hadoop

```
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);
    }
  }
}
```

HDFS: Interfaz Java

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Ejemplo: Leer datos desde una URL de Hadoop.

```
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);
    }
  }
}
```

Se abre el stream en un objeto de la clase java.net.URL para leer el archivo desde HDFS

HDFS: Interfaz Java

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Ejemplo: Leer datos desde una URL de Hadoop.

```
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);
    }
  }
}
```

Método copyBytes() de la clase org.apache.hadoop.io.IOUtils. El tercer parámetro es el tamaño del buffer y el último indica que no se cierre el stream al terminar de copiar

HDFS: Interfaz Java

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Ejemplo: Leer datos desde una URL de Hadoop.

```
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);
    }
  }
}
```

Se cierra el input stream. (no es necesario cerrar el output stream Sytem.out)

HDFS: Interfaz Java

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Ejemplo: Leer datos desde una URL de Hadoop.

Problema: El método setURLStreamHandlerFactory() solo se puede llamar una vez por cada JVM.

```java
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);
    }
  }
}
```

HDFS: Interfaz Java

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Ejemplo: Leer datos desde una URL de Hadoop.

Problema: El método setURLStreamHandlerFactory() solo se puede llamar una vez por cada JVM.

```
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);
    }
  }
}
```

Asegurarse de que el método setURLStreamHandlerFactory() no se use en otra parte del código.

HDFS: Interfaz Java

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Ejemplo de salida:

```
% export HADOOP_CLASSPATH=hadoop-examples.jar
% hadoop URLCat hdfs://localhost/user/tom/quangle.txt
On the top of the Crumpetty Tree
The Quangle Wangle sat,
But his face you could not see,
On account of his Beaver Hat.
```

HDFS: Interfaz Java

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Ejemplo:Leer datos usando la API del sistema de archivos

- Para evitar usar el método URLStreamHandlerFactory se usa la API FileSystem para abrir un imput stream para un archivo.
- FileSystem es una API general de sistema de archivos, en esta caso se usa FileSystem con HDFS

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Ejemplo:Leer datos usando la API del sistema de archivos

```
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);
    }
  }
}
```

HDFS: Interfaz Java

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Ejemplo:Leer datos usando la API del sistema de archivos

```
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);
    }
  }
}
```

Configuración leída desde el classpath, como etc/hadoop/core-site.xm

HDFS: Interfaz Java

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Ejemplo:Leer datos usando la API del sistema de archivos

```
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);
    }
  }
}
```

Se obtiene una instancia del sistema de archivos

Usa el *scheme* y la *authority* del URI dado para determinar el sistema de archivos a usar. Usa el sistema de archivos predeterminado si no hay *scheme* especificado en el URI dado.

HDFS: Interfaz Java

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Ejemplo:Leer datos usando la API del sistema de archivos

```
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);
    }
  }
}
```

Se obtiene el input stream del archivo. Este método usa tamaño de buffer por defecto de 4KB

HDFS: Interfaz Java

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Ejemplo:Leer datos usando la API del sistema de archivos
- Ejemplo de salida:

```
% hadoop FileSystemCat hdfs://localhost/user/tom/quangle.txt
On the top of the Crumpetty Tree
The Quangle Wangle sat,
But his face you could not see,
On account of his Beaver Hat.
```

HDFS: Interfaz Java

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Ejemplo:Escribir datos usando la API del sistema de archivos

- El método de la clase FileSystem más simple para crear un archivo toma un objeto Path del archivo a crear y retorna un output stream en donde escribir.

```
public FSDataOutputStream create(Path f) throws IOException
```

- El método de la clase FileSystem más simple para crear un archivo toma un objeto Path del archivo a crear y retorna un output stream en donde escribir.
- Hay versiones sobrecargadas de este método que le permiten especificar si debe forzar sobrescribir los archivos existentes, el factor de replicación del archivo, el tamaño del búfer que se usará cuando se escriba el archivo, el tamaño de bloque para el archivo y los permisos de archivo.

HDFS: Interfaz Java

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Ejemplo:Escribir datos usando la API del sistema de archivos

- Se puede concatenar información a un archivo ya existente con el método append(). Útil para archivos de gran tamaño que crecen de forma progresiva en el tiempo, por ejemplo arhivos de logs.

```
public FSDataOutputStream append(Path f) throws IOException
```

- La implementación del método append() es opcional, HDFS lo soporta.

HDFS: Interfaz Java

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Ejemplo: Copiar un archivo desde el sistema de archivos local hacia HDFS

```
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);
  }
}
```

HDFS: Interfaz Java

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Ejemplo: Copiar un archivo desde el sistema de archivos local hacia HDFS

```
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);
  }
}
```

Imprime un punto cada vez que Hadoop llama al método progress (). Por ejemplo: después de que cada paquete de 64 KB de datos se escribe en el pipeline de datanodes. (Esto no está especificado por la API, por lo tanto puede variar en las distintas implementaciones)

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

FSDataOutputStream:

- El método create() en FileSystem devuelve un objeto del tipo FSDataOutputStream que, como FSDataInputStream, tiene un método para consultar la posición actual en el archivo:

```
package org.apache.hadoop.fs;

public class FSDataOutputStream extends DataOutputStream implements Syncable {

  public long getPos() throws IOException {
    // implementation elided
  }

  // implementation elided

}
```

- A diferencia de FSDataInputStream, FSDataOutputStream no permite hacer seeking. Esto se debe a que HDFS solo permite escrituras secuenciales en un archivo un archivo abierto o agregar datos a un archivo ya escrito.

HDFS: Interfaz Java

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Directorios:

- FileSystem provee el método mkdirs para crear directorios.

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

- Este método crea todos los directorios padres si no existen tal como lo hace el método mkdirs() de la calse java java.io.File.
- En el caso de crear un archivo no es necesario crear directorios ya que la función create() para archivos, ya crea los directorios padres.

HDFS: Interfaz Java

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Consultas al sistema de archivos (File metadata: FileStatus):

- Permite navegar la estructura de directorios del sistema de archivos y recuperar información sobre los archivos y directorios que almacena.
- La clase FileStatus encapsula los metadatos del sistema de archivos para archivos y directorios, que incluyen la longitud del archivo, el tamaño del bloque, la replicación, el tiempo de modificación, la propiedad y la información de permisos. El método getFileStatus () en FileSystem proporciona una forma de obtener un objeto FileStatus para un solo archivo o directorio.

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Consultas al sistema de archivos (Listar archivos):

```
public FileStatus[] listStatus(Path f) throws IOException
public FileStatus[] listStatus(Path f, PathFilter filter) throws IOException
public FileStatus[] listStatus(Path[] files) throws IOException
public FileStatus[] listStatus(Path[] files, PathFilter filter)
    throws IOException
```

- Cuando el argumento es un archivo, la variante más simple devuelve una matriz de objetos FileStatus de longitud 1. Cuando el argumento es un directorio, devuelve cero o más objetos FileStatus que representan los archivos y directorios contenidos en el directorio.
- Las variantes sobrecargadas permiten que se suministre un PathFilter para restringir los archivos y directorios que coincidan con el filtro.

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Ejemplo: listar una collección de rutas a la vez

- Si especifica una arreglo de rutas, el resultado es equivalente a llamar el método listStatus() para cada ruta y acumular los arreglos de objetos FileStatus en un único arreglo.

```
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);
      }
    }
  }
```

HDFS: Interfaz Java

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

File patterns:

- Es un requisito común procesar conjuntos de archivos en una sola operación. Por ejemplo, un trabajo de MapReduce para el procesamiento de registros podría analizar el valor de un mes de los archivos contenidos en una serie de directorios. En lugar de tener que enumerar cada archivo y directorio para especificar la entrada, es conveniente usar caracteres comodín para unir múltiples archivos con una sola expresión, una operación que se conoce como globbing.
- Hadoop proporciona dos métodos FileSystem para procesar globs:

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

- Los métodos globStatus() devuelven una arreglo de objetos FileStatus cuyas rutas coinciden con el patrón suministrado, ordenadas por ruta. Se puede especificar un PathFilter opcional para restringir aún más las coincidencias.

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

File patterns:

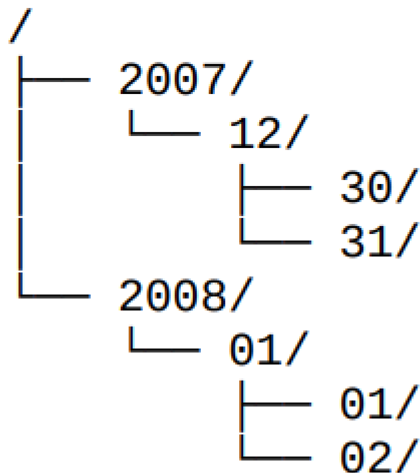
- Hadoop admite el mismo conjunto de caracteres glob que el shell bash de Unix

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

HDFS: Interfaz Java

Ejemplo File patterns: archivos de logs que se almacenan en una estructura de directorios organizada jerárquicamente por fecha.

Estructura de los directorios



Resultado de aplicar los siguientes patrones

Glob	Expansion
`/*`	/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

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

PathFilter

- Los patrones Glob no siempre son lo suficientemente potentes para describir un conjunto de archivos al que desea acceder. Por ejemplo, generalmente no es posible excluir un archivo particular usando un patrón global. Los métodos listStatus () y globStatus () de FileSystem toman un PathFilter opcional, que permite el control programático de la coincidencia.

```java
package org.apache.hadoop.fs;

public interface PathFilter {
  boolean accept(Path path);
}
```

- PathFilter es el equivalente de java.io.FileFilter para los objetos Path en lugar de objetos File.

HDFS: Interfaz Java

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Ejemplo PathFilter: excluir Path que coincidan con una expresión regular.

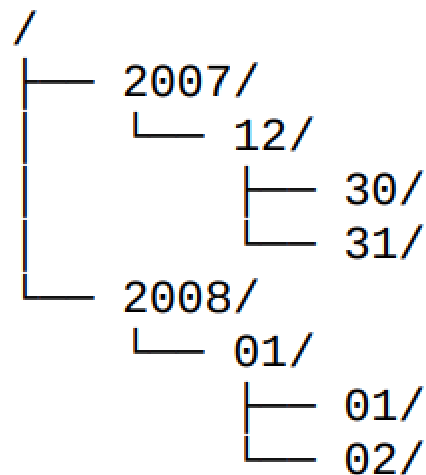
```
public class RegexExcludePathFilter implements PathFilter {

  private final String regex;

  public RegexExcludePathFilter(String regex) {
    this.regex = regex;
  }

  public boolean accept(Path path) {
    return !path.toString().matches(regex);
  }
}
```

- El filtro pasa solo aquellos archivos que no coinciden con la expresión regular. Una vez que glob selecciona un conjunto inicial de archivos para incluir, el filtro se utiliza para refinar los resultados.
- Aplicando la implementación RegexExcludePathFilter de PathFilter

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Ejemplo PathFilter: excluir Path que coincidan con una expresión regular.

Estructura de los directorios



Se aplica:

```
fs.globStatus(new Path("/2007/*/*"), new RegexExcludeFilter("^.*/2007/12/31$"))
```

Resultado : **/2007/12/30**

- Los filtros solo pueden actuar sobre la path de un archivo. No pueden usar las propiedades de un archivo, como la hora de creación, por ejemplo.

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Borrado de datos

- Se usa el método delete() de FileSystem para eliminar permanentemente archivos o directorios:

```
public boolean delete(Path f, boolean recursive) throws IOException
```

- Si f es un archivo o un directorio vacío, se ignora el valor de recursivo. Se elimina un directorio no vacío, junto con su contenido, solo si el recursivo es verdadero (de lo contrario, se lanza una IOException).

HDFS: Flujo de datos

Lectura de archivo

HDFS client
1: open
3: read
6: close
Distributed FileSystem
FSData InputStream
client JVM
client node
2: get block locations
NameNode
namenode
4: read
5: read
DataNode
datanode
DataNode
datanode
DataNode
datanode

HDFS: Flujo de datos

Lectura de archivo

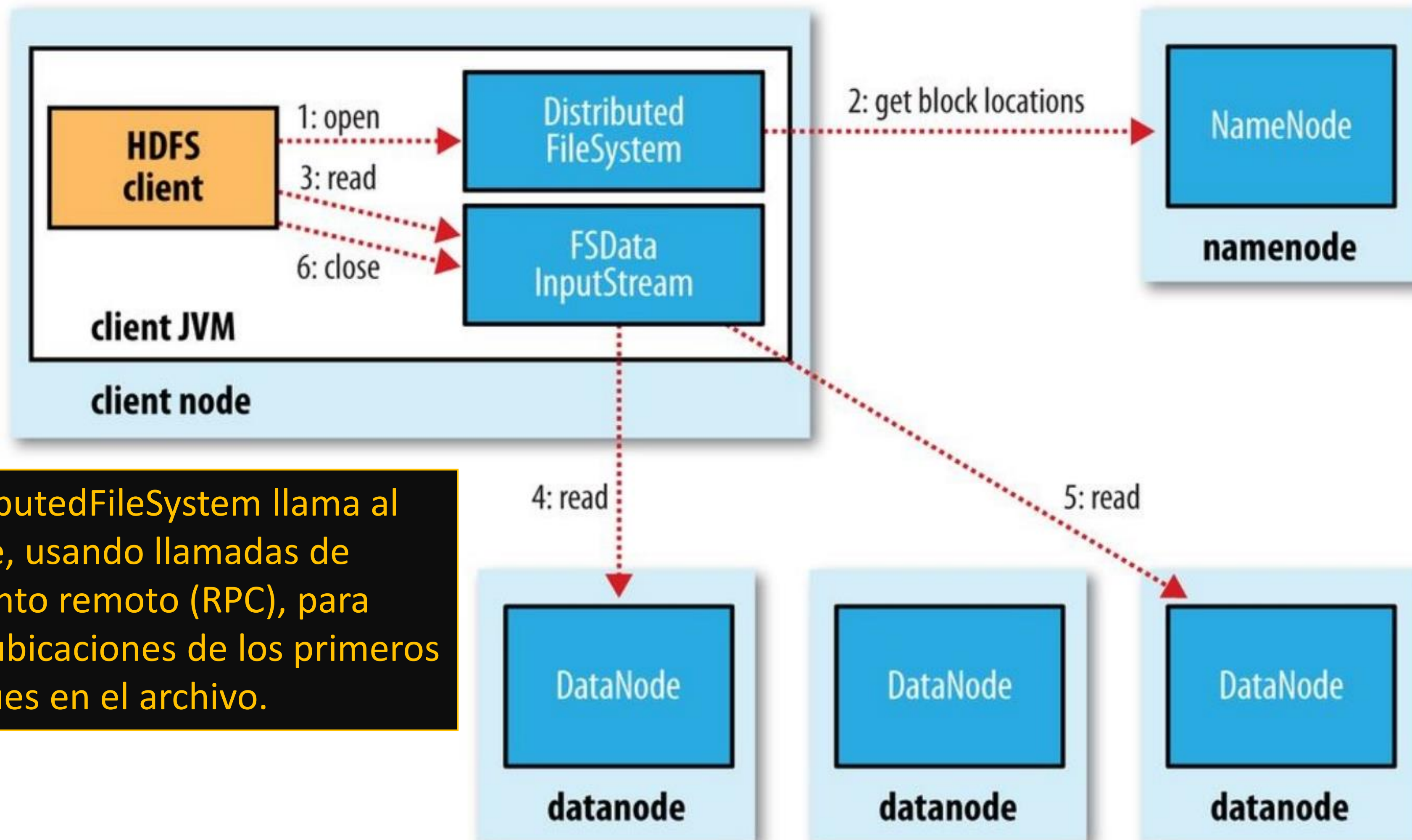
HDFS client
1: open
3: read
6: close
Distributed FileSystem
FSData InputStream
client JVM
client node
2: get block locations
NameNode
namenode
4: read
5: read
DataNode
datanode
DataNode
datanode
DataNode
datanode

Paso1: El cliente llama a open() en el objeto FileSystem, que para HDFS es una instancia de DistributedFileSystem

HDFS: Flujo de datos

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Lectura de archivo



Paso2: DistributedFileSystem llama al namenode, usando llamadas de procedimiento remoto (RPC), para determinar las ubicaciones de los primeros bloques en el archivo.

HDFS: Flujo de datos

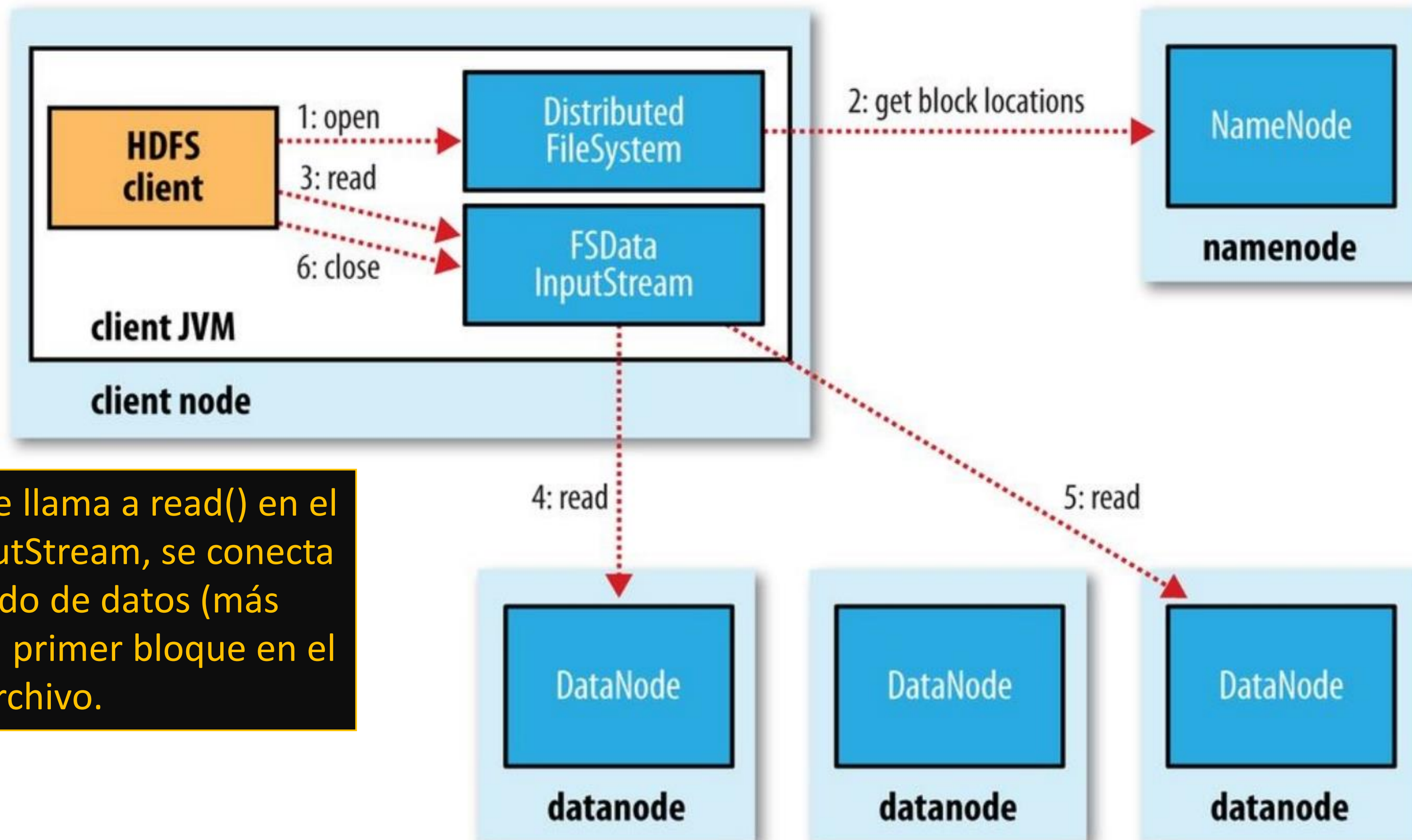
UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Lectura de archivo (paso 2)

- Para cada bloque, el namenode devuelve las direcciones de los nodos de datos que tienen una copia de ese bloque.
- Los nodos de datos se ordenan según su proximidad al cliente (según la topología de la red del clúster.
- DistributedFileSystem devuelve un FSDataInputStream al cliente para que pueda leer los datos. FSDataInputStream a su vez envuelve DFSInputStream, que administra el nodo de datos y la E/S namenode.

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Lectura de archivo



Paso 3: El cliente llama a read() en el stream. DFSInputStream, se conecta al primer nodo de datos (más cercano) para el primer bloque en el archivo.

HDFS: Flujo de datos

Lectura de archivo

HDFS client
1: open
3: read
6: close
Distributed FileSystem
FSData InputStream
client JVM
client node
2: get block locations
NameNode
namenode
4: read
5: read
DataNode
datanode
DataNode
datanode
DataNode
datanode

Paso 4: Los datos se transmiten desde el nodo de datos al cliente, que llama a read() repetidamente en el stream.

HDFS: Flujo de datos

Lectura de archivo

HDFS client
1: open
3: read
6: close
Distributed FileSystem
FSData InputStream
client JVM
client node
2: get block locations
NameNode
namenode
4: read
5: read
DataNode
datanode
DataNode
datanode
DataNode
datanode

Paso 5: Cuando se llega al final del bloque, DFSInputStream cerrará la conexión al nodo de datos, luego buscará el mejor nodo para el siguiente bloque. Los bloques se leen en orden, con el DFSInputStream abriendo nuevas conexiones a los nodos de datos a medida que el cliente lee el stream.

HDFS: Flujo de datos

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Lectura de archivo (paso 5)

- Ocurre de forma transparente para el cliente, que desde su punto de vista solo lee un flujo continuo.
- Se llamará al namenode para recuperar las ubicaciones de datanode para el siguiente lote de bloques según sea necesario.

HDFS: Flujo de datos

Lectura de archivo

HDFS client
1: open
3: read
6: close
Distributed FileSystem
FSData InputStream
client JVM
client node
2: get block locations
NameNode
namenode
4: read
5: read
DataNode
datanode
DataNode
datanode
DataNode
datanode

Paso 6: Cuando el cliente ha terminado de leer, llama a close() en el FSDataInputStream.

HDFS: Flujo de datos

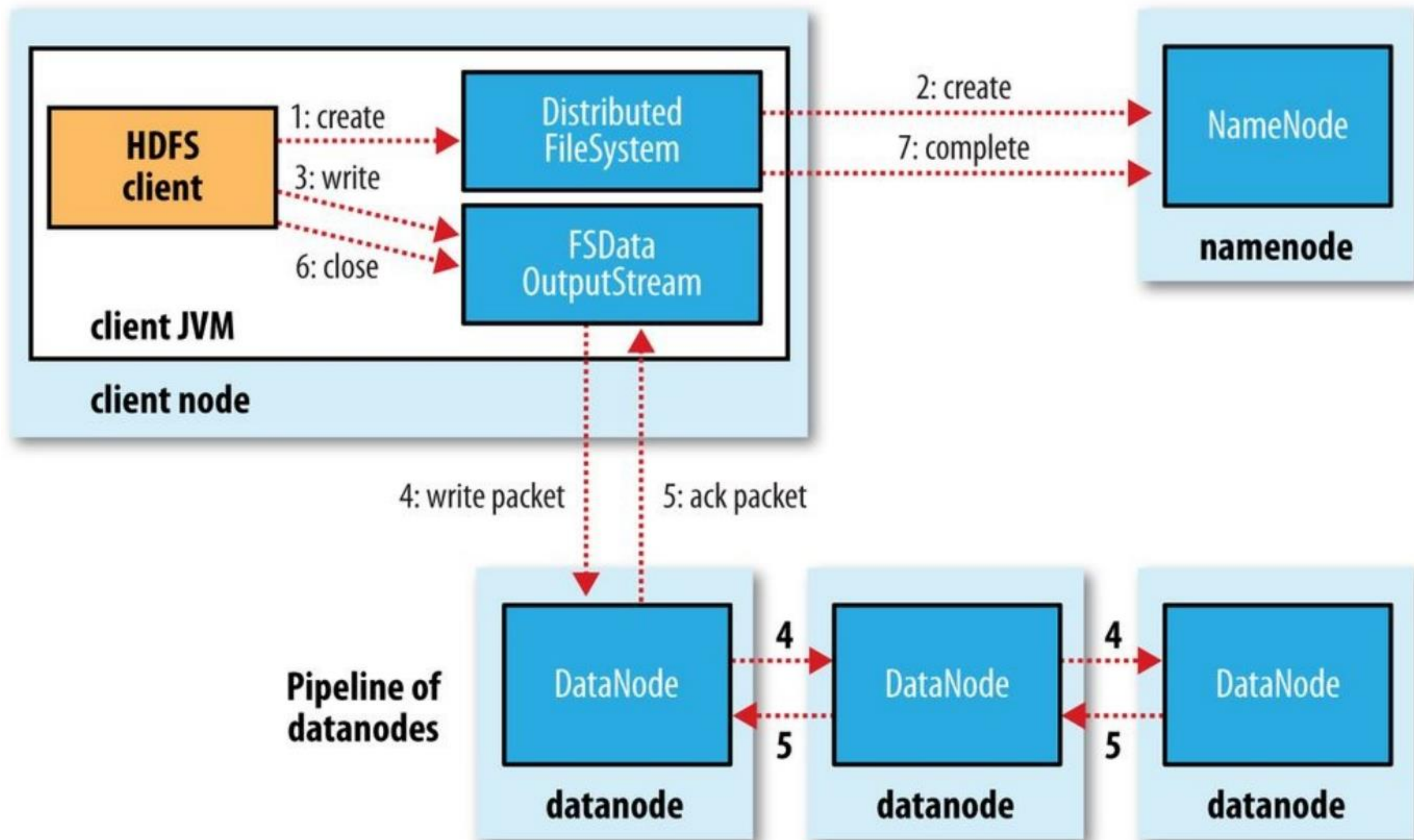
UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Lectura de archivo (paso 6)

- Durante la lectura, si el DFSInputStream encuentra un error al comunicarse con un datanode, intentará con el siguiente más cercano para ese bloque.
- DFSInputStream recordará los datanodes que han fallado para que no lo intente innecesariamente para bloques posteriores.
- DFSInputStream verifica las sumas de comprobación para los datos transferidos desde el nodo de datos. Si se encuentra un bloque dañado, DFSInputStream intenta leer una réplica del bloque desde otro nodo de datos; también informa el bloque corrupto al namenode.

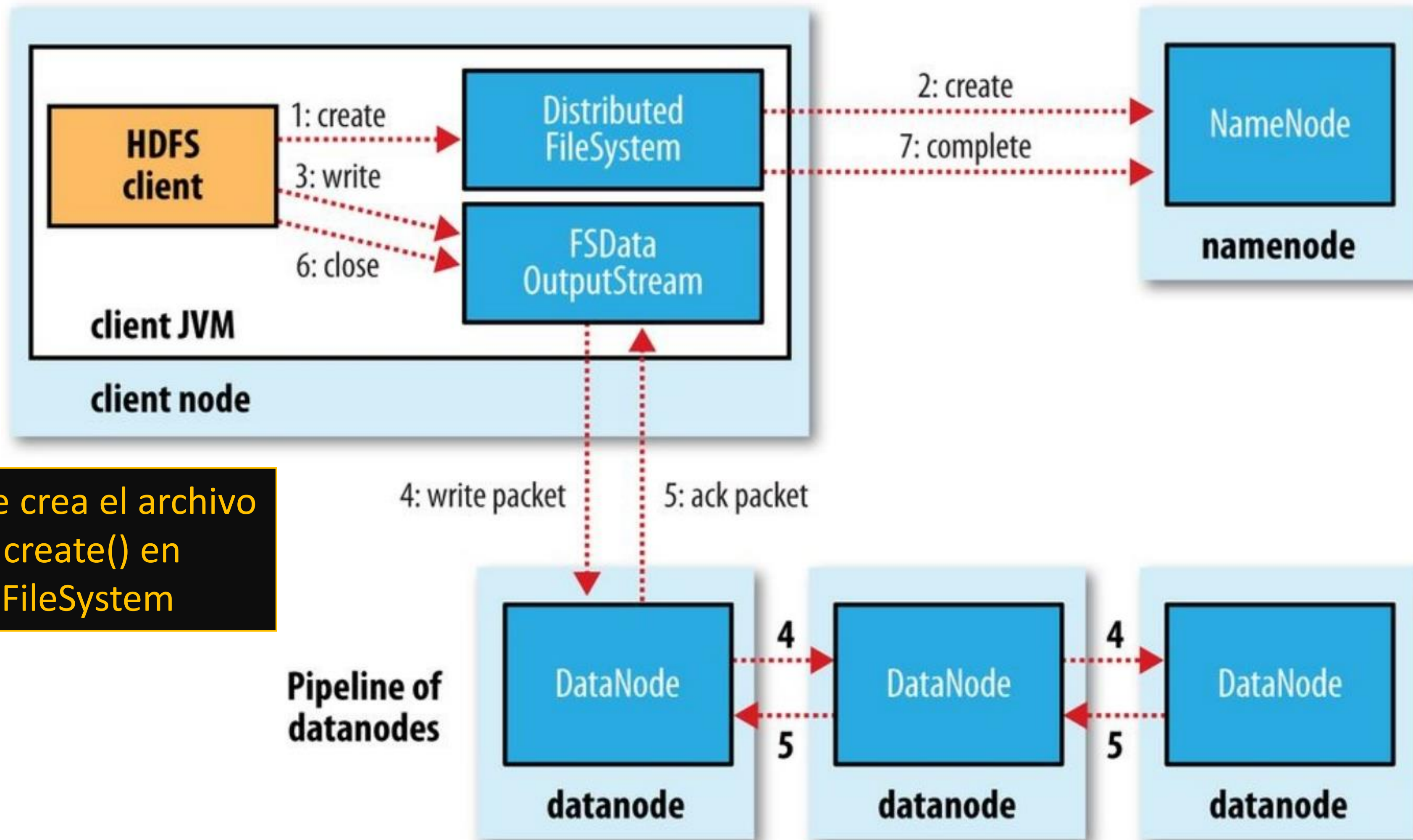
UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Escritura de archivo



UNIVERSIDAD DE LA REPÚBLICA URUGUAY

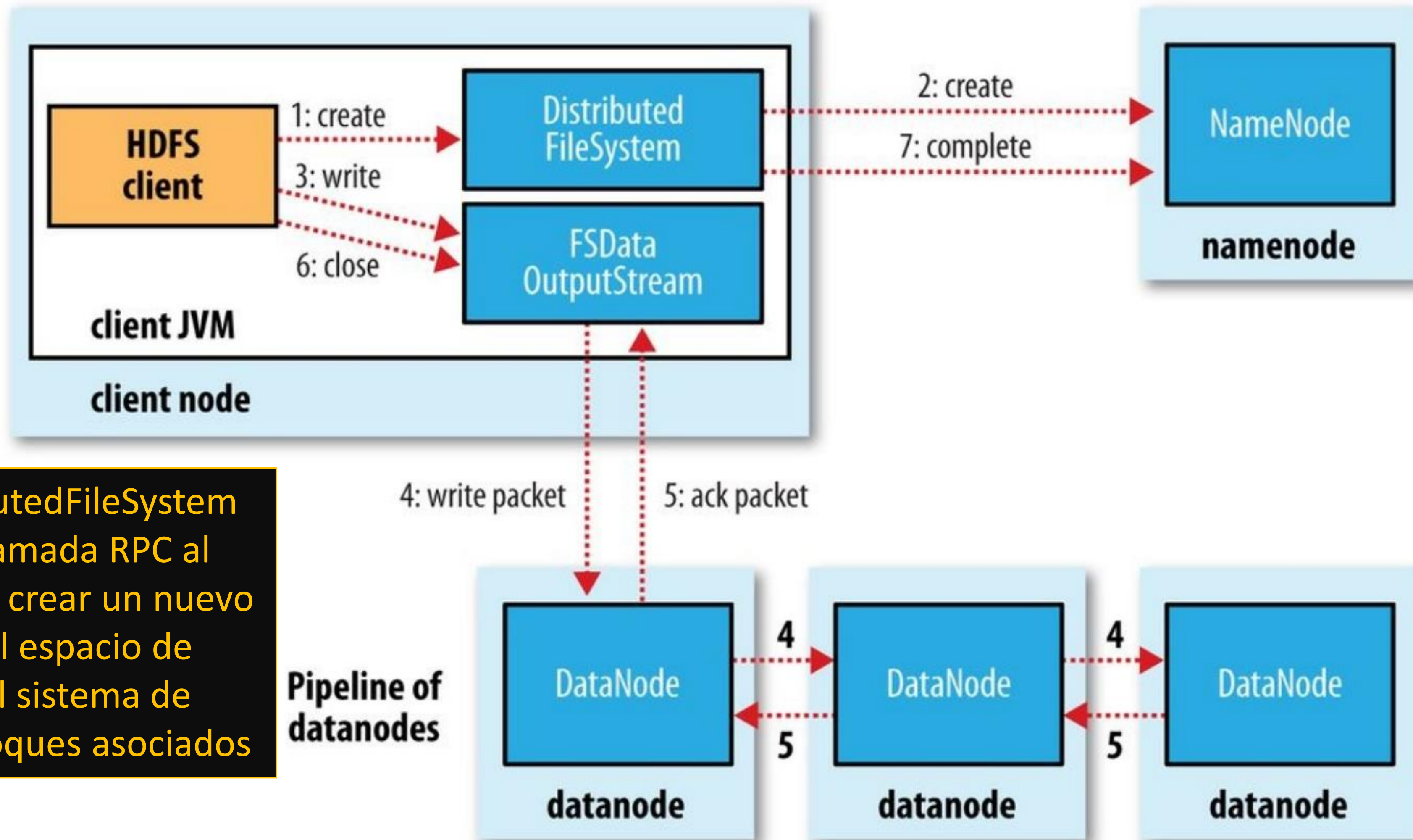
Escritura de archivo



Paso 1: El cliente crea el archivo llamando a create() en DistributedFileSystem

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Escritura de archivo



Paso 2: DistributedFileSystem realiza una llamada RPC al namenode para crear un nuevo archivo en el espacio de nombres del sistema de archivos, sin bloques asociados

HDFS: Flujo de datos

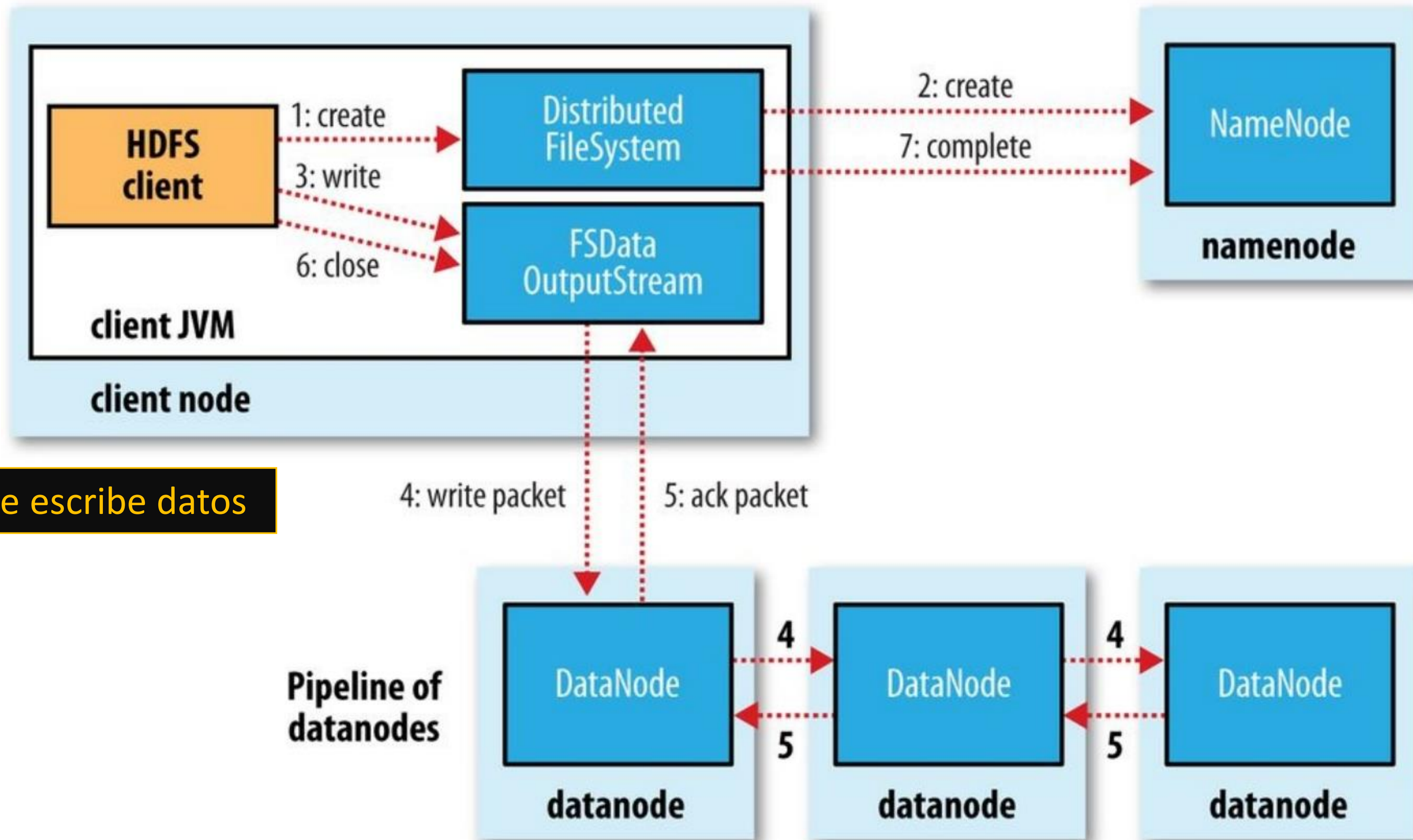
UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Escritura de archivo (paso 2)

- El namenode realiza varias verificaciones para asegurarse de que el archivo no exista y que el cliente tenga los permisos correctos para crear el archivo. Si estos controles pasan, el namenode hace un registro del nuevo archivo; de lo contrario, la creación del archivo falla y el cliente recibe una excepción IOException.
- DistributedFileSystem devuelve un FSDataOutputStream para que el cliente comience a escribir datos.
- FSDataOutputStream envuelve un objeto DFSOutputStream, que maneja la comunicación con los nodos de datos y namenode.

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Escritura de archivo



Paso 3: El cliente escribe datos

HDFS: Flujo de datos

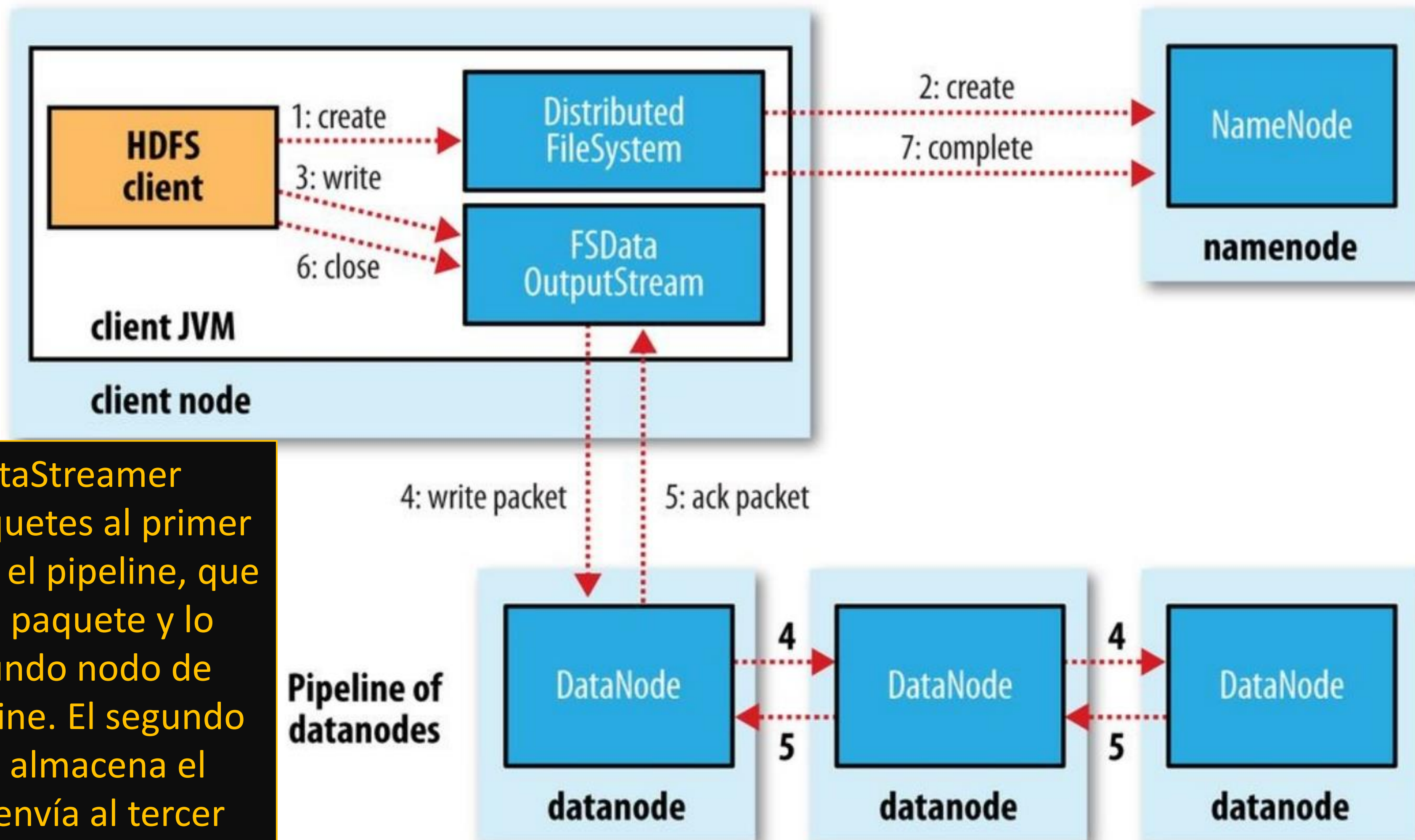
UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Escritura de archivo (paso 3)

- DFSOutputStream divide los datos en paquetes, que se escriben en una cola interna llamada cola de datos.
- La cola de datos es consumida por el DataStreamer, que es responsable de pedir al namenode que asigne nuevos bloques al seleccionar una lista de nodos de datos adecuados para almacenar las réplicas.
- La lista de nodos de datos forma un pipeline, y aquí asumiremos que el nivel de replicación es tres, por lo que hay tres nodos en el pipeline.

HDFS: Flujo de datos

Escritura de archivo

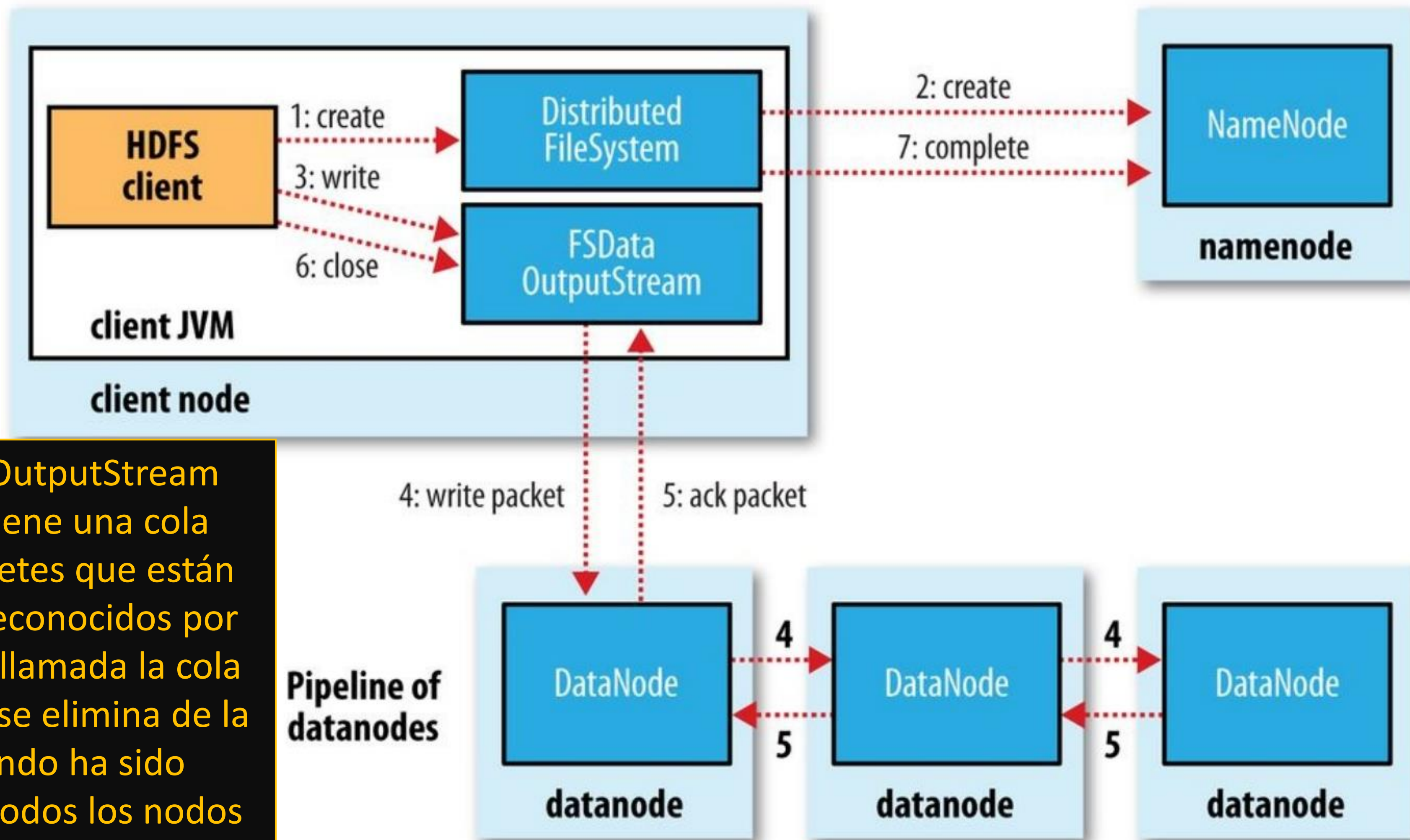


Paso 4: El DataStreamer transmite los paquetes al primer nodo de datos en el pipeline, que almacena cada paquete y lo reenvía al segundo nodo de datos en el pipeline. El segundo nodo de datos almacena el paquete y lo reenvía al tercer nodo de datos

HDFS: Flujo de datos

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Escritura de archivo



Paso 5: El DFSOutputStream también mantiene una cola interna de paquetes que están esperando ser reconocidos por nodos de datos, llamada la cola ack. Un paquete se elimina de la cola solo cuando ha sido reconocido por todos los nodos de datos en el pipeline

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Escritura de archivo (paso 5)

- Si cualquier nodo de datos falla mientras se escriben datos en él, entonces se toman las siguientes acciones:
  1) El pipeline será cerrado y todos los paquetes en la cola de ack se agregan al frente de la cola de datos para que los nodos de datos no pierdan ningún paquete.
  2) El bloque actual en los datanodes que no fallaron recibe una nueva identidad, que se comunica al namenode, de modo que el bloque parcial en el nodo de datos que falló se eliminará si el nodo se recupera posteriormente.
  3) El nodo de datos que falló se elimina del pipeline, y un nuevo pipeline se construye a partir de los datanodes que no fallaron.
  4) El resto de los datos del bloque se escribe en los datanodes del pipeline que no fallaron.

HDFS: Flujo de datos

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

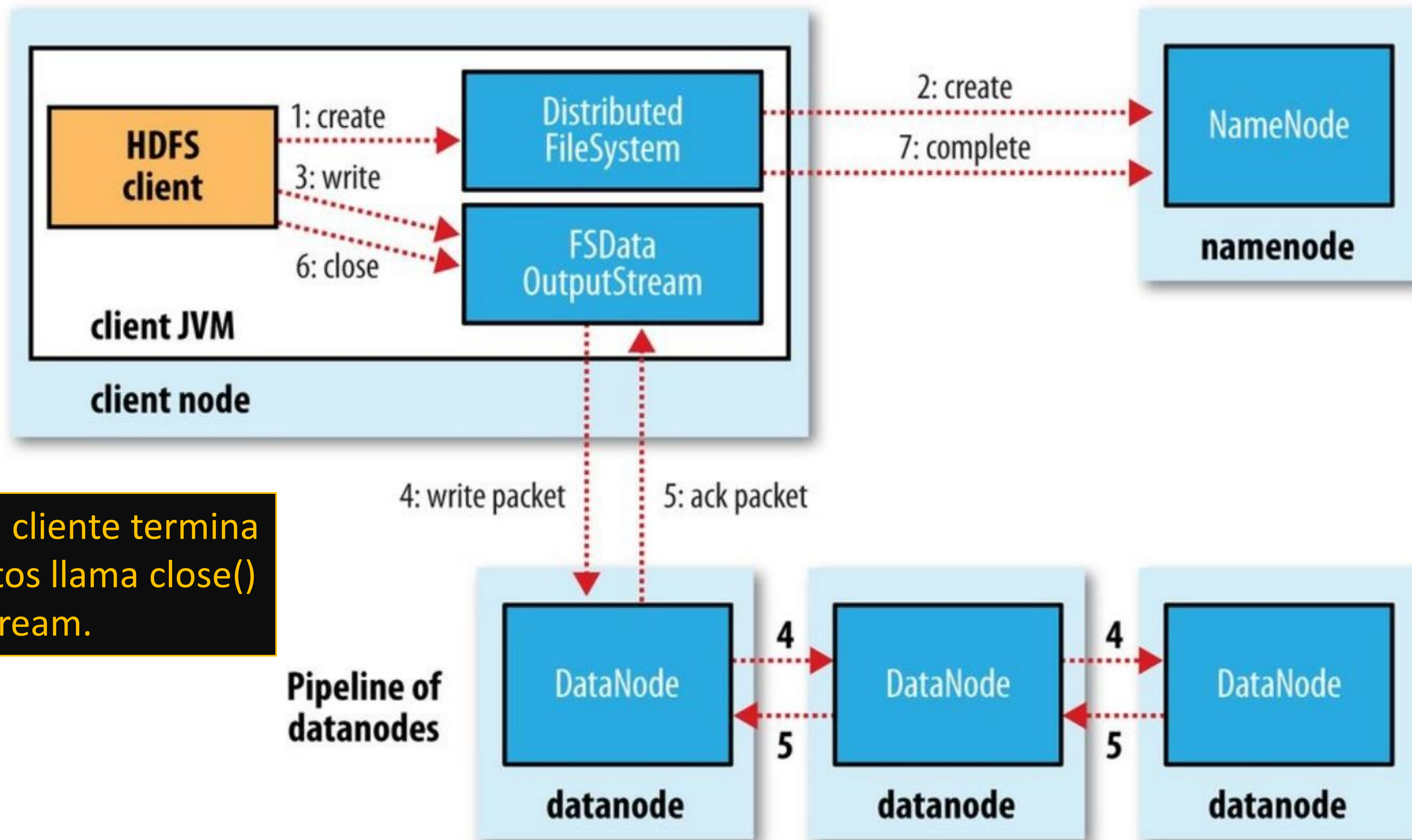
Escritura de archivo (paso 5)

- Si cualquier nodo de datos falla mientras se escriben datos en él, entonces se toman las siguientes acciones:

  5) El namenode advierte que el bloque está sub-replicado, y ordena la creación de otra réplica en otro nodo. Los bloques subsiguientes se tratan como normales. Siempre se escriben una cantidad de réplicas dfs.namenode.replication.min (que por defecto es 1), la escritura tendrá éxito y el bloque se replicará de manera asincrónica en el clúster hasta que se alcance su factor de replicación objetivo (dfs.replication, que por defecto es 3).

HDFS: Flujo de datos

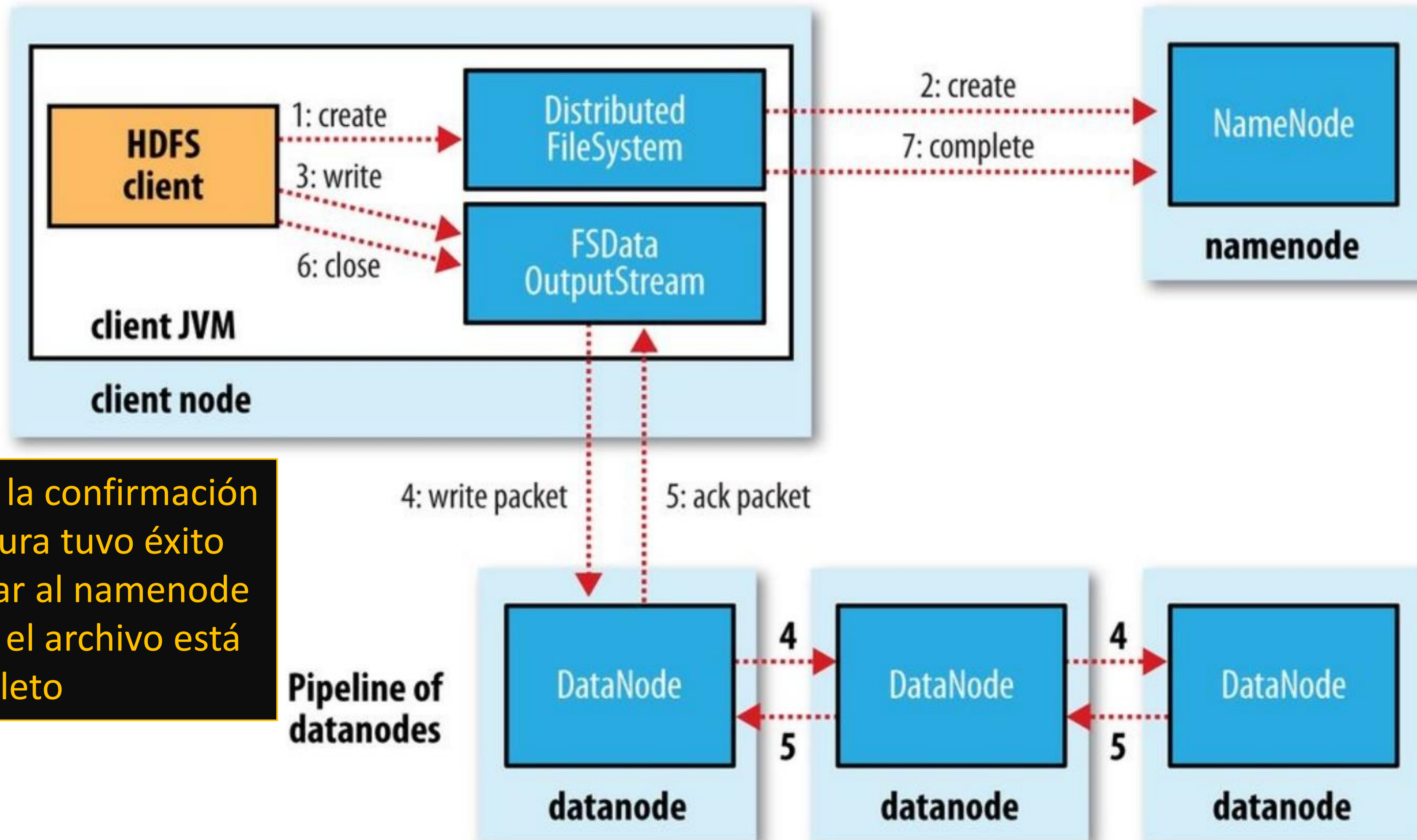
Escritura de archivo



Paso 6: Cuando el cliente termina de escribir los datos llama close() en el stream.

HDFS: Flujo de datos

Escritura de archivo



Paso 7: Se espera la confirmación de que la escritura tuvo éxito antes de contactar al namenode para indicar que el archivo está completo

HDFS: Serialización

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- **Serialización**: Proceso de convertir objetos estructurados en una secuencia de bytes para transmitirlos a través de una red o para escribirlos en un almacenamiento persistente. La **deserialización** es el proceso inverso de convertir una secuencia de bytes en una serie de objetos estructurados.
- Se utiliza en dos áreas distintas de procesamiento de datos distribuidos:
  - La comunicación entre procesos
  - El almacenamiento persistente.
- En Hadoop, la comunicación entre procesos entre nodos en el sistema se implementa mediante llamadas de procedimiento remoto (RPC). El protocolo RPC utiliza la serialización para convertir el mensaje en una secuencia binaria que se enviará al nodo remoto, que luego deserializa la secuencia binaria en el mensaje original.

HDFS: Serialización

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Es deseable que un formato de serialización RPC sea:
  - **Compacto**: Mejor uso del ancho de banda de la red.
  - **Rápido**: Es esencial que exista la menor sobrecarga de rendimiento posible para el proceso de serialización y deserialización.
  - **Extensible**: Los protocolos cambian con el tiempo para cumplir con los nuevos requisitos, por lo que debe ser sencillo desarrollar el protocolo de forma controlada para los clientes y servidores. Por ejemplo, debería ser posible agregar un nuevo argumento a una llamada de método y hacer que los nuevos servidores acepten mensajes en el formato antiguo (sin el nuevo argumento) de clientes anteriores.
  - **Interoperable**: Para algunos sistemas, es deseable poder admitir clientes que están escritos en diferentes lenguajes para el servidor, por lo que el formato debe diseñarse para que esto sea posible.

HDFS: Serialización

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Las cuatro propiedades deseables del formato de serialización de RPC también son cruciales para un formato de almacenamiento persistente:
  - **Compacto**: Para hacer un uso eficiente del espacio de almacenamiento.
  - **Rápido**: Para que la sobrecarga en la lectura o escritura de terabytes de datos sea mínima.
  - **Extensible**: Para que poder leer datos escritos de forma transparente en un formato anterior.
  - **Interoperable**: Para poder leer o escribir datos persistentes utilizando diferentes lenguajes.

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Writables

- Formato de serialización de Hadoop, que es compacto y rápido, pero no tan fácil de extender o usar desde otros lenguajes además de Java.
- Los Writables son fundamentales para Hadoop, la mayoría de los programas MapReduce los usan para sus tipos de clave y valor

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Interfaz Writables

- Dos métodos:
  - write() para escribir su estado en un *stream* binario DataOutput.
  - readFields() para leer su estado de un *stream* binario DataInput.

```
package org.apache.hadoop.io;

import java.io.DataOutput;
import java.io.DataInput;
import java.io.IOException;

public interface Writable {
  void write(DataOutput out) throws IOException;
  void readFields(DataInput in) throws IOException;
}
```

HDFS: **caché distribuido**

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Distributed cache

- Servicio de HDFS para almacenar datos a ser utilizados en el framework Map-Reduce.
- Puede almacenar archivos de solo lectura: texto, archivos comprimidos (gzip, tar, etc.), jars, etc.
- Usado para almacenar pequeños archivos, datos compartidos.
- Los archivos definidos en el distributed cache de un job se copian a todos los nodos esclavos por parte del Name Node, antes de ejecutar el job.
- Permite evitar operaciones de I/O desde nodos esclavos a HDFS.
- La existencia de los archivos en el distributed cache solo se garantiza durante la ejecución del job.

HDFS: **caché distribuido**

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Distributed cache

- El job debe especificar una dirección disponible y accesible mediante URLs (hdfs:// o http://) a través del JobConf.
- El distributed cache asume que los archivos especificados están presentes en el filesystem (HDFS) y son accessibles desde cada máquina del cluster Hadoop.
- Al indicar la inclusión en el distributed cache, un archivo se copia automáticamente a todos los nodos del cluster Hadoop a ser utilizados para ejecutar el job.
- El distributed cache monitorea los timestamps de modificación de los archivos y notifica que no deben modificarse mientras el job esté en ejecución.
- El tamaño por defecto del distributed cache es de 10GB. Puede modificarse mediante el parámetro local.cache.size en hdfs-sit.xml

HDFS: **caché distribuido**

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Distributed cache

- Procedimiento:
  - Copiar el archivo a HDFS:

```
$ hdfs dfs-put/user/dataflair/lib/jar_file.jar
```

  - Agregar el distributed cache en el JobConf de la aplicación:

```
DistributedCache.addFileToClasspath(new Path
      (“/user/dataflair/lib/jar-file.jar”),conf)
```

  - Usarlo en map y/o reduce.

HDFS: **caché distribuido**

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Distributed cache

Copiar archivos

```
$ bin/hadoop fs -copyFromLocal lookup.dat /myapp/lookup.dat
$ bin/hadoop fs -copyFromLocal mylib.jar /myapp/mylib.jar
$ bin/hadoop fs -copyFromLocal mytar.tar /myapp/mytar.tar
$ bin/hadoop fs -copyFromLocal mytargz.tar.gz /myapp/mytargz.tar.gz
```

Agregar a distributed cache

```
JobConf job = new JobConf();
DistributedCache.addCacheFile(new URI("/myapp/lookup.dat#lookup.dat"),job);
DistributedCache.addFileToClassPath(new Path("/myapp/mylib.jar"), job);
DistributedCache.addCacheArchive(new URI("/myapp/mytar.tar", job);
DistributedCache.addCacheArchive(new URI("/myapp/mytargz.tar.gz", job);
```

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Distributed cache

Usarlo en map y/o reduce

```
public static class MapClass extends MapReduceBase
            implements Mapper<K, V, K, V> {

    protected void setup(Context, context) throws IOException {

        // obtener los archivos
        Path[] files = distributedCache.getLocalCacheFiles(context.getConfiguration());

        // leer los archivos
        for (Path p : files) {
          BufferedReader rdr = new BufferedReader(new InputStreamReader(... p.toString()))
        }

          // usar datos de los archivos cacheados (rdr)
          // ...
          // ...
       }
     }
```

APACHE
HBASE

HBase

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- HBase es una base de datos distribuida no relacional de código abierto, desarrollada por la fundación Apache Software.
- Es una implementación Java de BigTable de Google.
- Puede almacenar volúmenes de datos del orden de terabytes a petabytes.
- Diseñado para operaciones de baja latencia.
- Particionamiento (sharding) automático de las tablas
- Tolerancia a fallas
- Lecturas y escrituras

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Modelo de datos

- **Fila**: pareja clave/valor, la clave es única por cada fila.
- **Columna**: claves dentro del valor de cada fila.
- **Familia de columnas**: agrupación de columnas. Se definen cuando se crea el esquema y luego se le puede agregar más columnas.
- **Celdas**: valores para cada columna con un timestamp.

Tabla persona					
Clave de la fila	**Datos personales**		**Datos demográficos**		……
Id. Persona	**Nombre**	**País**	**Fecha de Nacimiento**	**Sexo**	…
1	H. Houdini	Budapest, Hungary	1926-10-31	M	
2	M. Bosh	New Jersey, USA	1956-09-16	F	…..
…	….	….	…	…	…
500.000.000	F. Cadillac	Nevada, Usa	1964-01-07	M	….

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Modelo de datos

Cada fila tiene una clave

Cada fila es dividida en familias de columnas

Tabla persona

Clave de la fila	**Datos personales**		**Datos demográficos**		……
Id. Persona	**Nombre**	**País**	**Fecha de Nacimiento**	**Sexo**	…
1	H. Houdini	Budapest, Hungary	1926-10-31	M	
2	M. Bosh	New Jersey, USA	1956-09-16	F	…..
…	….	….	…	…	…
500.000.000	F. Cadillac	Nevada, Usa	1964-01-07	M	….

Cada familia de columnas consiste en una o más columnas

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Datos dispersos

- Las aplicaciones de big data suelen almacenar contenido variable.
  - Ejemplo: una base de datos para almacenar imágenes de satélites. Los metadatos almacenados asociados a las imágenes pueden incluir la dirección de la calle de la imagen o solo la latitud y longitud si la imagen se captura desde el desierto

Tabla imagen

Clave	**Datos nivel 1**			**Ubicación**		… …
Id. Img	**scale=1**	**scale=2**	**scale=4**	**Coordenadas**	**Dirección**	…
1	640x640	640x640	NULL	37.6735925, -1.696835705	NULL	
2	2048x2048	2048x2048	512x512	-34.91776, -56.165808500000026	Reissig esq Itapúa	…..
3	640x640	640x640	512x512	NULL	Cuzco 1232	
….	…	…	..	…….	……	…...

HBase

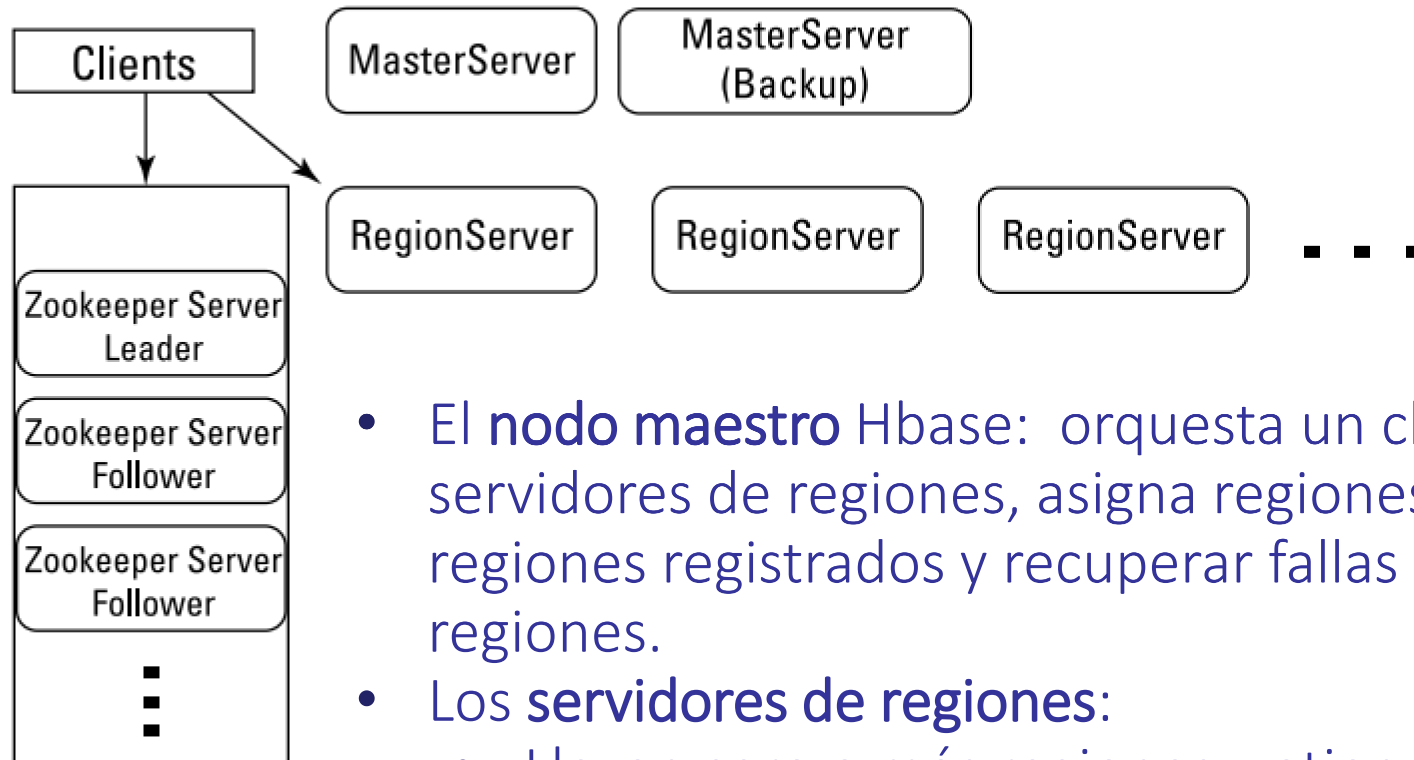
UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Se puede omitir campos sin costo alguno cuando no se tienen los datos.
- Cada valor puede tener **múltiples versiones**. Por defecto, las versiones de datos se implementan con una marca de tiempo
- Almacén de datos **distribuido y persistente**. Por defecto, HBase usa HDFS para persistir sus datos en el almacenamiento en disco.

HBase

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Arquitectura



- El **nodo maestro** Hbase: orquesta un clúster de uno o más servidores de regiones, asigna regiones a servidores de regiones registrados y recuperar fallas de servidores de regiones.
- Los **servidores de regiones**:
  - Llevan cero o más regiones y atienden las solicitudes de lectura/escritura del cliente.
  - Administran divisiones regionales, informando al maestro de HBase sobre las nuevas regiones hijas.

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

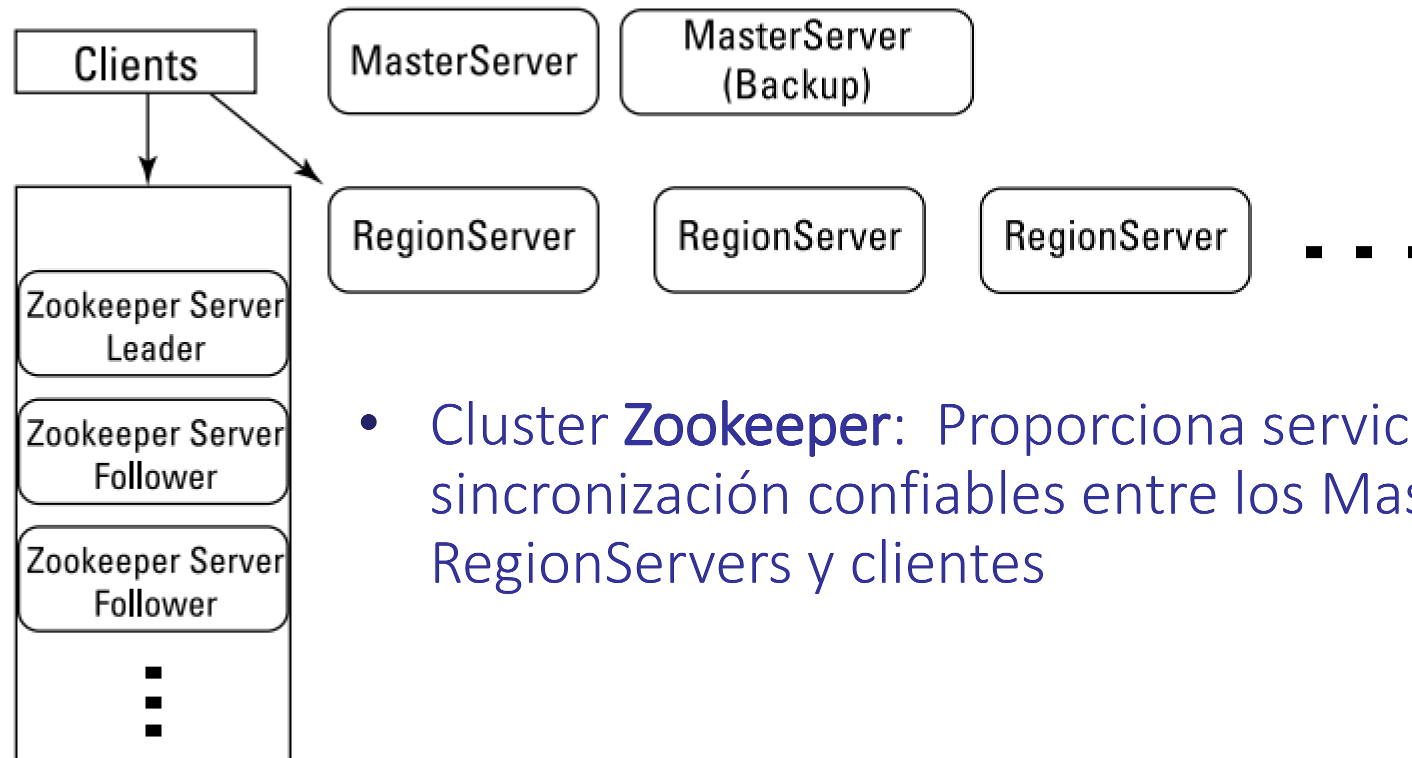
Arquitectura

RegionServer

- Procesos que se usan para **almacenar y recuperar datos** en HBase.
- En entornos de producción, cada RegionServer se implementa en su propio nodo de cómputo dedicado.
- HBase divide automáticamente las tablas que crecen y distribuye la carga entre los RegionServers.
- Auto-sharding: Hbase escala automaticamente a medida que se crece la cantidad de datos del sistema.

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Arquitectura

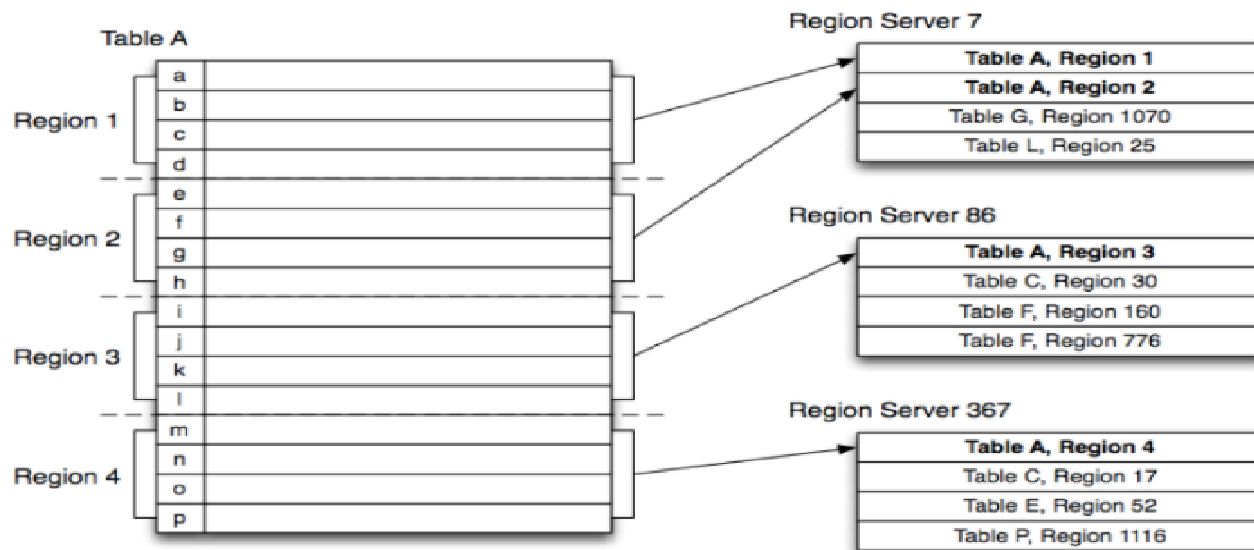


- Cluster **Zookeeper**: Proporciona servicios de coordinación y sincronización confiables entre los MasterServers, RegionServers y clientes

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Regiones

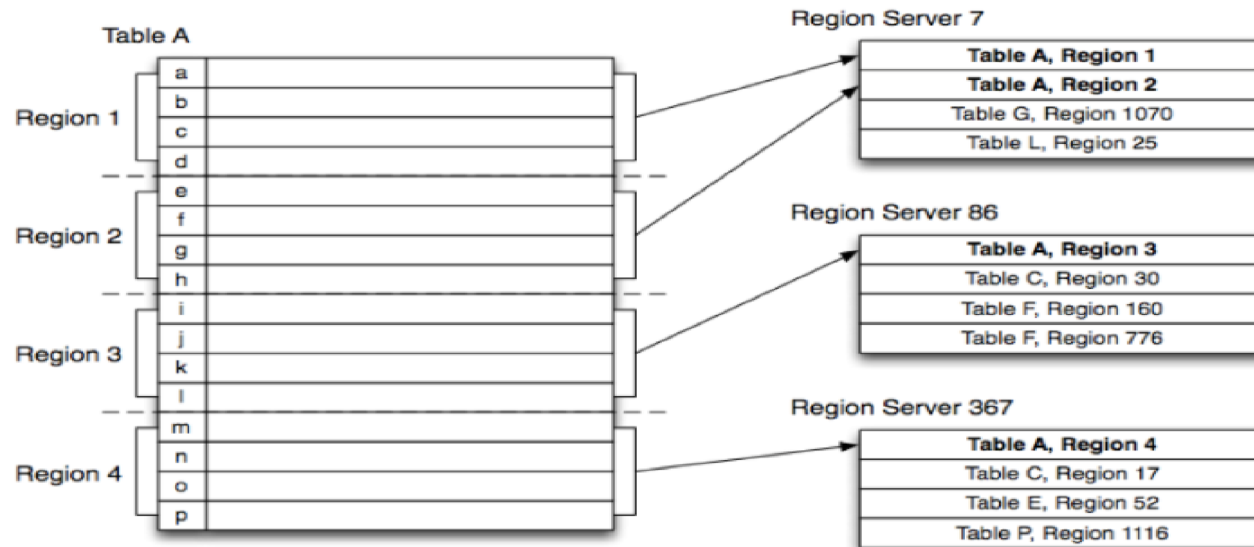
- Cada región se asigna a un RegionServer.
- Cada región comprende un subconjunto de las filas de una tabla.
- Una región se denota por la tabla a la que pertenece, su primera fila (inclusive) y su última fila (exclusiva).



UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Regiones

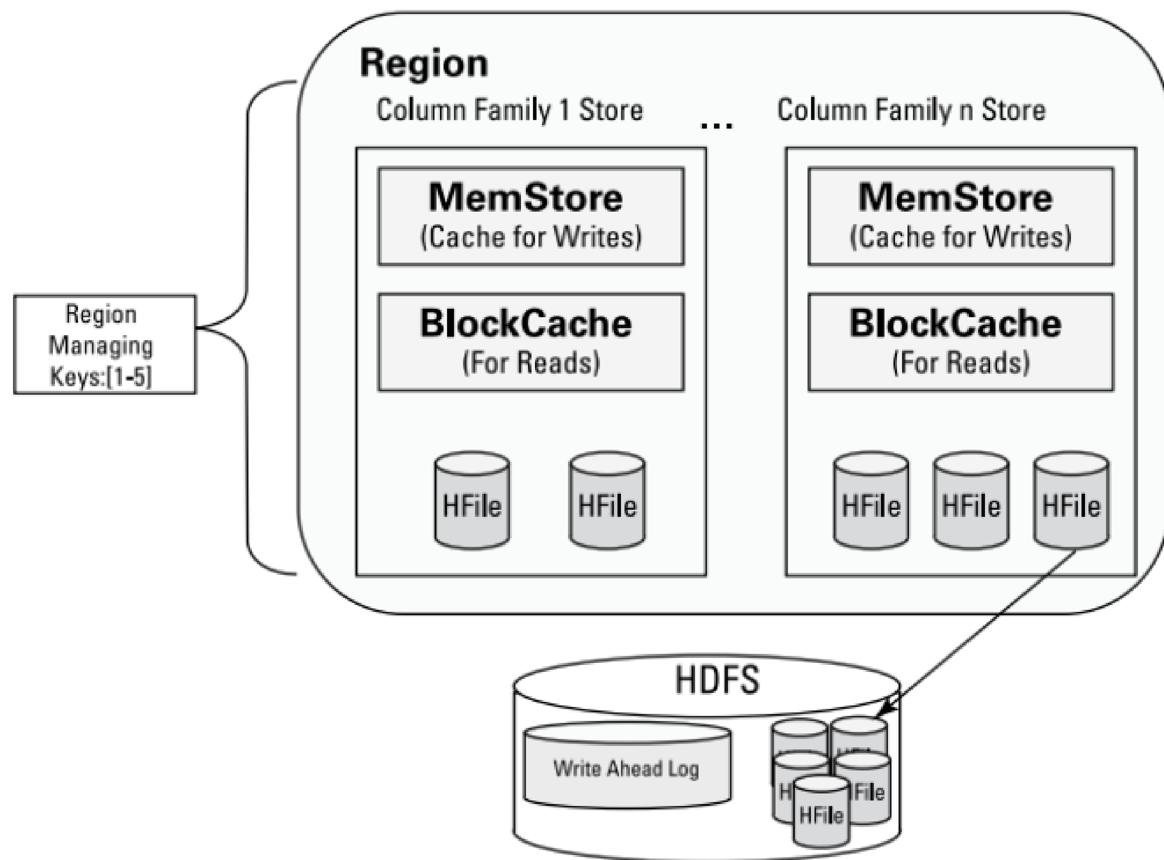
- Cuando una tabla crece y cruza un umbral de tamaño configurable se divide en dos regiones nuevas de tamaño aproximadamente igual y se distribuyen en un clúster HBase.



UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Regiones

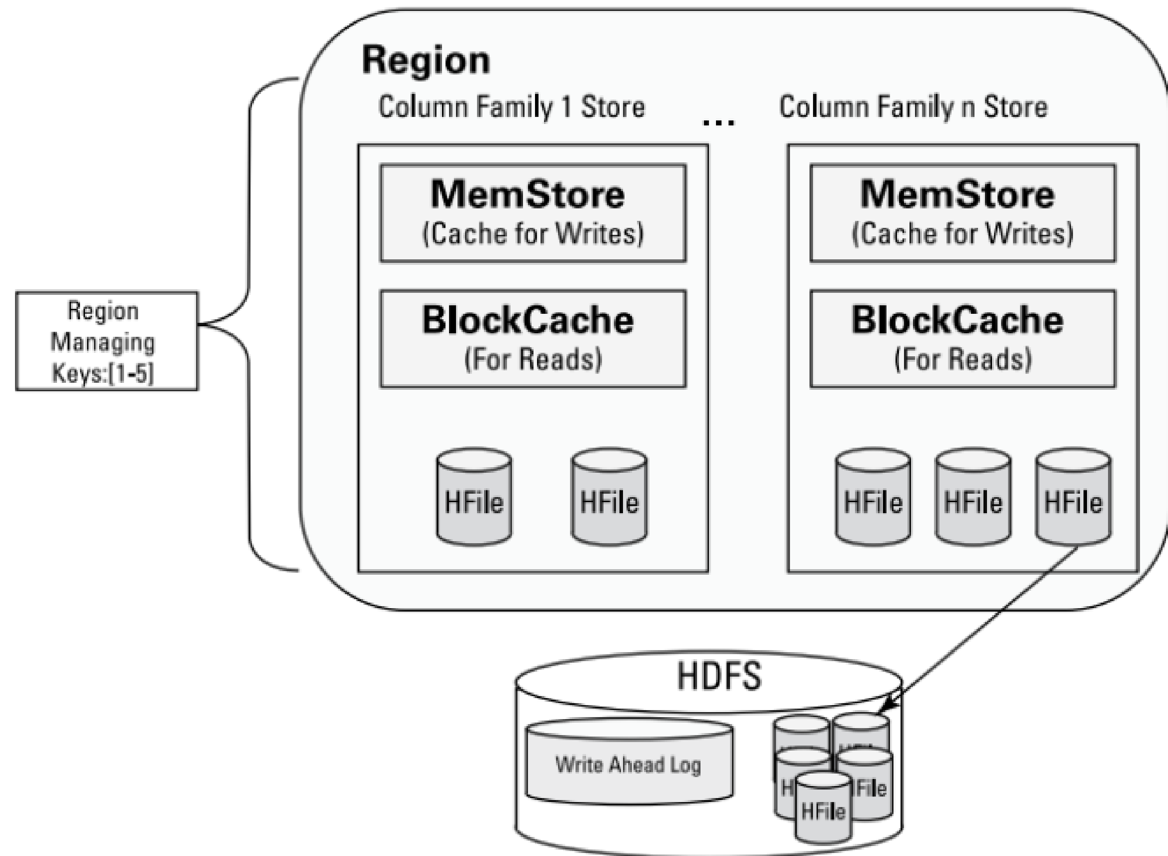
- Las regiones separan los datos en familias de columnas y almacenan los datos en HDFS utilizando objetos **Hfile**.
- Cada objeto de la familia de columnas tiene un caché de lectura llamado **BlockCache** y un caché de escritura llamado **MemStore**.



UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Regiones

- Los datos se leen en bloques desde el HDFS y se almacenan en **BlockCache**.
- El registro de escritura anticipada **WAL (Write Ahead Log)** asegura que las escrituras en HBase sean confiables. Hay un WAL por RegionServer.

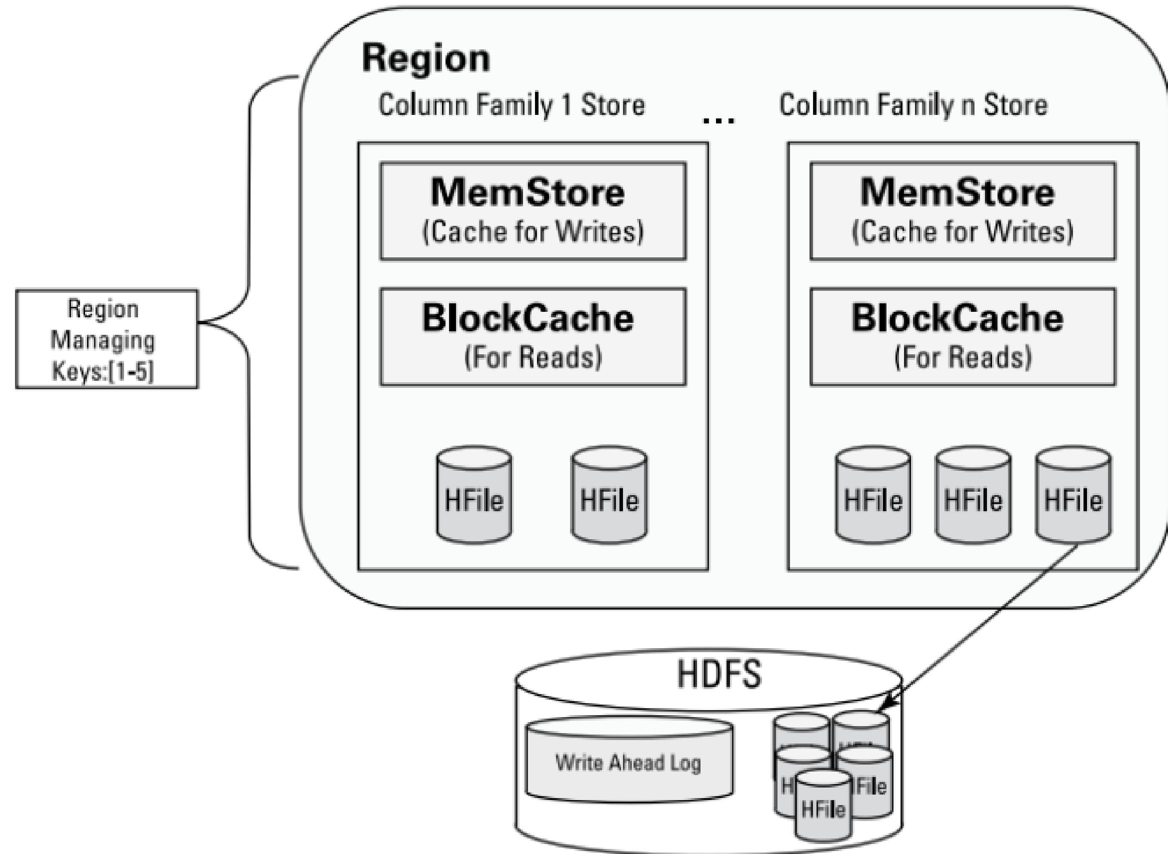


- Cuando se escribe o modifica datos en HBase, los datos primero se persisten en el WAL, que se almacena en el HDFS, y luego los datos se escriben en el caché de MemStore.

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Regiones

- Cuando se escribe o modifican datos en Hbase:
  a. Se persisten en el WAL.
  b. Se almacena en el HDFS
  c. Se escriben en el caché de MemStore

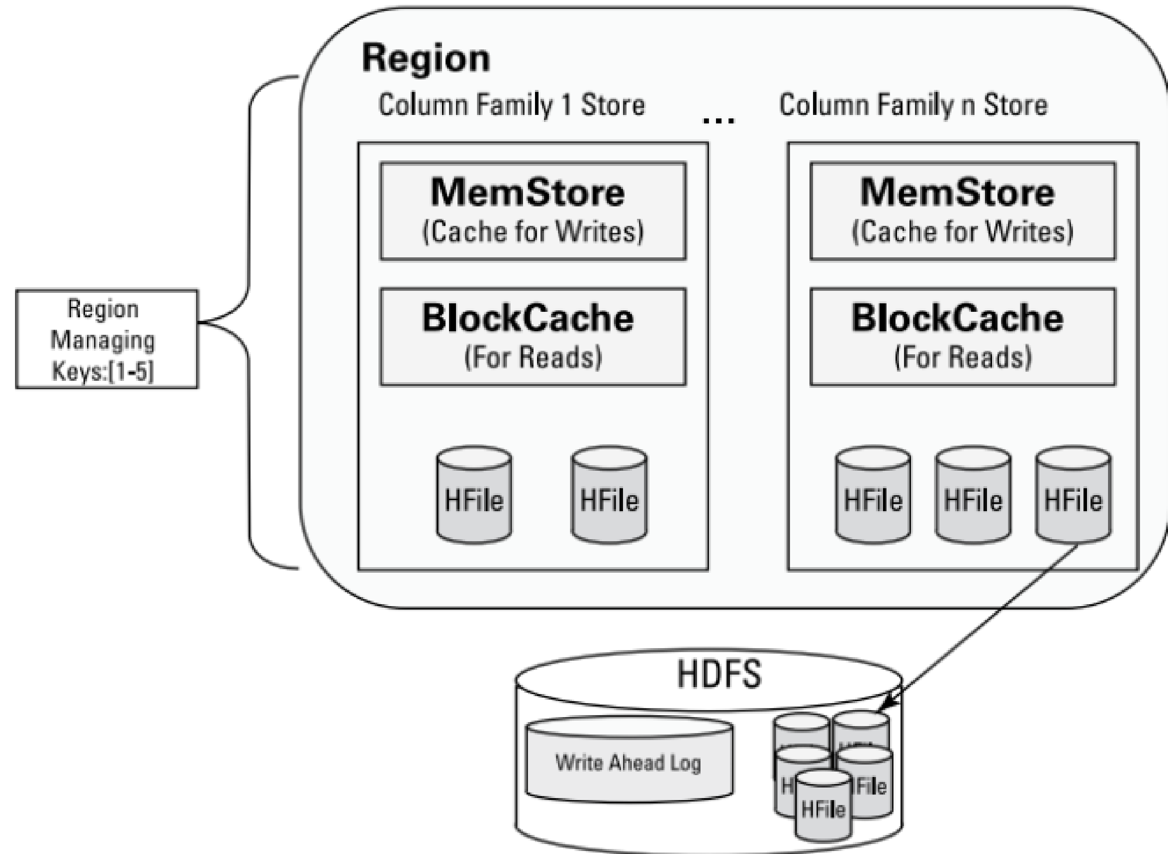


- En intervalos configurables, los pares clave-valor almacenados en MemStore se escriben en HFiles en el HDFS y luego se borran las entradas WAL

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Regiones

- Si el RegionServer tiene un error antes de que MemStore se vacíe, el WAL asegura que los cambios en los datos puedan reproducirse. Si falla la escritura en el WAL, la operación completa para modificar los datos falla.



UNIVERSIDAD DE LA REPÚBLICA URUGUAY

MasterServer

- Los RegionServers dependen del MasterServer para ciertas funciones, pero no tienen una relación de maestro-esclavo en el caso del amacenamiento y recuperación de datos.

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

MasterServer

- Principales funciones del MasterServer:
  - Monitorea los RegionServers en el cluster Hbase.
  - Se encarga de las operaciones de metadata.

    Cuando se crea una tabla o se alteran sus atributos (configuración de compresión, configuración de caché, control de versiones y más) MasterServer maneja la operación y almacena los metadatos requeridos.
  - Asigna regiones a los RegionServers .

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

MasterServer

- Principales funciones del MasterServer:
  - Supervisa el equilibrio de carga de las regiones en todos los RegionServers disponibles.
  - Inicia las operaciones de conmutación por error y restauración cuando falla algún RegionServers.

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Ejemplo en Hbase shell

- Para administrar la instancia de HBase lanzar el shell ejecutando:

```
hbase shell
```

- Para crear una tabla, se le debe asignar un nombre y definir su esquema.

```
create 'test', 'data'
```

- Se insertan datos agregando tres columnas.

```
put 'test', 'row1', 'data:1', 'value1'
put 'test', 'row2', 'data:2', 'value2'
put 'test', 'row3', 'data:3', 'value3'
```

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Ejemplo en Hbase shell

- Para administrar la instancia de HBase lanzar el shell ejecutando:

```
hbase shell
```

- Para crear una tabla, se le debe asignar un nombre y definir su esquema.

```
create 'test', 'data'
```

- Se insertan datos agregando tres columnas.

```
put 'test', 'row1', 'data:1', 'value1'
put 'test', 'row2', 'data:2', 'value2'
put 'test', 'row3', 'data:3', 'value3'
```

test			
	data		
	1	2	3
row 1	value1		
row 2		value2	
row 3			value3

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Ejemplo en Hbase shell

- Se obtiene la primera fila

```
get 'test', 'row1'
COLUMN                CELL
 data:1               timestamp=1524011888184, value=value1
1 row(s) in 0.0240 seconds
```

- Se lista el contenido de la tabla

```
scan 'test'
ROW                   COLUMN+CELL
 row1                 column=data:1, timestamp=1524011888184, value=value1
 row2                 column=data:2, timestamp=1524011897915, value=value2
 row3                 column=data:3, timestamp=1524011908166, value=value3
3 row(s) in 0.0850 seconds
```

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Ejemplo en Hbase shell

- Para borrar la tabla primero hay que deshabilitarla:

```
disable 'test'
```

- Se borra la tabla.

```
drop 'test'
```

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Ejemplo Java

```
public class ExampleClient {

  public static void main(String[] args) throws IOException {
    Configuration config = HBaseConfiguration.create();
    // Create table
    HBaseAdmin admin = new HBaseAdmin(config);
    try {
      TableName tableName = TableName.valueOf("test");
      HTableDescriptor htd = new HTableDescriptor(tableName);
      HColumnDescriptor hcd = new HColumnDescriptor("data");
      htd.addFamily(hcd);
      admin.createTable(htd);
      HTableDescriptor[] tables = admin.listTables();
      if (tables.length != 1 &&
          Bytes.equals(tableName.getName(), tables[0].getTableName().getName())) {
        throw new IOException("Failed create of table");
      }
```

HBaseConfiguration lee la configuración desde hbase-site.xml y hbase-default.xml

Se crea la tabla ‘test’ con la familia de columnas ‘data’

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Ejemplo Java (continuación)

```
// Run some operations—three puts, a get, and a scan—against the table.
HTable table = new HTable(config, tableName);
try {
  for (int i = 1; i <= 3; i++) {
    byte[] row = Bytes.toBytes("row" + i);
    Put put = new Put(row);
    byte[] columnFamily = Bytes.toBytes("data");
    byte[] qualifier = Bytes.toBytes(String.valueOf(i));
    byte[] value = Bytes.toBytes("value" + i);
    put.add(columnFamily, qualifier, value);
    table.put(put);
  }
```

Para operar con tablas se utiliza la clase Htable, con la configuración leída y el nombre de la tabla

Usando el objeto del tipo Put se inserta:
put 'test', 'row1', 'data:1', 'value1'
put 'test', 'row2', 'data:2', 'value2'
put 'test', 'row3', 'data:3', 'value3'

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Ejemplo Java (continuación)

```
Get get = new Get(Bytes.toBytes("row1"));
Result result = table.get(get);
System.out.println("Get: " + result);
Scan scan = new Scan();
ResultScanner scanner = table.getScanner(scan);
try {
  for (Result scannerResult : scanner) {
    System.out.println("Scan: " + scannerResult);
  }
} finally {
  scanner.close();
}
// Disable then drop the table
admin.disableTable(tableName);
admin.deleteTable(tableName);
```

Se ejecutan los comandos:

get 'test', 'row1'
scan 'test'
disable 'test‘
drop 'test'

- El ejemplo se puede correr en el proyecto del libro de la siguiente manera:

```
export HBASE_CLASSPATH=hbase-examples.jar
hbase ExampleClient
```

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Ejemplo Java (MapReduce con TableInputFormat)

- Ejecutar una tarea de map para contar filas utilizando TableInputFormat.
- Las clases y utilitarios de HBase en el paquete org.apache.hadoop.hbase.mapreduce facilitan el uso de HBase como fuente o destino de datos en tareas MapReduce.
- La clase TableInputFormat crea splits en límites de región para que los maps trabajen con datos en una única región.
- La clase TableOutputFormat escribe los resultados del reduce en Hbase.

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Ejemplo Java (MapReduce con TableInputFormat)

Ejecuta una tarea de map para contar filas utilizando TableInputFormat.

```
public class SimpleRowCounter extends Configured implements Tool {

  static class RowCounterMapper extends TableMapper<ImmutableBytesWritable, Result> {
    public static enum Counters { ROWS }

    @Override
    public void map(ImmutableBytesWritable row, Result value, Context context) {
      context.getCounter(Counters.ROWS).increment(1);
    }
  }
```

InmutableBytesWritable: claves de filas.
Result: filas resultado de un comando scan.

- La clase anidada RowCounterMapper es una subclase de la clase abstracta de HBase TableMapper, una especialización de org.apache.hadoop.mapreduce.Mapper que establece los tipos de entrada del map pasados por TableInputFormat.
- Este trabajo no emite salida de map, solo incrementa un Contador (Counters.ROWS) por cada fila.

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Ejemplo Java (MapReduce con TableInputFormat)

```
@Override
public int run(String[] args) throws Exception {
  if (args.length != 1) {
    System.err.println("Usage: SimpleRowCounter <tablename>");
    return -1;
  }
  String tableName = args[0];
  Scan scan = new Scan();
  scan.setFilter(new FirstKeyOnlyFilter());

  Job job = new Job(getConf(), getClass().getSimpleName());
  job.setJarByClass(getClass());
  TableMapReduceUtil.initTableMapperJob(tableName, scan,
      RowCounterMapper.class, ImmutableBytesWritable.class, Result.class, job);
  job.setNumReduceTasks(0);
  job.setOutputFormatClass(NullOutputFormat.class);
  return job.waitForCompletion(true) ? 0 : 1;
}

public static void main(String[] args) throws Exception {
  int exitCode = ToolRunner.run(HBaseConfiguration.create(),
      new SimpleRowCounter(), args);
  System.exit(exitCode);
}
}
```

En objeto scan se usa para configurar el trabajo invocando el método de utilidad TableMapReduceUtil.initTableMapJob(), que establece la clase de map que se usará y el formato de entrada a TableInputFormat.

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Ejemplo Java (MapReduce con TableInputFormat)

```
@Override
public int run(String[] args) throws Exception {
  if (args.length != 1) {
    System.err.println("Usage: SimpleRowCounter <tablename>");
    return -1;
  }
  String tableName = args[0];
  Scan scan = new Scan();
  scan.setFilter(new FirstKeyOnlyFilter());

  Job job = new Job(getConf(), getClass().getSimpleName());
  job.setJarByClass(getClass());
  TableMapReduceUtil.initTableMapperJob(tableName, scan,
      RowCounterMapper.class, ImmutableBytesWritable.class, Result.class, job);
  job.setNumReduceTasks(0);
  job.setOutputFormatClass(NullOutputFormat.class);
  return job.waitForCompletion(true) ? 0 : 1;
}

public static void main(String[] args) throws Exception {
  int exitCode = ToolRunner.run(HBaseConfiguration.create(),
      new SimpleRowCounter(), args);
  System.exit(exitCode);
}
}
```

Se setea un filtro para que se cargue el objeto Result solamente con el primer campo en cada fila (optimización útil porque el mapper no toma en cuenta los valores).

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Ejemplo Java (MapReduce con TableInputFormat)

- El número de filas de una tabla puede calcularse en el shell de Hbase con el commando 'tablename', pero no es distribuido. La version Mapreduce puede ser más eficiente para tablas de gran dimension.

- El ejemplo se puede correr en el proyecto del libro de la siguiente manera:

```
export HBASE_CLASSPATH=hbase-examples.jar
hbase SimpleRowCounter '<nombre de tabla>'
```

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Ejemplo: Aplicación de consultas online

- Interfaz simple en línea (en lugar de lote) que permite al usuario navegar por las diferentes estaciones y recorrer sus observaciones históricas de temperatura en orden cronológico.
- Se desarrollarán aplicaciones Java de línea de comandos simples para esto.
- El ejemplo está diseñado para que el conjunto de datos sea masivo, que las observaciones lleguen a miles de millones, y que la velocidad a la que llegan las actualizaciones de temperatura sea significativa, cientos o miles de actualizaciones por segundo en todo el mundo y en toda la gama de estaciones meteorológicas.

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Ejemplo: Aplicación de consultas online

- La aplicación en línea debe mostrar la observación más actualizada en un segundo o más después de la recepción.
- El requisito de tamaño debería excluir el uso de una instancia simple de RDBMS y convertir a HBase en una base de datos candidata.
- El requisito de latencia descarta HDFS sin formato. Un trabajo de MapReduce podría construir índices iniciales que permitieran el acceso aleatorio sobre todos los datos de observación, pero mantener este índice a medida que llegan las actualizaciones no es para lo que HDFS y MapReduce son buenos.

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Ejemplo: Aplicación de consultas online

- Se construirán dos tablas en el ejemplo:
  - estaciones: contiene datos de las estaciones. la clave de cada fila es el identificador de estación. La tabla tiene una familia de columnas que actúa como un diccionario de clave-valor para la información de la estación. Las claves del diccionario son los nombres de las columnasinfo:name, info:location, y info:description. Esta tabla es estática y, en este caso, la familia de información refleja fielmente un diseño de tabla RDBMS típico.
  - observaciones: esta tabla contiene observaciones de temperatura. La clave de cada fila es una clave compuesta de identificador de estación más una marca de tiempo de orden inversa. La tabla tiene una familia de columnas que contendrán una columna, airtemp, con la temperatura observada como el valor de la columna.

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Ejemplo: Aplicación de consultas online

- La elección del esquema se deriva de conocer la forma más eficiente en la que se puede leer de HBase. Las filas y las columnas se almacenan en orden lexicográfico creciente. Aunque hay instalaciones para la indexación secundaria y la coincidencia de expresión regular, tienen una penalización de rendimiento.
- Para la tabla de estaciones, la elección del identificador de estación como clave es porque siempre se accederá a la información de una estación en particular por su ID.

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Ejemplo: Aplicación de consultas online

- La tabla de observaciones usa una clave compuesta que agrega la marca de tiempo de observación al final. Esto agrupará todas las observaciones para una estación particular juntas, y usando una marca de tiempo de orden inverso (Long.MAX_VALUE - timestamp) y almacenándola como binaria, las observaciones para cada estación se ordenarán primero con la observación más reciente.

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Ejemplo: Aplicación de consultas online (carga de datos)

- Para definir las tablas se ejecuta en la shell de HBase:

```
create 'stations', {NAME => 'info'}
create 'observations', {NAME => 'data'}
```

- Se cargan algunas estaciones de prueba en la tabla stations usando el ejemplo del libro:

```
export HADOOP_CLASSPATH=hbase-examples.jar
hbase HBaseStationImporter input/ncdc/metadata/stations-fixed-width.txt
```

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Ejemplo: Aplicación de consultas online (carga de datos)

- MapReduce para importar temperaturas desde HDFS hacia Hbase
- HBaseTemperatureImporter implementa la clase Tool y realiza la configuración para iniciar el trabajo que consiste solamente en un map.

```
public class HBaseTemperatureImporter extends Configured implements Tool {

  static class HBaseTemperatureMapper<K> extends Mapper<LongWritable, Text, K, Put> {
    private NcdcRecordParser parser = new NcdcRecordParser();

    @Override
    public void map(LongWritable key, Text value, Context context) throws
        IOException, InterruptedException {
      parser.parse(value.toString());
      if (parser.isValidTemperature()) {
        byte[] rowKey = RowKeyConverter.makeObservationRowKey(parser.getStationId(),
            parser.getObservationDate().getTime());
        Put p = new Put(rowKey);
        p.add(HBaseTemperatureQuery.DATA_COLUMNFAMILY,
              HBaseTemperatureQuery.AIRTEMP_QUALIFIER,
              Bytes.toBytes(parser.getAirTemperature()));
         context.write(null, p);
      }
    }
  }
```

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Ejemplo: Aplicación de consultas online (carga de datos)

- Implementación del map() en la clase HBaseTemperatureMapper anidada en HBaseTemperatureImporter

```
public class HBaseTemperatureImporter extends Configured implements Tool {

  static class HBaseTemperatureMapper<K> extends Mapper<LongWritable, Text, K, Put> {
    private NcdcRecordParser parser = new NcdcRecordParser();

    @Override
    public void map(LongWritable key, Text value, Context context) throws
        IOException, InterruptedException {
      parser.parse(value.toString());
      if (parser.isValidTemperature()) {
        byte[] rowKey = RowKeyConverter.makeObservationRowKey(parser.getStationId(),
            parser.getObservationDate().getTime());
        Put p = new Put(rowKey);
        p.add(HBaseTemperatureQuery.DATA_COLUMNFAMILY,
              HBaseTemperatureQuery.AIRTEMP_QUALIFIER,
              Bytes.toBytes(parser.getAirTemperature()));
          context.write(null, p);
      }
    }
  }
```

Se construye la clave usando un objeto del tipo RowKeyConverter.

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Ejemplo: Aplicación de consultas online (carga de datos)

- Clase encargada de construir la clave de las filas tomando el identificador de estación y la marca de tiempo de la medición.

```java
public class RowKeyConverter {

  private static final int STATION_ID_LENGTH = 12;

  /**
   * @return A row key whose format is: <station_id> <reverse_order_timestamp>
   */
  public static byte[] makeObservationRowKey(String stationId,
      long observationTime) {
    byte[] row = new byte[STATION_ID_LENGTH + Bytes.SIZEOF_LONG];
    Bytes.putBytes(row, 0, Bytes.toBytes(stationId), 0, STATION_ID_LENGTH);
    long reverseOrderTimestamp = Long.MAX_VALUE - observationTime;
    Bytes.putLong(row, STATION_ID_LENGTH, reverseOrderTimestamp);
    return row;
  }
}
```

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Ejemplo: Aplicación de consultas online (carga de datos)

- Clase encargada de construir la clave de las filas tomando el identificador de estación y la marca de tiempo de la medición.

```java
public class RowKeyConverter {

  private static final int STATION_ID_LENGTH = 12;

  /**
   * @return A row key whose format is: <station_id> <reverse_order_timestamp>
   */
  public static byte[] makeObservationRowKey(String stationId,
      long observationTime) {
    byte[] row = new byte[STATION_ID_LENGTH + Bytes.SIZEOF_LONG];
    Bytes.putBytes(row, 0, Bytes.toBytes(stationId), 0, STATION_ID_LENGTH);
    long reverseOrderTimestamp = Long.MAX_VALUE - observationTime;
    Bytes.putLong(row, STATION_ID_LENGTH, reverseOrderTimestamp);
    return row;
  }
}
```

El tamaño total de la fila es el tamaño del identificador de estación más el tamaño de la marca de tiempo

Ejemplo: Aplicación de consultas online (carga de datos)

- Clase encargada de construir la clave de las filas tomando el identificador de estación y la marca de tiempo de la medición.

```
public class RowKeyConverter {

  private static final int STATION_ID_LENGTH = 12;

  /**
   * @return A row key whose format is: <station_id> <reverse_order_timestamp>
   */
  public static byte[] makeObservationRowKey(String stationId,
      long observationTime) {
    byte[] row = new byte[STATION_ID_LENGTH + Bytes.SIZEOF_LONG];
    Bytes.putBytes(row, 0, Bytes.toBytes(stationId), 0, STATION_ID_LENGTH);
    long reverseOrderTimestamp = Long.MAX_VALUE - observationTime;
    Bytes.putLong(row, STATION_ID_LENGTH, reverseOrderTimestamp);
    return row;
  }
}
```

Se usa la clase Bytes de Hbase para transformar objetos Java comunes a bytes. Se coloca el identificador de estación al comienzo de la fila

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Ejemplo: Aplicación de consultas online (carga de datos)

- Clase encargada de construir la clave de las filas tomando el identificador de estación y la marca de tiempo de la medición.

```
public class RowKeyConverter {

  private static final int STATION_ID_LENGTH = 12;

  /**
   * @return A row key whose format is: <station_id> <reverse_order_timestamp>
   */
  public static byte[] makeObservationRowKey(String stationId,
      long observationTime) {
    byte[] row = new byte[STATION_ID_LENGTH + Bytes.SIZEOF_LONG];
    Bytes.putBytes(row, 0, Bytes.toBytes(stationId), 0, STATION_ID_LENGTH);
    long reverseOrderTimestamp = Long.MAX_VALUE - observationTime;
    Bytes.putLong(row, STATION_ID_LENGTH, reverseOrderTimestamp);
    return row;
  }
}
```

Se calcula el inverso de la marca del tiempo y se lo coloca al final de la cadena de bytes de la fila.

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Ejemplo: Aplicación de consultas online (carga de datos)

- Implementación del map() en la clase HBaseTemperatureMapper anidada en HBaseTemperatureImporter

```
public class HBaseTemperatureImporter extends Configured implements Tool {

  static class HBaseTemperatureMapper<K> extends Mapper<LongWritable, Text, K, Put> {
    private NcdcRecordParser parser = new NcdcRecordParser();

    @Override
    public void map(LongWritable key, Text value, Context context) throws
        IOException, InterruptedException {
      parser.parse(value.toString());
      if (parser.isValidTemperature()) {
        byte[] rowKey = RowKeyConverter.makeObservationRowKey(parser.getStationId(),
            parser.getObservationDate().getTime());
        Put p = new Put(rowKey);
        p.add(HBaseTemperatureQuery.DATA_COLUMNFAMILY,
            HBaseTemperatureQuery.AIRTEMP_QUALIFIER,
            Bytes.toBytes(parser.getAirTemperature()));
        context.write(null, p);
      }
    }
  }
```

Se usa el objeto Put para guardar en Hbase cada registro leído.

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Ejemplo: Aplicación de consultas online (carga de datos)

- Implementación del método run() de la clase HBaseTemperatureImporter

```
  @Override
  public int run(String[] args) throws Exception {
    if (args.length != 1) {
      System.err.println("Usage: HBaseTemperatureImporter <input>");
      return -1;
    }
    Job job = new Job(getConf(), getClass().getSimpleName());
    job.setJarByClass(getClass());
    FileInputFormat.addInputPath(job, new Path(args[0]));
    job.getConfiguration().set(TableOutputFormat.OUTPUT_TABLE, "observations");
    job.setMapperClass(HBaseTemperatureMapper.class);
    job.setNumReduceTasks(0);
    job.setOutputFormatClass(TableOutputFormat.class);
    return job.waitForCompletion(true) ? 0 : 1;
  }

  public static void main(String[] args) throws Exception {
    int exitCode = ToolRunner.run(HBaseConfiguration.create(),
        new HBaseTemperatureImporter(), args);
    System.exit(exitCode);
  }
}
```

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Ejemplo: Aplicación de consultas online (carga de datos)

- Implementación del método run() de la clase HBaseTemperatureImporter

```
@Override
public int run(String[] args) throws Exception {
  if (args.length != 1) {
    System.err.println("Usage: HBaseTemperatureImporter <input>");
    return -1;
  }
  Job job = new Job(getConf(), getClass().getSimpleName());
  job.setJarByClass(getClass());
  FileInputFormat.addInputPath(job, new Path(args[0]));
  job.getConfiguration().set(TableOutputFormat.OUTPUT_TABLE, "observations");
  job.setMapperClass(HBaseTemperatureMapper.class);
  job.setNumReduceTasks(0);
  job.setOutputFormatClass(TableOutputFormat.class);
  return job.waitForCompletion(true) ? 0 : 1;
}

public static void main(String[] args) throws Exception {
  int exitCode = ToolRunner.run(HBaseConfiguration.create(),
      new HBaseTemperatureImporter(), args);
  System.exit(exitCode);
}
}
```

Se indica en que tabla escribir usando la propiedad TableOutputFormat.OUTPUT_TABLE

Ejemplo: Aplicación de consultas online (carga de datos)

- Es conveniente utilizar TableOutputFormat, ya que gestiona la creación de una instancia HTable, que de lo contrario hay que hacerlo en el método setup() del mapper (junto con una llamada a close () en el método cleanup()).

- La clase se puede probar ejecutando:

```
export HADOOP_CLASSPATH=hbase-examples.jar
hbase HBaseTemperatureImporter input/ncdc/all
```

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Ejemplo: Aplicación de consultas online (consultas)

- Obtener la información de la estación estática. Esta es una búsqueda de una sola fila, realizada mediante una operación get()

```
static final byte[] INFO_COLUMNFAMILY = Bytes.toBytes("info");
static final byte[] NAME_QUALIFIER = Bytes.toBytes("name");
static final byte[] LOCATION_QUALIFIER = Bytes.toBytes("location");
static final byte[] DESCRIPTION_QUALIFIER = Bytes.toBytes("description");

public Map<String, String> getStationInfo(HTable table, String stationId)
    throws IOException {
  Get get = new Get(Bytes.toBytes(stationId));
  get.addFamily(INFO_COLUMNFAMILY);
  Result res = table.get(get);
  if (res == null) {
    return null;
  }
  Map<String, String> resultMap = new LinkedHashMap<String, String>();
  resultMap.put("name", getValue(res, INFO_COLUMNFAMILY, NAME_QUALIFIER));
  resultMap.put("location", getValue(res, INFO_COLUMNFAMILY,
      LOCATION_QUALIFIER));
  resultMap.put("description", getValue(res, INFO_COLUMNFAMILY,
      DESCRIPTION_QUALIFIER));
  return resultMap;
}

private static String getValue(Result res, byte[] cf, byte[] qualifier) {
  byte[] value = res.getValue(cf, qualifier);
  return value == null? "": Bytes.toString(value);
}
```

Se utiliza la familia de columnas ‘info’ como un diccionario de clave-valor (nombre de columnas son las claves y los valores de las columnas como el valor)

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Ejemplo: Aplicación de consultas online (consultas)

- Obtener la información de la estación estática. Esta es una búsqueda de una sola fila, realizada mediante una operación get()

```
static final byte[] INFO_COLUMNFAMILY = Bytes.toBytes("info");
static final byte[] NAME_QUALIFIER = Bytes.toBytes("name");
static final byte[] LOCATION_QUALIFIER = Bytes.toBytes("location");
static final byte[] DESCRIPTION_QUALIFIER = Bytes.toBytes("description");

public Map<String, String> getStationInfo(HTable table, String stationId)
    throws IOException {
  Get get = new Get(Bytes.toBytes(stationId));
  get.addFamily(INFO_COLUMNFAMILY);
  Result res = table.get(get);
  if (res == null) {
    return null;
  }
  Map<String, String> resultMap = new LinkedHashMap<String, String>();
  resultMap.put("name", getValue(res, INFO_COLUMNFAMILY, NAME_QUALIFIER));
  resultMap.put("location", getValue(res, INFO_COLUMNFAMILY,
      LOCATION_QUALIFIER));
  resultMap.put("description", getValue(res, INFO_COLUMNFAMILY,
      DESCRIPTION_QUALIFIER));
  return resultMap;
}

private static String getValue(Result res, byte[] cf, byte[] qualifier) {
  byte[] value = res.getValue(cf, qualifier);
  return value == null? "": Bytes.toString(value);
}
```

Se configura la instancia de la clase Get de manera que retorne todos los valores de las columnas para la fila indentificada por stationId en la familia de columnas 'info'

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Ejemplo: Aplicación de consultas online (consultas)

- Obtener la información de la estación estática. Esta es una búsqueda de una sola fila, realizada mediante una operación get()

```
static final byte[] INFO_COLUMNFAMILY = Bytes.toBytes("info");
static final byte[] NAME_QUALIFIER = Bytes.toBytes("name");
static final byte[] LOCATION_QUALIFIER = Bytes.toBytes("location");
static final byte[] DESCRIPTION_QUALIFIER = Bytes.toBytes("description");

public Map<String, String> getStationInfo(HTable table, String stationId)
    throws IOException {
  Get get = new Get(Bytes.toBytes(stationId));
  get.addFamily(INFO_COLUMNFAMILY);
  Result res = table.get(get);
  if (res == null) {
    return null;
  }
  Map<String, String> resultMap = new LinkedHashMap<String, String>();
  resultMap.put("name", getValue(res, INFO_COLUMNFAMILY, NAME_QUALIFIER));
  resultMap.put("location", getValue(res, INFO_COLUMNFAMILY,
      LOCATION_QUALIFIER));
  resultMap.put("description", getValue(res, INFO_COLUMNFAMILY,
      DESCRIPTION_QUALIFIER));
  return resultMap;
}

private static String getValue(Result res, byte[] cf, byte[] qualifier) {
  byte[] value = res.getValue(cf, qualifier);
  return value == null? "": Bytes.toString(value);
}
```

Se carga la estructura de datos de retorno recuperando los datos desde el objeto Result, especificando el nombre de la familia de columnas y el nombre de cada columna

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Ejemplo: Aplicación de consultas online (consultas)

- Una de las ventajas de HBase en una base de datos relacional es que no se tiene que especificar todas las columnas por adelantado. Entonces, si cada estación ahora tiene al menos estos tres atributos pero hay cientos de opcionales, en el futuro podemos simplemente insertarlos sin modificar el esquema.

- La clase se puede probar ejecutando:

```
export HADOOP_CLASSPATH=hbase-examples.jar
hbase HBaseStationQuery 011990-99999
```

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Ejemplo: Aplicación de consultas online (consultas)

- Dado un identificador de estación *stationId*, una a marca de tiempo de inicio *maxStamp* y la cantidad máxima de filas a retornar *maxCount* el método getStationObservations() retorna las últimas mediciones de temperatura hechas por la estación identificada por *stationId.* Dado que las filas se almacenan en orden cronológico inverso por estación, las consultas arrojarán observaciones que preceden a la marca de tiempo de inicio.

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Ejemplo: Aplicación de consultas online (consultas)

- Método getStationObservations() de la clase HBaseTemperatureQuery

```
public class HBaseTemperatureQuery extends Configured implements Tool {
  static final byte[] DATA_COLUMNFAMILY = Bytes.toBytes("data");
  static final byte[] AIRTEMP_QUALIFIER = Bytes.toBytes("airtemp");

  public NavigableMap<Long, Integer> getStationObservations(HTable table,
      String stationId, long maxStamp, int maxCount) throws IOException {
    byte[] startRow = RowKeyConverter.makeObservationRowKey(stationId, maxStamp);
    NavigableMap<Long, Integer> resultMap = new TreeMap<Long, Integer>();
    Scan scan = new Scan(startRow);
    scan.addColumn(DATA_COLUMNFAMILY, AIRTEMP_QUALIFIER);
    ResultScanner scanner = table.getScanner(scan);
    try {
      Result res;
      int count = 0;
      while ((res = scanner.next()) != null && count++ < maxCount) {
        byte[] row = res.getRow();
        byte[] value = res.getValue(DATA_COLUMNFAMILY, AIRTEMP_QUALIFIER);
        Long stamp = Long.MAX_VALUE -
            Bytes.toLong(row, row.length - Bytes.SIZEOF_LONG, Bytes.SIZEOF_LONG);
        Integer temp = Bytes.toInt(value);
        resultMap.put(stamp, temp);
      }
    } finally {
      scanner.close();
    }
    return resultMap;
  }
```

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Ejemplo: Aplicación de consultas online (consultas)

- Método getStationObservations() de la clase HBaseTemperatureQuery

```
public class HBaseTemperatureQuery extends Configured implements Tool {
  static final byte[] DATA_COLUMNFAMILY = Bytes.toBytes("data");
  static final byte[] AIRTEMP_QUALIFIER = Bytes.toBytes("airtemp");

  public NavigableMap<Long, Integer> getStationObservations(HTable table,
      String stationId, long maxStamp, int maxCount) throws IOException {
    byte[] startRow = RowKeyConverter.makeObservationRowKey(stationId, maxStamp);
    NavigableMap<Long, Integer> resultMap = new TreeMap<Long, Integer>();
    Scan scan = new Scan(startRow);
    scan.addColumn(DATA_COLUMNFAMILY, AIRTEMP_QUALIFIER);
    ResultScanner scanner = table.getScanner(scan);
    try {
      Result res;
      int count = 0;
      while ((res = scanner.next()) != null && count++ < maxCount) {
        byte[] row = res.getRow();
        byte[] value = res.getValue(DATA_COLUMNFAMILY, AIRTEMP_QUALIFIER);
        Long stamp = Long.MAX_VALUE -
            Bytes.toLong(row, row.length - Bytes.SIZEOF_LONG, Bytes.SIZEOF_LONG);
        Integer temp = Bytes.toInt(value);
        resultMap.put(stamp, temp);
      }
    } finally {
      scanner.close();
    }
    return resultMap;
  }
```

Utiliza un escáner HBase para iterar sobre las filas de la tabla. Devuelve un NavigableMap <Long, Integer>, donde la clave es la marca de tiempo y el valor es la temperatura. Dado que el mapa ordena por clave en orden ascendente, sus entradas están en orden cronológico.

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Ejemplo: Aplicación de consultas online (consultas)

- Método getStationObservations() de la clase HBaseTemperatureQuery

```
public class HBaseTemperatureQuery extends Configured implements Tool {
  static final byte[] DATA_COLUMNFAMILY = Bytes.toBytes("data");
  static final byte[] AIRTEMP_QUALIFIER = Bytes.toBytes("airtemp");

  public NavigableMap<Long, Integer> getStationObservations(HTable table,
      String stationId, long maxStamp, int maxCount) throws IOException {
    byte[] startRow = RowKeyConverter.makeObservationRowKey(stationId, maxStamp);
    NavigableMap<Long, Integer> resultMap = new TreeMap<Long, Integer>();
    Scan scan = new Scan(startRow);
    scan.addColumn(DATA_COLUMNFAMILY, AIRTEMP_QUALIFIER);
    ResultScanner scanner = table.getScanner(scan);
    try {
      Result res;
      int count = 0;
      while ((res = scanner.next()) != null && count++ < maxCount) {
        byte[] row = res.getRow();
        byte[] value = res.getValue(DATA_COLUMNFAMILY, AIRTEMP_QUALIFIER);
        Long stamp = Long.MAX_VALUE -
            Bytes.toLong(row, row.length - Bytes.SIZEOF_LONG, Bytes.SIZEOF_LONG);
        Integer temp = Bytes.toInt(value);
        resultMap.put(stamp, temp);
      }
    } finally {
      scanner.close();
    }
    return resultMap;
  }
```

Se le indica al scanner que recupere las temperaturas a partir de la marca de tiempo indicada en startRow.

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Ejemplo: Aplicación de consultas online (consultas)

- Método getStationObservations() de la clase HBaseTemperatureQuery

```
public class HBaseTemperatureQuery extends Configured implements Tool {
  static final byte[] DATA_COLUMNFAMILY = Bytes.toBytes("data");
  static final byte[] AIRTEMP_QUALIFIER = Bytes.toBytes("airtemp");

  public NavigableMap<Long, Integer> getStationObservations(HTable table,
      String stationId, long maxStamp, int maxCount) throws IOException {
    byte[] startRow = RowKeyConverter.makeObservationRowKey(stationId, maxStamp);
    NavigableMap<Long, Integer> resultMap = new TreeMap<Long, Integer>();
    Scan scan = new Scan(startRow);
    scan.addColumn(DATA_COLUMNFAMILY, AIRTEMP_QUALIFIER);
    ResultScanner scanner = table.getScanner(scan);
    try {
      Result res;
      int count = 0;
      while ((res = scanner.next()) != null && count++ < maxCount) {
        byte[] row = res.getRow();
        byte[] value = res.getValue(DATA_COLUMNFAMILY, AIRTEMP_QUALIFIER);
        Long stamp = Long.MAX_VALUE -
            Bytes.toLong(row, row.length - Bytes.SIZEOF_LONG, Bytes.SIZEOF_LONG);
        Integer temp = Bytes.toInt(value);
        resultMap.put(stamp, temp);
      }
    } finally {
      scanner.close();
    }
    return resultMap;
  }
```

Se itera en el conjunto resultado y se guardan las temperaturas ordenas por marca de tiempo hasta alcanzar la máxima cantidad de filas especificada por parámetro

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Ejemplo: Aplicación de consultas online (consultas)

- Método run() de la clase HBaseTemperatureQuery

```
public int run(String[] args) throws IOException {
  if (args.length != 1) {
    System.err.println("Usage: HBaseTemperatureQuery <station_id>");
    return -1;
  }

  HTable table = new HTable(HBaseConfiguration.create(getConf()), "observations");
  try {
    NavigableMap<Long, Integer> observations =
        getStationObservations(table, args[0], Long.MAX_VALUE, 10).descendingMap();
    for (Map.Entry<Long, Integer> observation : observations.entrySet()) {
      // Print the date, time, and temperature
      System.out.printf("%1$tF %1$tR\t%2$s\n", observation.getKey(),
          observation.getValue());
    }
    return 0;
  } finally {
    table.close();
  }
}

public static void main(String[] args) throws Exception {
  int exitCode = ToolRunner.run(HBaseConfiguration.create(),
      new HBaseTemperatureQuery(), args);
  System.exit(exitCode);
}
}
```

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Ejemplo: Aplicación de consultas online (consultas)

- Método run() de la clase HBaseTemperatureQuery

```
public int run(String[] args) throws IOException {
  if (args.length != 1) {
    System.err.println("Usage: HBaseTemperatureQuery <station_id>");
    return -1;
  }

  HTable table = new HTable(HBaseConfiguration.create(getConf()), "observations");
  try {
    NavigableMap<Long, Integer> observations =
        getStationObservations(table, args[0], Long.MAX_VALUE, 10).descendingMap();
    for (Map.Entry<Long, Integer> observation : observations.entrySet()) {
      // Print the date, time, and temperature
      System.out.printf("%1$tF %1$tR\t%2$s\n", observation.getKey(),
          observation.getValue());
    }
    return 0;
  } finally {
    table.close();
  }
}

public static void main(String[] args) throws Exception {
  int exitCode = ToolRunner.run(HBaseConfiguration.create(),
      new HBaseTemperatureQuery(), args);
  System.exit(exitCode);
}
}
```

Se instancia el objeto Htable con la configuración por defecto y se le indica que la tabla usada es "observations"

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Ejemplo: Aplicación de consultas online (consultas)

- Método run() de la clase HBaseTemperatureQuery

```
public int run(String[] args) throws IOException {
  if (args.length != 1) {
    System.err.println("Usage: HBaseTemperatureQuery <station_id>");
    return -1;
  }

  HTable table = new HTable(HBaseConfiguration.create(getConf()), "observations");
  try {
    NavigableMap<Long, Integer> observations =
        getStationObservations(table, args[0], Long.MAX_VALUE, 10).descendingMap();
    for (Map.Entry<Long, Integer> observation : observations.entrySet()) {
      // Print the date, time, and temperature
      System.out.printf("%1$tF %1$tR\t%2$s\n", observation.getKey(),
          observation.getValue());
    }
    return 0;
  } finally {
    table.close();
  }
}

public static void main(String[] args) throws Exception {
  int exitCode = ToolRunner.run(HBaseConfiguration.create(),
      new HBaseTemperatureQuery(), args);
  System.exit(exitCode);
}
}
```

Se llama al método getStationObservations con el identificador de estación pasado por parámetro, Long.MAX_VALUE en lugar de maxStamp para indicar que retorne temperaturas a partir del presente y la cantidad máxima de filas igual a 10. Se ordena la salida.

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Ejemplo: Aplicación de consultas online (consultas)

- La ventaja de almacenar marcas de tiempo en orden cronológico inverso es que nos permite obtener las observaciones más recientes.
- Si las observaciones se almacenaran con las marcas de tiempo reales, se podría obtener solo las observaciones más antiguas para un desplazamiento y límite dados de manera eficiente.
- Obtener el más nuevo significaría obtener todas las filas y luego agarrar el más nuevo del final.
- Versiones más nuevas de Hbase incorporan la opción de setReversed(true), que permite que la salida del scan sea en orden inverso, pero es menos eficiente.

- La clase se puede probar ejecutando:

```
export HADOOP_CLASSPATH=hbase-examples.jar
hbase HBaseTemperatureQuery 011990-99999
```