Sommaire

[API 1](#_Toc419815981)

[Configuration 3](#_Toc419815982)

[a) Réplication et de la politique de placement 4](#_Toc419815983)

[b) API et l'accès client 4](#_Toc419815984)

[c) Hadoop Single Point Of Failure, chose du passé? 4](#_Toc419815985)

[d) Fully Distributed Hadoop Cluster - Automatic Failover HA Cluster with Zookeeper & QJM 5](#_Toc419815986)

[e) Failover 5](#_Toc419815987)

# API

**package** fr.haddad.dfs.interfaces;

/\*\*

\* **@author** HADDAD Riadh

\*

\*/

/\*\*

\* for the test of this API, I used 2 NameNode and three datanodes

\* that are placed on 5 machines, the flows are sent to the active NameNode that is chosed automatically

\* and it manages communication with datanodes blocks in terms

\* of management ..... .

\* it is tested locally

\* if you want to send a stream remotely, you just use a

\* web service which receives a file and in using this API.

\*/

**import** java.io.IOException;

**import** java.net.URISyntaxException;

**import** java.util.List;

**import** org.apache.hadoop.fs.Path;

**import** fr.haddad.dfs.impl.FileFuture;

**import** fr.haddad.riadh.control.FileInHdfsAlreadyExistException;

**import** fr.haddad.riadh.control.FileInHdfsNotFoundException;

**import** fr.haddad.riadh.control.FileInLocalNotFoundException;

**import** fr.haddad.riadh.control.IsNoStorageSpaceException;

**import** fr.haddad.riadh.control.NeedToUseLargeFileMethodeException;

**import** fr.haddad.riadh.control.UserPermissionHdfsException;

**public** **interface** UploadFileManagerInterface {

/\*\*

\* When size of small file over <code>LARGE\_FILE\_THRESHOLD</code>, large

\* file Mechanism must be preferred.

\*/

**public** **static** **long** ***LARGE\_FILE\_THRESHOLD*** = 50000;

/\*\*

\* A method for large files asynchronously.

\*

\* **@param** localpath

\* like file:///C:/Users/Riadh/test.txt

\* **@param** RemoteFileURL

\* like hdfs://192.168.20.8:9000/riadh/

\* **@return** FileFuture represents the result of an asynchronous computation.

\* Methods are provided to check if the computation is complete, to

\* wait for its completion, and to retrieve the result of the

\* computation.

\* **@throws** FileAlreadyExistException

\* **@throws** IOException

\*/

**public** FileFuture uploadLargeFile(**final** Path localpath,

**final** Path RemoteFilePath) **throws** FileInHdfsAlreadyExistException,

FileInLocalNotFoundException, IsNoStorageSpaceException,

UserPermissionHdfsException, FileInHdfsNotFoundException, Exception;

/\*\*

\* method for small files (less than LARGE\_FILE\_THRESHOLD) in synchronous

\* mode.

\*

\* **@param** localpath

\* like file:///C:/Users/Riadh/test.txt

\* **@param** RemoteFilePath

\* like hdfs://192.168.20.8:9000/riadh/

\* **@throws** Exception

\* tu use the second methode)

\* **@throws** IOException

\* **@throws** URISyntaxException

\*/

**public** **void** uploadtSmallFile(**final** Path localpath, **final** Path RemoteFilePath)

**throws** NeedToUseLargeFileMethodeException,

FileInHdfsAlreadyExistException, FileInLocalNotFoundException,

IsNoStorageSpaceException, UserPermissionHdfsException,

IOException, FileInHdfsNotFoundException;

/\*\*

\* **@param** dir

\* name of directory url of directory that will be created like

\* hdfs://192.168.20.8:9000/user/folder1

\* **@throws** FileInHdfsAlreadyExistException

\* **@throws** IOException

\*/

// Create a new directory in HDFS

**public** **boolean** mkdir(Path dir) **throws** FileInHdfsAlreadyExistException,

IOException;

/\*\*

\* **@param** dir

\* name of directory url of directory that will be deleted like

\* hdfs://192.168.20.8:9000/user/folder1

\* **@throws** FileInHdfsNotFoundException

\* **@throws** IOException

\* **@throws** FileInHdfsNotFoundExecption

\* **@throws** FileInHdfsNotFoundExeception

\*/

// delete dir named directory in HDFS

**public** **boolean** delete(Path dir) **throws** IOException,

FileInHdfsNotFoundException /\*

\* throws

\* IOException,fr.riadh.control.

\* FileNotFoundException

\*/;

/\*\*

\* Future<FilesStatus> The result type returned by this Future's get method

\* for an asynchronous computation.

\*/

/\*\*

\* this methode test if **@param** hdfsfile exist in hdfs

\*

\* **@param** hdfsfile

\* **@return**

\* **@throws** UserPermissionHdfsException

\* **@throws** IOException

\* **@throws** FileInHdfsNotFoundException

\*/

**public** **boolean** ifExistsInLocal(Path hdfsfile) **throws** IOException,

UserPermissionHdfsException /\*

\* throws

\* IOException,UserPermissionHdfsException

\*/;

/\*\*

\* this methode return the list of files

\*

\* **@param** path

\* of hdfs file

\* **@return**

\* **@throws** IOException

\* **@throws** URISyntaxException

\*/

**public** List<Path> getListFile() **throws** IOException, URISyntaxException;

// test if path is exist in hdfs

/\*

\* public boolean ifExistsInHDFS(Path distination) throws IOException,

\* UserPermissionHdfsException; }

\*/

**public** **boolean** writeFile(**byte**[] content, Path hdfs)

**throws** UserPermissionHdfsException, IOException;

}

# Configuration

Le choix de système de fichier tombe sur HDFS dont sa la version la plus stable 2.6.0,car d’une part hadoop version 1.X constitue un SPOF alors que 2.X résout ce problème, d’autre nos partenaires travaillent avec le langage java ,de plus hdfs est open source et propose des divers fonctionnalités native .

|  |  |  |  |
| --- | --- | --- | --- |
|  | Système d’exploitation | File Configuration | Langage de programmation d’API |
| Hadoop DFS | Version 2.6.0  Hadoop (1.x SPOF) | hdfs-site.xml  core-site.xml | JAVA  (version 1.7 plus stable que 1.8) |
| Name Node  (hadoopmaster) | 1 Machine ubuntu 14.04 |
| Data Nodes (Slaves)  (Hadoopslave1  Hadoopslave2  Hadoopslave3)  Pour garantir la disponibilité j’ai répliqué les données sur trois data nodes | 3 Machine ubuntu 14.04 |
| Zookeeper | 3.4.6 | Zoo.cfg | --- |

## Réplication et de la politique de placement

Afin de rendre les données toujours disponibles, même si un serveur tombe en panne, SFD utilisent la réplication des fichiers en faisant plusieurs copies d'un des données sur différents serveurs. Quand un client demande une donnée, il accède à une transparence des copies. Pour améliorer la tolérance aux pannes, les répliques sont stockées sur des serveurs différents selon une politique de placement. Par exemple, des répliques peuvent être stockées sur des noeuds différents, sur différents supports, en différents emplacements géographiques, de sorte que

si un défaut se produit ne importe où dans le système, les données sont encore disponibles

## API et l'accès client

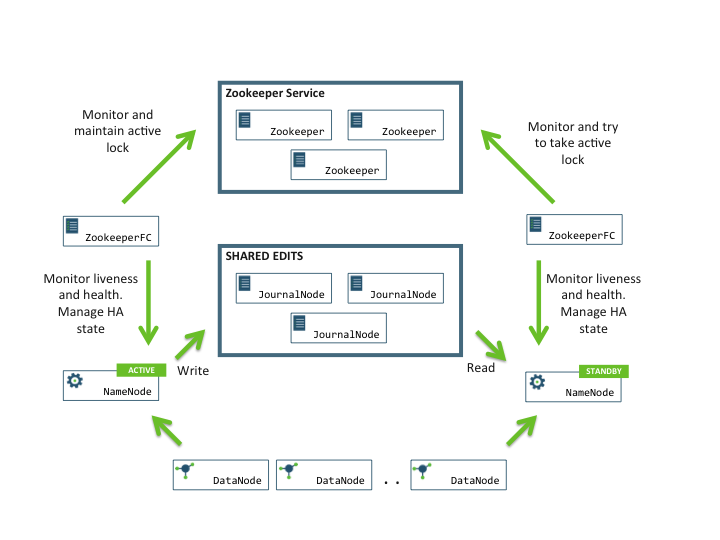
HDFS fournit une bibliothèque de code qui permet aux utilisateurs de lire, écrire et supprimer des fichiers et créer et supprimer des répertoires. Les utilisateurs accèdent à un fichier en utilisant son chemin dans l'espace de noms et de communiquer avec le NameNode pour savoir où récupérer ou mis les blocs de fichiers, puis il demande le transfert en communiquant directement avec les datanodes. Cela se fait de manière transparente en utilisant une API en C, Java ou REST.

## Hadoop Single Point Of Failure, chose du passé?

L'architecture initiale de Hadoop a été souvent critiqué pour le point de défaillance unique (SPOF) avec son unique desain de NameNode. (Hadoop 1.x)

Mais dans 2.x hadoop le problème est résolu.

## Fully Distributed Hadoop Cluster - Automatic Failover HA Cluster with Zookeeper & QJM



## Failover

Le client utilise la classe ConfiguredFailoverProxyProvider pour connaitre le Name Node active la configuration et comme suit :

<property>  
<name>dfs.client.failover.proxy.provider.mycluster</name>  
<value>org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider</value>  
</property>

Et coté java, j’ai inclus les fichiers hdfs-site.xml et core-site.xml qui sont indispensable par le proxy provider.