Paux J-Baptiste
Verstraeten Baptiste

HDFS v.0 final

Architecture (classes principales) :

- **NameNode.java** → classe implémentant un Name Node
- **DataNode.java** → classe implémentant un Data Node
- **HdfsClient.java** → Client Hdfs, interface avec le serveur Hdfs, permet de créer, lire, supprimer un fichier et lister les fichiers de l'architecture Hdfs.
- **HdfsServer.java** → Classe serveur Hdfs, lance un DataNode et éventuellement un NameNode et gère les communication via sockets entre le client et les nodes.

Points délicats :

→ Communication entre le client et le serveur Hdfs par le biais de sockets, quel protocole utiliser ?
→ Mise en place au sein du nameNode d'un système d'enregistrement des dataNodes actifs et au sein du dataNode d'un système d'enregistrement auprès du nameNode, comment faire ?

Décisions techniques :

- Il n'y a qu'un seul serveur Hdfs gérant la communication via socket, pas de distinction entre un serveur lancant un nameNode et un autre serveur lancant un dataNode. Le serveur Hdfs gère les communications avec le client et redirige les messages vers le nameNode ou le dataNode qu'il héberge selon le type de message.

- La communication via socket se fait à l'aide de deux formats de messages : **HdfsQuery.java** et **HdfsResponse.java**. Ces formats sont sérialisés dans la socket de communication pour être transmis. Cela permet une uniformisation des messages qui sont alors plus faciles à traiter par le Serveur, qui en fonction du champs type de **HdfsQuery.java** sait comment traiter la requête.

- Les dataNodes doivent s'enregistrer auprès du nameNode toutes les 10 secondes via une socket TCP. Si au bout de 10 secondes le nameNode ne reçoit pas de notification d'un dataNode il est considéré comme absent et ne sera plus pris en compte dans les opération d'Hdfs.

- Les données du nameNode sont sauvegardées en mémoire RAM, contrairement à celles du dataNode. Du coup à l'arrêt du serveur Hdfs l'état du nameNode lancé (si il y a lieu) est

sauvegardé dans **HdfsServer.conf**, afin de pouvoir relancer le nameNode à l'identique avec l'option **-f** du serveur.

Possibilités d'amélioration :

- La notification de présence des dataNodes auprès du nameNode se fait actuellement à l'aide de sockets TCP. c'est assez lourd sachant que cette notification n'as pas besoin de transmettre de données, il serait envisageable de l'implémenter plutôt avec un protocole comme ICMP.
- La commande listant les chunks présent sur un dataNode liste actuellement tout le contenu du répertoire source du dataNode, y compris des fichiers qui ne sont pas des chunks, le dataNode n'ayant pas de mémoire des chunks qu'il possède. Il serait possible de modifier ça en lui rajoutant en mémoire vive une liste de ses chunks.

Utilisation Hdfs :

Compiler depuis le répertoire src/ en utilisant les commandes :

```
$ make client
$ make server
```

HdfsServer.java :

Permet de construire l'architecture Hdfs en lancant un nameNode et des dataNodes. Pour lancer un nouveau nameNode (depuis répertoire src/) :

```
$ java hdfs/hdfsServer -n
```

Pour lancer un nameNode depuis un fichier de sauvegarde (généré automatiquement à l'arrêt du serveur) :

```
$ java hdfs/hdfsServer -f <fichier_save>
```

Notez que ces deux commandes lancent aussi un dataNode sur la machine locale, qui permet d'avoir au moins un dataNode enregistré dans le nameNode qu'on vient de lancer. Pour lancer un dataNode seul il suffit de faire :

```
$ java hdfs/hdfsServer
```

Il est ensuite demandé de préciser l'adresse du nameNode et le dataNode ira automatiquement s'enregistrer auprès du nameNode. l'option -h <nameNodeHost> permet de préciser l'adresse du nameNode en argument pour qu'il ne soit pas demandé ensuite.

Toute autre argument ajouté sera considéré comme la racine du serveur Hdfs, là où seront enregistrés les chunks du dataNode et la sauvegarde du nameNode lors de l'arrêt du serveur.

```
$  java hdfs/hdfsServer root/server/
```

HdfsClient.java :

Classe interface permettant la communication avec hdfsServer. Cette classe possède aussi un main permettant l'utilisation en ligne de commande.

Cette commande permet d'écrire un fichier sur le hdfs :

```
$ java hdfs/hdfsClient write <line|kv> <file_src> [file_hdfs]
```

Les arguments entre crochets sont optionnels. file_src est le fichier à copier en local, file_hdfs le nom sur le système hdfs, par défaut le nom du fichier en local, et line ou kv le type de fichier.

Lecture de fichier :

```
$ java hdfs/hdfsClient read <file_hdfs> [file_dst]
```

file_hdfs est le nom du fichier dans le système et file_dst le nom sous lequel il sera copié en local, par défaut le nom dans le système hdfs.

Suppression de fichier :

```
$ java hdfs/hdfsClient delete <file_hdfs>
```

file_hdfs nom du fichier à supprimer.

Liste des fichiers présents dans le système :

```
$ java hdfs/hdfsClient list
```

Liste des chunks composants un fichier :

```
$ java hdfs/hdfsClient list <file_hdfs>
```

Liste les chunks et les dataNodes gérant les chunks du fichier file_hdfs.

Liste des chunks enregistrés sur un dataNode :

```
$ java hdfs/hdfsClient chunks
```