Projet Hidoop Partie HDFS
Version 0, alpha

Yanis Pobel, Marine Médard

7 novembre 2017

Table des matières

1 Message

La classe Message permet l'envoi ou la réception de données d'un client au serveur et réciproquement. C'est une classe générique, qui permet d'envoyer ou recevoir n'importe quel type d'objets. Elle à deux méthodes, send et reception, qui sont chacunes surchargées de deux façon différente, selon si on est dans le serveur, ou si on cherche à s'y connecter.

1.1 send

La méthode send prend en paramètre l'objet à envoyer, ainsi que un ServerSocket si nous somme dans le serveur, et le numéro de port du serveur, si nous sommes dans le client. Elle ouvre un socket, puis y créer un ObjectOutputStream, avnt d'écrire l'objet à envoyer dans celui-ci.

1.2 reception

La méthode reception prend en paramètre un ServeurSocket si nous somme dans le serveur, et le numéro de port du serveur si nous sommes dans le client. Elle ouvre un socket, puis y créer un ObjectInputStream, avant de lire et retourner l'objet contenu dans celle-ci.

2 Client

Le HdfsClient parmet de à l'utilisateur d'entrer un fichier dans le système HDFS, le client gère la connexion avec les différents serveurs.

L'utilisateur peut proposer envoyer 3 commandes différentes à HdfsClient :
- write : permet d'enregistrer un fichier dans le système hdfs (c'est à dire sous forme de fragments sur différents serveurs (3 dans la version 0 de hidoop).
- read : permet de récupérer un ficier enregistré dans hidoop à partir des différents fragments enregistrés sur les serveurs
- delete : permet de suppimer un fichier entré dans le système hidoop (c'est à dire supprimer tous les fragments du fichier)

On lance le HdfsClient en ligne de commande.

2.1 Fonctionnalité : le "write"

Pour demander au HdfsClient une écriture de fichier sur les HdfsServers, il faut entrer sur un terminal une commande de la forme :
» java HdfsClient "write" format nomDuFichier
- format : correspond au format dans lequel est le fichier : soit "line" (fichier texte), soit "kv" (fichier de kv). Cela correspond également au format dans lequel seront stockés les fragments du fichier sur le serveur.
- nomDuFichier : correspond au nom du fichier que l'on veut fragmenter, puis sauvegarder dans différents serveurs.

HdfsClient se comporte ensuite différement selon s'il s'agit d'un format LINE ou d'un format KV.

2.1.1 Format LINE

HdfsClient commence par ouvrir le fichier (correspondant à nomDuFichier) en lecture grâce à un fileReader et à un BufferedReader pour compter le nombre de lignes du fichier initial. Puis il détermine la taille de chacun des fragments qui seront stockés sur les serveurs, la taille de chaque fragment est :
nb de lignes du fichier initial / 3 (+1 si le reste de cette division est positif et qu'il reste des lignes à écrire)

Puis on fait une boucle for sur les 3 serveurs afin de pouvoir écrire sur chaque serveur une ArrayList de String (chaque String correspond à une ligne).

HdfsClient envoie la commande CMD_WRITE au serveur. Ensuite, il lui envoie le nom du fragment de fichier (nomDuFichier + n° de serveur). Puis il lui envoie le format du fichier c'est à dire LINE et enfin la ArrayList de ligne.

2.1.2 Format KV

HdfsClient commence par ouvrir le fichier (correspondant à nomDuFichier) en lecture grâce à un FileInputStream et à un ObjectInputStream pour compter le nombre de KV du fichier initial. Puis il détermine la taille de chacun des fragments qui seront stockés sur les serveurs, la taille de chaque fragment est :
nb de KV du fichier initial / 3 (+1 si le reste de cette division est positif et qu'il reste des KV à écrire)
Ensuite, on ferme le FileInputStream et le ObjectInputStream afin d'en ouvrir de nouveau et de pouvoir, encore une fois pacourir le fichier.

Puis on fait une boucle for sur les 3 serveurs afin de pouvoir écrire sur chaque serveur une ArrayList de KV.

HdfsClient envoie la commande CMD_WRITE au serveur. Ensuite, il lui envoie le nom du fragment de fichier (nomDuFichier + n° de serveur). Puis il lui envoie le format du fichier c'est à dire KV et enfin la ArrayList de KV.

2.2 Fonctionnalité : le "read"

Pour demander au HdfsClient une écriture de fichier sur les HdfsServers, il faut entrer sur un terminal une commande de la forme :
» java HdfsClient "read" nomDuFichier

D'abord on crée un fichier qui s'appelle "testRead" // Puis, on commence par faire une boucle for sur les trois serveur. Cela nous permet d'envoyer à tous les serveurs la commande CMD_READ. Puis on envoie au serveur le nom du fragment qu'il doit envoyer. On reçoit le type dans lequel le fragment est écrit dans le fichier (soit KV soit LINE). Puis on reçoit une ArrayList d'Objets de la part du serveur. On fait donc une boucle for sur les éléments de cette liste :

- si le format est LINE, on crée un FileWriter et on écrit chaque objet de la liste (qui sont du coup des String) dans le fichier testRead. Puis on ferme le FileWriter.
- si le format est KV, on crée un FileOutPutStream et un ObjectInputStream afin d'écrire chaque objets dans le fichier testRead.

2.3 Fonctionnalité : le "delete"

Pour demander au HdfsClient une écriture de fichier sur les HdfsServers, il faut entrer sur un terminal une commande de la forme :
» java HdfsClient "delete" nomDuFichier

Pour chaque serveur, corres on envoie la commande de suppression de fichier, puis le nom du fragment à supprimer.

3 Serveurs

La classe HdfsServer prend en argument un numéro de port, et créer un Server Socket sur ce port. Ensuite, on effectue un while(true) de tel façon que le serveur tourne indéfiniement. On attend alors, la réception d'une commande, que l'on va traiter selon sa valeur :

3.1 CMD_OPEN_R

Un Format demande d'ouvrir en lecture un fichier contenu dans le serveur. Pour cela, on réceptionne le nom du fichier, puis on retourne son path afin que Format puisse lire les informations souhaitées sur le fichier.

3.2 CMD_OPEN_W

Un Format demande d'ouvrir en ecriture un fichier dans le serveur. Pour cela, on réceptionne le nom du fichier sur lequel travaille Format, puis on retourne son path, auquel on ajoute "-res". Afin, que Format écrive directement dans le serveur.

3.3 CMD_WRITE

Un HdfsClient peut envoyer une commande d'écriture. On réceptionne alors le nom du fragment de fichier envoyé, pour créer le fichier en dur dans le serveur. Puis on reçoit le format du fichier, et son contenu sous forme d'ArrayList que l'on écrit en tant qu'objets dans ce fichier.

3.4 CMD_READ

Un HdfsClient peut envoyer une commande de lecture On réceptionne alors le nom du fichier que l'on souhaite lire. Puis on récupère son contenu, c'est à dire son format et une ArrayList que l'on retourne au HdfsClient.

3.5 CMD_DELETE

Un HdfsClient peut envoyer une commande de suppression. On réceptionne alors le nom du fichier, et on le supprime dans le serveur.

4 Formats

Nous avons crée les classes FormatLine et FormatKV implémentant Format. Celle-ci permet de faire le lien entre le Daemon et le Serveur pour interpréter le fichier selon son format, puis écrire dans le serveur, un fichier résultant.

4.1 constructeur

Le constructeur prend en paramètre le nom du fichier à traiter, et le numéro de port du serveur sur lequel il est stocké. L'index lui est initialisé à un quand il est définit en tant qu'atribut. Nous créons aussi les Messages qui seront utilisés.

4.2 méthode open

La méthode open prend en paramètre un OpenMode :

- Si celui-ci est R, elle ouvre le fichier attribut en lecture, pour cela elle envoie au serveur du port attribut une commande CMD_OPEN_R, puis le nom du fichier traité. Le serveur retourne le chemin de celui-ci. On récupère alors la ArrayList de ce fichier représentant son contenu. On met aussi un boolean OpenR à true pour savoir que les descripteurs sont ouverts
- Si celui-ci est W, elle ouvre un fichier en écriture, pour cela elle envoie au serveur du port attribut une commande CMD_OPEN_W, puis le nom du fichier attribut. Le serveur retourne le chemin d'un nouveau fichier où écrire. On ouvre alors desdescripteurs en écriture, et mettons un boolean OpenW à true. Puis on écrit le Type de format correspondant à la classe utilisé (LINE pour FormatLine, KV pour FormatKV) en objet dans ce fichier. On créer aussi une ArrayList (de String pour FormatLine, de KV pour FormatKV) initialisé à vide.

4.3 méthode read

Pour FormatLine, on retourne un KV, ayant pour clé l'index courant, qui est le numéro de ligne de la valeur, qui est une ligne, correspondant au index-1 élément de la liste récupérer dans open,en lecture. Pour FormatKV, on retourne le index-1 élément de la liste de KV récupérer dans open, en lecture. Puis dans tous les cas, on incrémente de un la valeur de l'index.

4.4 méthode write

On ajoute à la liste de KV créer dans open en écriture lae KV en paramètre.

4.5 méthode close

Si openW est vrai, on écrit la liste de KV créer dans open en écriture, dans le fichier écriure. Puis si openR ou openW sont à true, on ferme les descripteurs associés. Enfin, on met ses booléens à false.

4.6 méthode getIndex

Retourne l'index courant

4.7 méthode getFname

retourne le nom du fichier traité courant

4.8 méthode setFname

Modifie le nom du fichier traité par celui en paramètre.

5 Conclusion

Cette implémentation du Hdfs permet notamment de rajouter de nouveaux formats (autres que KV ou LINE) sans toucher au code du hdfsServer, seulement en ajoutant des case aux switch dans HdfsClient et en créant une nouvelle classe implémentant Format pour ce nouveau format. Il faut cependant penser, quand on manipule un FormatLine ou un FormatKV à toujours appeler la méthode open avant de lire ou d'écrire sur un fichier du serveur et surtout à toujours appeler la méthode close quand on a finit d'utilier le formatKV (ou le formatLine) en écriture avant de l'utiliser en lecture. Car l'écriture sur le disque se fait dans le close.