Compte rendu lecture du code croisé

Sherwin Sabras et Luc Bourguignon

Ci-dessous vous trouverez la synthèse attendue : l'analyse de la partie HDFS réalisée par Aymen, Ali et Khalil. On s'intéresse dans un premier temps à une description du code pour montrer qu'il fait bien ce qui est attendu puis nous réalisons des remarques d'ordre plus général.

PARTIE I : Description du code

A) *Commande.java*

Comme proposé dans le sujet, l'implantation choisie est une énumération avec trois possibilités (CMD_READ, CMD_WRITE et CMD_DELETE).

B) *HdfsClient.java*

1) Description du code

La grappe (cluster) est composée de 5 machines (aymen, ali, luc, sherwin, khalil).

HdfsDelete : Un socket est créé pour chaque machine du cluster, la commande suppression est appliquée sur la machine considérée du cluster.

HdfsWrite : A partir de la ligne 49 débute la division du fichier en fragments. Une variable "division" est introduite, elle correspond au nombre de ligne que chaque fragment doit posséder pour que l'ensemble du fichier soit livré à la grappe (un "reste" est introduit dans le cas où la division ne tombe pas juste). On ajoute (à l'aide d'une boucle) chaque ligne que doit contenir le i-ème fragment plus éventuellement un reste (quelques lignes supplémentaires). Un socket est ensuite créé pour transmettre le fragment créé à la i-ème machine du cluster.

HdfsRead : Cette méthode permet de rassembler les fragments sur lesquels ont été appliqués le traitement souhaité en un fichier final. Pour cela, on utilise encore les sockets en mode connectés (TCP). On crée un nouveau fichier texte et pour chaque machine on crée un socket qui permet de lire le contenu du fragment de la i-ème machine et de l'écrire dans le fichier texte final précédemment créé.

Main : Dans le main, on analyse la ligne de commande rentrée et on applique la commande souhaitée à savoir : lecture (lire les fragments, les concaténer et les stocker dans un fichier texte résultat), écriture (le fichier texte initial est lu puis divisé en fragments qui sont transmis à chaque machine du cluster de manière plus ou moins équitable) ou enfin suppression (qui permet de supprimer les fragments stockés sur chaque machine de la grappe).

2)Pertinence et complétude

La description faite précédemment du code permet de montrer que le service HdfsClient est conforme à ce qui est attendu par le sujet : on a bien une utilisation en mode connecté (TCP) des sockets. Le code réalise les tâches attendues et gère les exceptions.

C) *HdfsServeur.java*

1) Description du code

Le serveur accepte la création de socket et applique la commande qui lui est transmise : lecture (lire le fragment), écriture (écrire dans le fragment) ou enfin suppression (qui permet de supprimer les fragments stockés sur le serveur).

2)Pertinence et complétude

Comme précédemment, le code de HdfsServeur semble cohérent avec ce qui est demandé, on a un traitement des exceptions et les ouvertures et fermetures des sockets sont réalisées conformément à ce qui était fait dans le cours.

PARTIE II : Cohérence globale et remarque générales

Le résultat obtenu a une structure cohérente et une architecture claire. Le code est écrit de façon logique et les fonctions proposées et réalisées sont bien complémentaires. L'utilisation des sockets assez "lourde" peut laisser supposer qu'il y a des doublons, mais nous ne voyons pas comment ils auraient pu être évités.

Enfin, le choix proposé par le sujet d'utiliser les sockets en mode connecté (et qui est respecté ici) permet de bénéficier des avantages du mode connecté : les problèmes de communications sont gérés automatiquement et les primitives d'émission et de réception sont simples.