Revue du travail réalisé par le binôme complémentaire

Armelle Jezequel
Loïs Roth
Paul Lacaille
Yannis Benabbi
Alexandre Chatain

15 janvier 2020

Table des matières

2ème année - Sciences du numérique

1 Revue de la partie HDFS

1.1 Partie technique

1.1.1 Présentation de l'architecture

Les classes : Le code pour la partie HDFS se décompose en trois classes : ClientHDFS, ServeurHDFS et DaemonHDFS. Le client représente l'utilisateur qui envoie son fichier sur le serveur et le récupère. Le serveur est le point d'entrée du cluster qui s'occupe de décomposer et de recomposer les fichiers envoyés par le client. Chaque ordinateur ou noeuds du cluster lance un daemon qui va récupérer des fragments de fichier. Il y a trois fonctionnalités réalisables pour un utilisateur : hdfsWrite qui envoie un fichier sur le serveur, hdfsRead qui récupère un fichier depuis le serveur et hdfsDelete qui supprime un fichier du serveur.

Le code des trois classes a été réalisé avec des sockets. On voit que la classe HdfsClient de base n'a pas été utilisée. La compilation des fichiers utilisés ne compile pas au début car les trois classes citées précédemment font appel à la classe FragmentTexte qui a été renommée FragmentTXT. Après avoir corrigé cette erreur, la compilation fonctionne. Il n'y a pas d'instructions sur comment utiliser ces classes donc nous avons demandé à l'autre binôme comment les tester.

1.1.2 Exécution du module

Pour l'instant, le daemon, le serveur ainsi que le client doivent tous être lancé sur la même machine car les noms ('localhost') et les ports des daemons et du serveur sont codés en dur dans les classes ServeurHDFS et ClientHDFS. Il faudrait que ces valeurs soient passées en argument lors du lancement des processus. Il n'y a que lors du lancement des daemons qu'il faut passer en argument le nom d'hôte et le port. Ils ne peuvent évidemment pas être codés en dur dans la classe DaemonHDFS vu qu'il y en a plusieurs. Il faut cependant que le nom d'hôte et le port utilisé soient ceux écrits dans la classe ServeurHDFS, sinon le serveur ne pourra pas communiquer avec. Cela implique aussi qu'il ne sert à rien de créer plus d'un daemon car les autres ne seront pas utilisés.

Donc lors du lancement du client celui-ci va se connecter au serveur qui va à son tour se connecter aux noeuds du cluster. Le client envoie ensuite la commande et le nom du fichier concerné par la commande au serveur qui va relayer cela aux noeuds. Si la commande est hdfsWrite, le client va en plus envoyer le contenu du fichier et le serveur va se charger de découper puis de répartir les fragments sur les noeuds. Les commandes hdfsRead et hdfsDelete n'ont pas encore été implémenté mais le début restera le même.

1.1.3 Test

Pour pouvoir tester les méthodes, il faut d'abord envoyer un fichier sur le serveur avant de de pouvoir le récupérer ou le supprimer. Il faut donc d'abord tester la méthode hdfsWrite. Pour faire cela, il faut lancer le daemon, le serveur et après le programme du client avec comme argument le numéro de la commande, donc ici 1 pour write, ainsi que le nom du fichier concerné. L'envoi du fichier semble fonctionner puisqu'il n'y a pas d'erreur affiché par les programmes. En regardant dans la classe DaemonHDFS, on peut voir que les fragments sont pour l'instant stockés dans une HashMap. Pour savoir si le fichier a été correctement découpé et les fragments correctement stockés, il faudrait utiliser ensuite la méthode hdfsRead, mais celle-ci n'a pas encore été implémentée. Mais le code des classes suggère que si aucune erreurs n'est renvoyée, c'est que les fragments sont arrivés jusqu'au daemon. Il reste juste à vérifier si le fichier a été découpé proprement. En ajoutant une ligne dans la classe du daemon pour afficher le texte du fragment, on peut voir que les fragments arrivent effectivement jusqu'au daemon et que le fichier texte est découpé par rapport aux sauts de ligne.

1.2 Synthèse