Projet systèmes concurrents/intergiciels
Implantation et utilisation du modèle Map-Reduce
Rapport final de la Première Version - V0

Ismail Moussaoui
Youssef Bendagha

23 novembre 2017

Table des matières

1 Introduction

Le but de ce projet est implémenter une version simplifiée de la plateforme Hidoop. Durant cette phase du projet (V0), nous nous occuperons de la réalisation d'un service HDFS permettant un accés massivement concurrent à de larges volumes de données.

2 Architecture générale

2.1 Principe de fonctionnement

Notre partie du projet (HDFS) se compose principalement des classes suivantes :

- HdfsClient qui contact le Name Node afin d'obtenir la configuration des différents Data Nodes ainsi que les métadonnées, pour ensuite contacter les Data Nodes nécessaires pour effectuer une commande donnée (Read, Write ou Delete).
- HdfsServer impémente chacun des différents Data Nodes, qui gèrent les accès aux chunks conservés sur la machine à laquelle ils sont affectés. Habituellement chaque machine de la grappe HDFS accueille un DataNode.
- NameNode est un serveur maitre qui coordonne un pool (cluster) de serveurs HDFS ; il gère le catalogue du système de fichiers (métadonnées et localisation des chunks de chaque fichier).

Le tout communique via des sockets en mode TCP.

2.2 HdfsClient

Au démarrage du client HDFS (HdfsClient), ce dernier charge le fichier "configClient.properties" contenant l'adresse IP ainsi que le numéro de port du Name Node, ainsi que le chemin local vers où sera stocké le fichier lu depuis HDFS.
HdfsClient pourra alors manipuler les fichier dans HDFS, et ce en contactant les différents Data Nodes (HdfsServer) qui gèrent les accés aux chunks conservés sur la machine à laquelle ils sont affectés, mais le client HDFS contacte tout d'abord le Name Node afin d'obtenir les métadonnées nécessaires à ladite manipulation.
Chaque intéraction entre HdfsClient et HdfsServer sera alors initiée par l'envoi d'un message spécifiant la commande à exécuter par HdfsServer. Nous avons utilisé une syntaxe assez simple pour ces messages : Le code commande, suivi du nom de fichier (path du fichier dans la machine client dans le cas du Write, ou path des chunks dans les Data Nodes dans le cas de read ou delete). La classe HdfsClient peut être utilisée depuis la ligne de commande ou directement depuis du code pour lire, écrire ou détruire des fichiers :

- HdfsWrite(Format.Type fmt, String localFSSourceFname, int repFactor) permet d'écrire un fichier dans HDFS. Le fichier localFSSourceFname est lu sur le système de fichiers local, découpé en fragments (de taille égale à celle du fichier à écrire divisé par le nombre de Data Nodes disponible) et les fragments sont envoyés pour stockage sur les différentes machines. fmt est le format du fichier (Format.Type.LINE ou Format.Type.KV). repFactor est

le facteur de duplication des fragments ; pour cette version il sera considéré comme valant 1 (pas de duplication).

- HdfsRead(String hdfsFname, String localFSDestFname) permet de lire un fichier de nom hdfsFname à partir de HDFS. Les fragments du fichier sont lus à partir des différentes machines, concaténés et stockés localement dans un fichier de nom localFSDestFname. Pour des besoins de simplification, nous avons adopté une convention de nommage pour les différents chunks d'un fichier dans HDFS ainsi que du fichier destination (localFSDestFname) : les différents chunks porteront le nom "hdfsFname"+i avec i un entier déterminant l'ordre des chunks d'un fichier donné ; localFSDestFname portera quand à lui en réalité le nom "hdfsFname-read".

- HdfsDelete(String hdfsFname) permet de supprimer les fragments d'un fichier stocké dans HDFS.

2.3 HdfsServer

Le démon HdfsServer doit être lancé sur chacune des machines du pool (cluster) de serveurs HDFS, en commençant par charger son fichier de configuration "configDemon"+i+".properties" ; celui-ci contient le numéro de port du démon ainsi que le path du fichier dans lequel sera stocké le chunk, fichier qui doit exister au préalable dans la machine ; i est un entier indiquant le numéro du démon dans le cluster.
Il communique ensuite avec HdfsClient via les sockets et exécute la commande envoyée par le client HDFS (le traitement du message reçu par le client est conféré à "InteractionClient" qui identifie la commande à exécuter et l'exécute).

2.4 NameNode

Le NameNode gère les métadonnées du système de fichiers : pour chaque fichier, une structure regroupe les informations utiles à la gestion du fichier. Cette structure comporte en particulier :
— le nom dudit fichier
— la taille du fichier
— le format du fichier
— la taille des blocs (chunks)
— la liste des identifiants de blocs (chunk handles) constituant le fichier
Ces éléments se trouvent dans le fichier "chunks.metadata".
La taille importante des chunks permet à cet ensemble de métadonnées de rester suffisamment compact pour être conservé en mémoire vive, ce qui assure l'efficacité des accès aux données du NameNode.
Lors de son lancement, NameNode charge le fichier de configuration "confi-

gNameNode.properties" qui fournit les adresses IP et ports utilisés par les différents serveurs.

3 La gestion de format

Chaque bloc (chunk) contient une suite d'enregistrements, dont le format est spécifique à l'application. Pour chaque format, une classe doit être définie, ce qui permet de lire et écrire dans un fichier dans ce format. Ces classes de lecture/écriture de formats sont bien nécessaires, car étant donné que les fichiers peuvent être coupés en fragments par HDFS, il faut qu'HDFS puisse faire une coupure cohérente. En particulier, un fichier de structures ne peut pas être coupé au milieu d'une structure. Dans notre cas, on ne considère que 2 formats : le format texte (LINE) et le format Clé-Valeur (KV), mais d'autres formats peuvent être envisagés et introduits par la suite.
Une classe implantant un format implante l'interface Format. Le format KV correspond au format des données intermédiaires produites durant l'exécution de l'application MapReduce.

4 Utilitaires

Dans cette partie, nous introduiront des classes secondaires mais néanmoins nécessaires au bon fonctionnement du système HDFS.

NameNodeInteraction Cette classe traite les messages (contenant des commandes) du Client HDFS que reçoit le NameNode, et les traite de manière à en déduire la commande à effectuer (modifier les métadonnées à la suite d'une écriture dans HDFS, founir les métadonnées dont a besoin le client, ...).

ChunkIdentifier Un objet ChunkIdentifier possède trois attributs : le chunkId (ou le nom du chunk dans le DataNode), l'adresse du DataNode dans lequel se trouve le chunk, ainsi que le numéro de port de ce DataNode. Cet objet sert à identifier les chunks d'un fichier donné pour une lecture ou une supression dudit fichier du système HDFS.

ChunkMetaData et FilMetaData Un objet ChunkMetaData possède 3 attributs : le numéro du node (à chaque node du cluster est attribué un entier, ce numéro est propre au node auquel il est attribué), l'identifiant (nom de chunk), ainsi que sa taille. Chaque objet FileMetaData représente la métadonnée d'un fichier dans HDFS ; il contient entre autres une ArrayList de ChunkMetaData, cela permet de faciliter la lecture d'un fichier dans HDFS ainsi que l'affichage des informations relatives à un fichier à l'aide d'une commande ls (expliquée ultérieurement).

NodeIdentifier Un objet NodeIdentifier possède les attributs suivants : le numéro de node, son adresse IP ainsi que son numéro de port. Cet objet est utilisé pour communiquer avec les nodes nécessaires pour exécuter une commande (en réalité, c'est une ArrayList contenant tout les NodeIdentifier nécessaires qui est utilisée pour contacter les nodes un à un).

Le package Json En ce qui concerne le dossier "json-simpe-master", c'est un package qui sert à lire et écrire des fichiers json en java, ce qui permet par exemple d'utiliser "chunks.metadata" qui est un fichier utilisant la syntaxe json. La raison de ce choix de json est la simplicité de la lecture écriture depuis un fichier json (une fois le parseur bien en place).

5 Exemple / cas d'utilisation

Dans cette section nous présenterons quelques cas concrets de l'utilisation d'HDFS en ligne de commande.

5.1 Lancement d'HDFS

On commence tout d'abord par lancer les DataNodes un à un en utilisant la commande "HdfsServer ../config/configDemon1.properties" - "../config/configDemon1.properties" étant le chemin local (au Data Node) au fichier de configuration du démon1 (en prenant une convention de nommage des fichiers de configurations, celui du démon2 sera nommé "configDemon2.properties" - ce qui permet d'initialiser son numéro de port ainsi que le chemin du repértoire dans lequel seront stockés les chunks.

Ensuite on lance le NameNode avec la commande "NameNode ../config/configNameNde.properties" "../config/configNameNde.properties" étant le chemin vers son fichier de configuration. Cela permet de charger son numéro de port ainsi que le chemin local vers lequel on stockera le fichier "chunks.metadata", en plus des numéros de ports et adresses IP des différents DataNodes.

La partie principale du système HDFS étant lancée, un client HDFS peut maintenant se connecter à ce système via le NameNode pour exécuter une certaine commande, et ce en lançant l'une des commandes suivantes (dépends de ce que l'on veut exécuter) :
"HdfsClient read <file>"
"HdfsClient write <line|kv> <file>"
"HdfsClient delete <file>"
"HdfsClient ls"
"HdfsClient ls <file>"
Les formats utilisés dans notre cas sont les formats "LINE" et "KV" (respectivement "line" et "kv" en line de commande), <file> étant tout simplement

le nom du fichier dans HDFS. à son lancement, HdfsClient charge automatiquement son fichier de configuration "../config/configClient.properties", permettant de se connecter au NameNode.

5.2 Les différents scénarios d'exécution

HdfsClient read fileName Lors de l'exécution de cette commande, le client HDFS demande au NameNode les métadonnées correspondantes au fichier fileName, puis en se basant sur ces métadonnées contacte un à un les DataNodes contenant les chunks de fileName, lis ces derniers, et stocke le résultat de cette lecture dans un fichier de nom fileName-read. un message "ce fichier n'existe pas" est affiché si le fichier est introuvable sur le système HDFS.

HdfsClient write line fileName Cette commande permet d'écrire le fichier fileName sous le format LINE dans les différents DataNodes disponibles. Le client HDFS, ayant au préalable chargé les informations relatives au DataNodes (dont notamment le numéro de port et l'adresse IP), les contacte un à un pour y écrire les différents chunks du fichier en question. Les chunks résultant portent le nom (par convention) "fileName"i, avec i étant le numéro du chunk, permettant de les différencier.
Après l'écriture, on met à jour les métadonnées.
un message "ce fichier n'existe pas" est affiché si le fichier est introuvable en local.

HdfsClient delete fileName Fonctionne de la même façon que read, sauf qu'au lieu de lire les différents chunks d'un fichier fileName, il les supprime un à un. Aprés suppression des chunks, les métadonnées sont mises à jour. un message "ce fichier n'existe pas" est affiché si le fichier est introuvable dans HDFS.

HdfsClient ls Cette commande est supplémentaire et n'est pas demandé pour notre partie du projet, elle est néanmoins utile car permettant de lister les fichiers disponibles dans HDFS, en affichant leurs noms, taille et format, ainsi que le nom, Node (numéro de node), et taille de leurs différents chunks. "HdfsClient ls fileName" affiche les informations relatives au seul fichier "fileName". un message "ce fichier n'existe pas" est affiché si ce fichier est introuvable sur HDFS.

6 Conclusion

Nous avons amélioré notre partie du projet (version V0) en se basant sur les remarques de notre binome complémentaire. Ainsi, nous gérons nous

même les erreurs d'exécution. Une autre amélioration notable est l'ajout du NameNode à notre système HDFS, ce qui permet notamment d'éviter au client, qui ne connait pas forcèment tout les DataNodes et l'état actuel du système HDFS, de devoir savoir au préalable ces informations (il se contente de connaitre le NameNode). Le NameNode permet aussi d'éviter une recompilation (couteuse en ressource et en temps de calcul) à chaque modification de l'état d'HDFS (en somme, les métadonnées et les DataNodes).
NameNode permettera par la suite (version V1) de gérer les défaillances des DataNodes.