Projet systèmes concurrents
Rapport version finale (étape 2)

Ismail Moussaoui
Youssef Bendagha

10 janvier 2018

Table des matières

1 Introduction

Le but de ce projet est d'implémenter une version simplifiée de la plateforme Hadoop. Durant cette phase du projet (l'étape 2), nous nous occuperons d'étendre le service Hidoop (support pour l'exécution répartie) développé par le binôme complémentaire et d'améliorer ses performances, permettant d'exécuter une application de type MapReduce. Nous présenterons ici notre version finale du service Hidoop.

2 Principales modifications par rapport à la dernière version de Hidoop

Dans cette version de Hidoop, nous avons ajouté un prétraitement (similaire au Shuffle décrit dans le sujet) intermédiaire des résultats des Map avant de transmettre les résultats de ce prétraitement au différentes tâches Reduce. Cela permettra de répartir les tâches entre plusieurs reduces et ainsi - en principe - gagner en performance.

2.1 Les formats nécessaires au multiple-reduce

Les formats dont nous disposons au départ sont inefficaces pour lire ou écrire dans les noeuds correspondant aux différentes tâches Reduce (noeuds distantes). En effet, il faut connaitre au préalable les identifiants (adresse IP, numéro de port, ..) du noeud dans lequel on veut lire/écrire des données, et aussi pouvoir choisir dans quel noeud écrire les données intermédiaire (là intervient la fonction ShuffleSimple qui sera détaillée par la suite) c'est pour cela on a pensé à définir des autres classes qui implementents l'interface format .

RemoteFormat Ce format permet de lire/écrire à distance des fichiers sur des DataNodes. Il possède - entre autres - comme attributs l'adresse IP et le numéro de port du DataNode visé, ainsi que le type de format a utiliser (LINE ou KV). Le fonctionnement de ses méthodes read() et write(...) est similaire à celui des méthodes de lecture/écriture du système HDFS, notamment par le fait d'utiliser les sockets en mode TCP pour permettre la lecture/écriture à distance (en communicant avec le HdfsServer correspondant en utilisant l'addresse IP et le port de celui ci) .

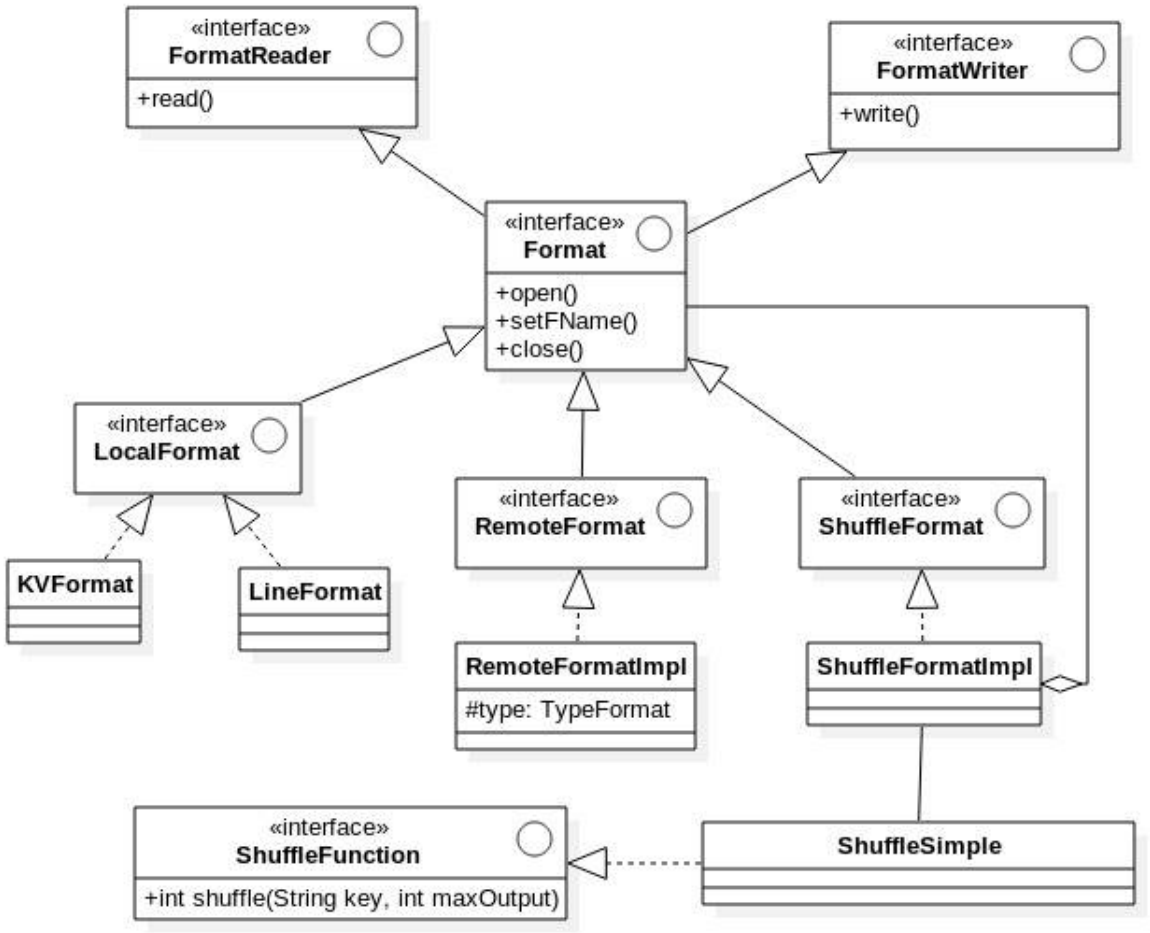
ShuffleFormat Ce format permet de lire de plusieurs fichier , et d'écrire sur plusieur fichier , pour l'écriture il utilise un critére pour savoir sur quel fichier écrire (c'est le shuffel). pour la lecture il lit des fichiers l'un aprés l'autre . Pour cela ShuffleFormat possède parmis ces attributs une liste de formats contenant les formats sous lesquels lire et/ou écrire et un objet de type ShuffleFunction. Pour l'écriture il utilise l'objet de type ShuffelFunction pour savoir sous quel format de la liste écrire. Pour la lecture l'objet de type ShuffelFunction n'est pas utilisé puisque il lit tous simplement les fichiers l'un aprés l'autre . ce format sera utile dans 2 étapes le processus du calcul réparti :

- Lors d'exécution des tâches map le ShuffelFormat permet d'écrire le résultat d'une map sur les différentes nodes reduces(en effet chaque map à un résultat pour chaque reduce). La taille de la liste des format dans ce cas correspond au nombre de reduces utilisés, et les formats de la liste pouvent être un RemoteFormat (contenant l'adresse IP et le numéro de port du DataNode correspondant au reduce à utiliser.), ou un local format (LINEFormat ou KVFormat) si le data node de map est le même que celle de reduce.
- Lors d'exécution des tâches reduce le ShuffelFormat permet de lire les résultats des maps pour ce reduce (les formats de la liste dans ce cas sont des format local)

L'interface ShuffleFunction ce interface a une seule méthode de signature : int shuffle(String key , int MaxOutput) il consiste a renvoyer un entier

(l'indice du format dans liste des format de ShuffelFormat qui correspond au DataNode du reduce à utiliser) ce entier doit être entre O et MaxOutput-1 (maxOutput est simplement le nombre de reduces)

La classe ShuffleSimple C'est une simple implementation de Shuffle-Function qui renvoie ce selon la première lettre de Clé un entier . Par exemple, avec 2 reduces, la fonction ShuffleSimple fera en sorte que les paires dont la clé commence par l'une des premières lettres de l'aphabet sera traitée par le premier reduce (int = 0), et celles dont la clé commence par l'une des dernières lettres de l'alphabet sera traitée par le second reduce (int = 1).



ci-dessus, le diagramme de classe des formats utilisés.

2.2 Fonctionnement général

Une fois le système HDFS en marche, nous lançons le calcul parallèle avec la classe Job, cette classe étant lancée avec en paramètres "./config/job/Job.properties" et "./config/clients/localhost.properties" qui sont les fichiers de configura-

tion du Job (contiennent entre autres : le nombre de reduces, le facteur de réplication, le chemin vers le fichier sur lequel appliquer le MapReduce, l'application MapReduce à utiliser, ainsi que l'adresse IP et le numéro de port du NameNode).

Première étape : répartition du fichier dans HDFS Pour pouvoir effectuer un calcul parallèle sur le fichier en question, ce dernier doit être réparti dans HDFS, et ce par le moyen de la méthode HdfsClient.HdfsWrite(...). Le fichier est divisé en un nombre de chunks (de taille à peu près égale) équivalent au nombre de DataNodes disponibles (pour le nommage, le nom d'un chunk numéro i est "<nomDuFichier> + i". Après la répartition du fichier dans HDFS, cette méthode (HdfsWrite(..)) enregistre les métadonnées dudit fichier dans le NameNode.

Seconde étape : l'exécution du MapReduce Une méthode de type Map s'applique sur chacun des différents chunks du fichier à traiter . Le résultat de chacun des maps est écrit dans les différents DataNodes correpondants aux reduces à utiliser, et porte comme nom <fileName>+"-red-temp"+i (i est l'indice du chunk sur lequel le Mp a été appliqué).

Ce choix du DataNode dans lequel écrire est dicté par le ShuffleFormat dans les lequel sont stockés les formats à utiliser (format "normal" s'il faut écrire dans le même DataNode que celui où est appliqué le Map, ou un RemoteFormat sinon). Les métadonnées des différents reduces (adresse IP, numéro de port, ..) sont obtenues en contactant en premier lieu le NameNode ; elles seront nécessaires pour définir le ShuffleFormat utilisé.

Ensuite, dans chaque DataNode correspondant à un reduce, une méthode de type reduce est appliquée sur les fichiers résultants de l'opération précédente .(le ShuffelFormat permet dans ce cas de lire ces différents résultats sans les merger pour gangner de performance) , et ce après avoir attendu la fin de l'éxecution de tout les maps. Les fichiers résultants des reduces portent le nom <fileName>+"red"+i , avec i étant l'indice du reduce.

Au final, le résultat des différents reduces est simplement concaténé via la méthode HdfsClient.HdfsRead(..).

3 Exemple concret et test des performances

Dans notre exemple, nous testons notre application de comptage de mot MyMapReduce avec 1 puis 2 reduces, sur un fichier en format LINE de 130,5 Mo (4.1 millions de lignes). Nous comparerons par la suite le temps d'éxecution de ces deux tests avec celui de l'application séquentielle de comptage de mots.

Remarque : La procédure de tests est similaire à celle des tests des versions précédentes : nous démarrons d'abord le système HDFS, puis les Dé-

mons un à un, et puis finalement la classe Job (en modifiant au besoin le nombre de reduce à utiliser). Les tests sont effectués en local sur les machines de l'ENSEEIHT.

Test avec 1 reduce Le comptage de mots du fichier en question s'effectue en t = 2915 ms.

Test avec 2 reduces Le comptage de mots du fichier en question s'effectue en t = 2609 ms

Comptage de mots version séquentielle Le comptage de mots s'effectue cette fois en t = 3292 ms.

La comparaison des résultats des différents tests permet de vérifier que les résultats sont effectivement cohérents.

Autre remarque : nous avons remarqué lors de notre test en local que le temps effectué par ce test diminue au fur et à mesure du relancement du test, et ce jusqu'à arriver à une certaine stabilité en terme de temps d'éxecution. Les résultats affichés sont ceux obtenus lors de cette "stabilité". Nous n'arrivons hélas pas à expliquer cette augmentation des performances entre deux essais du même test

4 Conclusion

Les résultats du test de performances démontrent l'utilité de Hidoop, qui serait d'autant plus performant s'il venait à être lancé sur plusieurs machines (permettant d'évitant la concurrence au niveau de la lecture/écriture sur disque dur) et sur des fichiers de plus grande taille. Néammoins, nous avons tenté son exécution - toujours en local - sur un fichier 3 fois plus grand (324 Mo) mais cela a conduit au plantage de la machine. De plus, le nombre de reduce joue un rôle non négligeable sur l'efficacité de Hidoop : la répartition des tâches entre les différents reduces améliore les performances de Hidoop.