PROJET HIDOOP PARTIE 2 : POINT D'AVANCEMENT 2

BENABBI YANNIS
CHATAIN ALEXANDRE
LACAILLE PAUL
JEZEQUEL ARMELLE
ROTH LOIS

14 FÉVRIER 2020

Sommaire

1 Améliorations implémentées depuis le dernier rendu

Nous avons apportées les corrections suivantes à notre projet :
- Un fichier par fragment (auparavant : un fichier pour tout les fragments).
- fichier stockés en local sur la machine (dans /tmp/).
- Job lance les maps en passant en paramètre la listes de fragments stockés sur le noeud du daemon appelé.
- concernant la partie hidoop, les objets reader et writer sont a présents créés par les daemons, et plus par le Job
- Refactoring du code HDFS afin de le rendre plus extensible
- Il n'y a plus de double envoi de fichier (Client → Serveur → Démons), mais simplement Client → Démons
- script de lancement en local (à distance en cours)
- le choix du nombre de machine se fait en modifiant un paramètre dans config/ClusterConfig.java

2 outils développés

2.1 scripts de déploiement

Pour déployer la plateforme en local : il suffit de vérifier que les lignes 14 et 18 de config/ClusterConfig.java sont respectivement décommentées et commentées, et d'exécuter le script lancerHidoop.sh .
Pour déployer la plateforme en distant : il suffit de vérifier que les lignes 14 et 18 de config/ClusterConfig.java sont respectivement commentée et décommentées, et d'exécuter le ssh_signin.sh puis le script lancerHidoopDistant.sh.

2.1.1 fin d'utilisation

Il faut lancer le script clean.sh pour libérer les ports hdfs et supprimer les registres rmi de hidoop.

2.2 création d'un fichier lourd

nous utilisons le script generer_ lourd.sh .

3 Résultats de l'étude de scalabilité sur WordCount

Nous n'avons fait que de tests qualitatifs : le programme fonctionne sur des fichiers de 1Mo, 10Mo, 100Mo.

4 Améliorations envisagées

4.1 Redondance

Nous voudrions implémenter un système de redondance (stocker les fragments en plusieurs exemplaires) afin que notre hidoop soit plus résistant aux pannes.

4.2 Optimisation des maps

Actuellement, les daemons de la partie hidoop reçoivent les listes des fragments qu'ils doivent traiter. Il font ce traitement séquentiellement, en lance les maps l'un après l'autre.

Nous voudrions instaurer ici une parallélisation, en créant le nombre de threads optimal en fonction du nombre de coeur de la machine. Nous avons dans l'idée que 2 ou 3 threads par coeur serait un nombre intéressant, nous copte faire des tests pour déterminer ce nombre.