Informatique et Mathématiques Appliquées

Rapport de la 3eme etape projet Hidoop

Youssef Achenchabe & Clement Caroff & Hamza Boukraichi & Mohamed Amine Tourari

Janvier 2018

Table des matières

1 Une autre application de Map/Reduce **1**
1.1 Principe de la méthode de Quasi MonteCarlo 1
1.2 Justification de l'utilisation du schéma Map/Reduce 1
1.3 Application à notre Map/Reduce 1
1.4 Resultats obtenus en local 1
1.5 Resultats obtenus en réseau 1

2 Limitations de l'application Hidoop développée **2**
2.1 NBmachines & problème shuffle 2

1 Une autre application de Map/Reduce

1.1 Principe de la méthode de Quasi MonteCarlo

Le but de la méthode de Quasi-MonteCarlo est en effet d'approximer la valeur de pi. Elle consiste à générer un nombre important de points dont les abscisses et ordonnées sont comprises dans l'intervalle [0 ;1] à l'aide de la suite de Halton. Pour chaque point généré, on calcule sa norme au carré : si celle-ci est inférieure à 1/4, le point appartient au cercle de rayon 1/4 et l'on marque le point comme tel. Le décompte du nombre de points appartenant au cercle sur le nombre total de points générés par la suite de Halton nous donne l'aire du cercle de rayon 1/4 et donc une approximation de Pi.

1.2 Justification de l'utilisation du schéma Map/Reduce

La partie coûteuse de cette méthode est la lecture dans le fichier contenant tous les points générés, ainsi que le calcul de la norme de ceux-ci. Ainsi, l'utilisation de l'architecture Map/Reduce semble être judicieuse dans l'utilisation de cette application. En effet, la configuration de cette application nous permet de diviser le traitement du fichier contenant tous les points générés par la suite de Halton en plusieurs fichiers contenant moins de points. Ces fichiers peuvent donc être traités séparément et simultanément par différents démons.

1.3 Application à notre Map/Reduce

Le type de format utilisé pour représenter le fichier contenant les points est donc naturellement le format LINE.

- La méthode map() consiste donc à lire un KV sur un *reader*, tester si la value V du KV (le point) est dans la région du cercle à l'aide de la méthode $isInRegion()$, puis écrire le KV résultant à l'aide du *writer* : la value V valant 1 ou 0 selon si le point est dans la région ou non.

- La méthode reduce() récupère la valeur de chaque point du fichier résultat et détermine l'approximation de pi.

1.4 Resultats obtenus en local

Dans les tests de la version local de l'application, avec un grand nombre de points, l'application commence à mettre beaucoup de temps pour s'éxecuter.

— Pour 2 700 000 points, l'application met 142 secondes.
— Pour 3 000 000 points, l'application met 174 secondes.

1.5 Resultats obtenus en réseau

Les tests ont été effectués sur 3 machines.

— Pour 2 700 000 points, l'application met 44 secondes.
— Pour 3 000 000 points, l'application met 49 secondes.
— Pour 4 000 000 points, l'application met 70 secondes.

2 Limitations de l'application Hidoop développée

2.1 NBmachines & problème shuffle

Lors des tests de notre application nous avons remarqué une congestion des serveurs au moment du lancement des shuffles et des reduces. En effet, à partir d'un certain nombre de serveur, le serveur shuffle reçoit plus de connexion que ce qu'il peut traiter, entrainant une congestion du réseau et l'arrêt de l'application. Afin de détécter l'erreur nous avons tout d'abord retardé la connexion des reduces avec la methode sleep de Thread, après afin de resoudre ce problème nous avons adopté 2 méthodes, mais sans réussite.

— Nous avons essayé une méthode avec CallBack, c'est à dire qu'à chaque Shuffle on lui associe un CallBack qu'on passe en paramètre des Reduces, chaque Reduce doit attendre sur le CallBack du shuffle pour pouvoir lancer la demande de connexion. Néanmoins cette méthode nous créait un problème d'interblocage qu'on a pas pu résoudre.

— Nous avons aussi essayé de lancer un Thread à chaque réception d'une demande de connexion, néanmoins étant donné que le Thread devait modifier des variables locales, on devait alors les réenvoyer par socket vers le Serveur Shuffle ce qui congestionnait encore plus le réseau.