TELECOM-PARIS

SLR207

SLR207 Project

Auteur:
CORDEIRO, Franco

Email:
franco.cordeiro@telecom-paris.fr

May 3, 2020

Contents

1 Étape 1 **2**
1.1 Questions . . . 2
1.2 Implémentation . . . 3

2 Étape 2 **3**

3 Étape 3 **4**
3.1 Questions . . . 4

4 Étape 4 **4**

5 Étapes 5 à 12 **4**
5.1 Deploy . . . 4
5.2 Clean . . . 5
5.3 Phase de Map . . . 5
5.4 Phase de Shuffle . . . 5
5.5 Phase de Reduce . . . 6

6 Étape 13 **6**
6.1 Phase de Split . . . 6
6.2 Affichage des résultats . . . 7
6.3 Optimisations et sécurité . . . 7

1 Étape 1

1.1 Questions

1. La structure plus adéquat est HashMap, car c'est une structure qui fait un lien entre une clé hash, dans ce cas le mot, et une valeur, dans ce cas le nombre occurrences.

4. Le programme n'a pas fonctionné du premier coup, car il y a beaucoup de lignes sans mots et les lettres majuscule viennent avant les mêmes mots avec des lettres minuscules.

5. Les résultats pour le fichier deontologie_police_nationale.txt:

   de 98

   la 51

   police 38

   et 36

   des 33

6. Les résultats pour le fichier domaine_public_fluvial.txt:

   de 630

   le 429

   la 370

   du 347

   et 295

7. Les résultats pour le fichier sante_publique.txt:

   de 190889

   la 82599

   des 67813

   à 65546

   et 62702

8. Les résultats de chronométrage pour le fichier sante_publique.txt:

   Time to count: 730 ms

Time to sort: 47 ms

9. Les résultats pour le fichier qui contient une partie de l'internet:

   Time to count: 15865 ms

   Time to sort: 3964 ms

   the 720559

   to 466751

   and 456510

   a 430027

   of 394733

1.2 Implémentation

Le programme reçois comme argument le nom du fichier à faire la comptage. Si aucun argument est donné, il cherche le fichier input.txt.

Premièrement, il lit chaque ligne du fichier et partage en mot en retirant les points. Puis, pour chaque mot de la ligne, il la change en minuscule et utilise la structure HashMap pour sauvegarder les paires (mot, nombre occurrences.). Chaque fois qu'il rencontre un mot, il lit la structure pour savoir le nombre précédent, y ajoute un et le remplace dans la structure.

Après avoir tout compté, il transforme le map en stream et utilise les fonctions de triage pour premièrement trier par mot, et après pour trier par nombre d'occurrences. À la fin, il fait un print de toute la structure.

2 Étape 2

1. Nom court tp-1a222-01 : hostname -s

   Nom long tp-1a222-01.enst.fr : hostname -f

   Les noms court peuvent être obtenu par le nom de la salle et le numero de l'ordinateur dans la salle

2. On peut decouvrir l'ip avec le commande hostname -i à la ligne de commande. On peut aussi utiliser des sites en cherchant myIp en Google.

3. On peut découvrir l'ip avec le commande ping <nom de la machine>.

4. On peut découvir le nom avec le commande nslookup
5. Oui, les trois méthode fonctionnent.
7. On peut utiliser echo $((2+5)), expr 2 + 5, ou calc 2+5

   Tous les méthodes peuvent être utilisé sans appuyer plus d'une fois sur Entrée.
9. On peut faire ssh <nom de la machine> <commande>. Si on est déjà connecté à une machine de l'école, on n'a pas besoin d'un mot de passe.

3 Étape 3

3.1 Questions

1. chemin absolu: /cal/homes/cordeiro
3. df nom_du_fichier

   Ce fichier est stocké sur l'ordinateur local
4. df nom_du_fichier

   Ce fichier est stocké sur le serveur centrale de l'école
9. Commande pour copier le fichier:

   scp /tmp/cordeiro/local.txt tp-1a222-14:/tmp/cordeiro
10. ssh tp-1a222-14 scp /tmp/cordeiro/local.txt tp-1a222-05:/tmp/cordeiro

4 Étape 4

3. Commande pour exécuter le fichier .jar à distance:

   ssh tp-1a222-14 java -jar /tmp/cordeiro/Slave.jar

5 Étapes 5 à 12

5.1 Deploy

La fonction main de la classe Deploy commence par lire un fichier de nom tousMachines.txt. Ce fichier contient des lignes avec des lignes de format suivant:

<nom de la salle> <nombre de machines>

Puis, elle créer le fichier deploySuccess.txt qui contiendra les machines auxquels elle a réussi à envoyer le Slave.jar. Elle parcourt le fichier entrée et à chaque ligne crée des threads de nom GetHostName qui simplement envoi le commande gethostname et attend une réponse. Si la machine répond en moins de 30s, la fonction sauvegarde son nom dans le fichier de sortie et envoi le commande de copie du Slave.jar.

Pour envoyer le Slave.jar, elle utilise la classe thread CopySlave, qui simplement envoi un commande qui copie le Slave.jar seulement si le commande de création du répertoire a bien réussi.

5.2 Clean

La fonction main de la classe Clean a le même début de la classe Deploy. Elle lit le fichier tousMachines.txt et teste la réponse de toutes les machines. Après, elle envoi le commande rm -rf /tmp/cordeiro pour toutes les machines qui ont répondu. Ainsi, toutes les machines qui ont répondu effacent le répertoire de travail.

5.3 Phase de Map

Pour commencer la phase map il faut envoyer les fichier de split et le fichier machines.txt. Donc, la classe Master appelle la classe thread CopySplit qui crée le répertoire splits de la même manière utilisé précédemment et copie le split adéquat et le fichier machines.txt.

La classe Master utilise une fonction qui appelle ExecPhase pour lancer exécution d'une phase désirée. Cette fonction reçoit les paramètres nécessaires pour la fonction ExecSlave et lance les threads qui lancent le commande:

ssh <nom machine> java -jar /tmp/cordeiro/Slave.jar <phase> <arg>

La phase map envoi donc le número 0 pour indiquer phase map et comme argument le nom du fichier split.

La classe Slave commence la phase map en créent le répertoire maps avec un ProcessBuilder. Puis, elle ouvre le fichier du split et un fichier pour le résultat du map. Elle lit donc chaque ligne du fichier d'entrée et pour chaque mot ajoute une ligne au fichier de map avec le mot suivi de 1.

Le temps d'exécution de cette phase avec le fichier de test a été 4623 ms.

5.4 Phase de Shuffle

Pour lancer la phase de shuffle, la classe master appelle la fonction ExcecPhase avec la phase 1 et arg comme le nom du fichier map

La classe Slave commence cette phase en ouvrant le fichier de map et en créent le répertoire shuffles. Elle utilise donc la structure HashSet pour sauvegarder les hashs crées et pouvoir les envoyer plus tard. Puis, elle lit chaque ligne du fichier de maps, calcule le hash du mot, ouvre le fichier de shuffle en mode append et ajoute la ligne à ce fichier.

Après avoir créé tous les shuffles, elle lit le HashSet et le fichier machines.txt et calcule la machine qui doit recevoir ce fichier. Puis, il crée une un objet de la classe thread CopyShuffle qui crée le répertoire shufflesreceived si il n'est pas créé et envoi le fichier à la machine désignée.

Le temps d'exécution de cette phase avec le fichier de test a été 6664 ms.

5.5 Phase de Reduce

Pour lance la phase de reduce, la classe master appelle la fonction ExecPhase avec la phase 2 et arg vide.

La classe Slave commence en créent le répertoire reduces. Puis, elle lance la commande ls dans le répertoire shufflesreceived pour connaître tous les shuffles. Comme ces fichier sont montré en ordre alphabétique, les fichier de même hash vont être ensemble. Donc, la fonction lit le nom d'un fichier et fait la somme des occurrences d'un mot en utilisant un HashMap. Puis, elle lit tous les fichier suivant qui ont le même hash et ajoute à la somme. Quand elle rencontre un fichier avec un hash différent, elle écrit la somme finale au fichier de reduce et ouvre un nouveau fichier.

Le temps d'exécution de cette phase avec le fichier de test a été 4547 ms.

Donc, on voit que la classe qui prend plus de temps est la phase de shuffle, puisqu'elle exécute la plus grande quantité de commande ssh, qui est la partie qui prends plus de temps.

6 Étape 13

6.1 Phase de Split

La fonction main de la classe Master reçoit deux paramètres: le nombre de splits et le fichier pour faire la comptage. Le fichier ouvert par défaut est le fichier input.txt.

Premièrement, la fonction main calcule la taille des splits en octets. Puis, elle crée un fichier S<numéro du split>.txt et copie le numéro d'octets calculé du

fichier d'entrée dans ce fichier. Pour ne pas couper aucun mot, la fonction cherche le prochain caractère de espace dans le fichier, tout en copiant dans le fichier de sortie. Ça se répète jusqu'au dernier split qui copie le reste du fichier d'entrée.

6.2 Affichage des résultats

Après tout les phases, la fonction main de la classe Master utilise des threads de la classe GetReduces pour récupérer les résultats des reduces de chaque machine. Cette thread utilise un commande ssh <machine> ls /tmp/cordeiro/reduces pour connaître tous les reduces généré dans une machine. Après, elle utilise la commande cat pour récupérer le contenu de tous ces fichier, et ajoute chaque ligne à la structure ConcurrentHashMap.

Après avoir tout les résultats, la fonction main ordonne la structure et affiche le résultat dans le terminal.

Un comparateur de texte en-ligne a été utilisé pour vérifier que le résultat a été le même avec la solution séquentielle en utilisant le fichier domaine_public_fluvial.txt.

6.3 Optimisations et sécurité

Comme toutes les parties qui pourrait être faite en parallèle sont déjà concurrentes, il n'y a aucune optimisation nécessaire à faire pour pouvoir utiliser des fichier plus grand. Le projet peut-être compilé et exécuté avec le commande ant. Pour changer le nombre de splits, il faut changer la propriété "splits" et pour changer le fichier d'entrée, il faut changer la propriété "input". Les valeurs par défaut de ces propriétés sont 10 et vide. Il a aussi la cible clean-build pour nettoyer les fichier générés par la compilation et l'exécution, et une cible pour chaque classe principale.

Comme le master peut-être appelé indépendamment du Deploy, au début de la fonction main, il vérifie encore quelles sont les machines qui sont toujours connecté avant de les envoyer les splits.