06/11/2021

Compte rendu du TP Map Reduce

FERHATI AMINA, LAIOUA AMINE
MASTER 2 DATASCALE, UNIVERSITE PARIS SACLAY

I. Conception

Question 1 :

Pour répondre à cette question. On a opté pour la démarche suivante :

- Le mapper :
- Il permet de sélectionner les champs userTel et userDept
- Il compare userDept à la valeur '78'
- Si le numéro du département est ‘78’, alors, Il sélectionne userVille
- Il attribue à chaque ville le nombre 1 (userVille,1)
- La sortie finale du mapper est : une série de tuples selon l'ordre d'arrivée des fichiers (userVille,1, userTel) puis (userVers).
- Le reducer :
- il reçoit les tuples de users d'abord et il les stocke dans le dictionnaire “Users”
- Dès l'arrivée du premier champs userVers, il va le comparer aux userTel enregistrés dans le dictionnaire “Users”
- Si les deux champs sont égaux, alors, on stocke la ville du user dans un dictionnaire “Villes” qui a comme clé la ville et comme valeur son occurrence.
- La sortie finale du reducer est : userVille, son occurrence qui doit être supérieur à 2.

Question 2 :

Dans le cas de jointure entre une petite table et une grande table nous avons opté pour la jointure map-side, la jointure côté mappeur est plus rapide car elle n'a pas à attendre que tous les mappeurs aient terminé comme dans le cas du reducer.

- Dans le fichier map_petit.py :
- Le mapper permet de sélectionner le département “78”
- Il permet de stocker dans un dictionnaire ”ville” le userTel et userVille correspondant à ce département
- Il compare chaque userTel du Dictionnaire “ville” au userVers du fichier calls ensuite il envoie le couple (userVille, 1) au reducer.
- Dans le fichier reduce_petit.py :
- Les villes sont triées à la sortie mapper (l’entrée de reducer).
- Il permet de calculer le nombre d'apparences de chaque couple (ville,1) pour les différentes villes.

Question 3 :

Le combinateur MapReduce améliore la performance globale du reducer en résumant la sortie du Mapper. Dans la première solution on a deux sorties : (UserVille,1,userTel) et (UserVers,1) on peut faire un “combine” pour calculer le nombre d'occurrence pour chaque userVers et éliminer ses copies et dans le reducer nous allons sommer le nombre d'occurrences de userVers pour chaque ville. Ça aurait été mieux d’utiliser le combine dans le cas où on a plusieurs Mappers (un vrai cluster), car ça permet de les regrouper afin de les envoyer au reducer.

Question 4:

La commande sort permet d'ordonner les données, chaque entrée du reducer est triée par clé.

- On n'a pas eu besoin de cette commande pour la première solution (map.py et reduce.py).
- Pour la deuxième solution, on a eu besoin de l'utiliser afin de trier les villes par ordre alphabétique et compter le nombre d'occurrence de ces dernières dans reducer_petit.py
- Pour la solution combine implémentée, la commande sort a joué un grand rôle, car en premier lieu, elle a permis de trier les numéros UsersVers afin de compter leur occurrence puis elle est utilisé pour trier dans l'ordre inverse les résultats du combine afin de mettre les couples qui commences par (userVille, userTel) en premier puis (userVers, occurrence) en deuxième lieu, ce qui a permis de sauvegarder les couples (userVille, userTel) dans un dictionnaire afin de minimiser le nombre d'éléments stockés.

II. Partie technique

Résultat du map.py et reduce.py:

```
C:\Users\CBS Compter\Downloads\MAP>type users.txt calls.txt | python map.py |  python reduce.py
users.txt


calls.txt

Versailles,5039
```

Résultat du map_petit et reduce_petit:

```
C:\Users\CBS Compter\Downloads\MAP>type users.txt calls.txt | python map_petit.py | sort | python reduce_petit.py
users.txt


calls.txt

Versailles      5039
```

Résultat du map.py, combine.py et reduce_combine.py:

```
C:\Users\CBS Compter\Downloads\MAP>type users.txt calls.txt | python map.py | sort | python combine.py | sort /R | python reduce_combine.py
users.txt


calls.txt

Versailles,5039
```