TP MAP REDUCE

Table of Contents

Cet exercice propose de chercher comment formuler un équivalent d'une requête SQL en *map* et *reduce*. C'est ce processus qui est au coeur de *Pig* ou *Hive*. Le *MapReduce* est un paradigme de programmation qui vous obligent à avoir une tournure d'esprit vous permettant de résoudre des problèmes de big Data.

Les exemples qui suivent sont écrits en Python. Même si vous ne connaissez pas Python, ces exemples ci-dessous associés aux commentaires sont suffisant pour vous permettent de comprendre et rédiger ce qu'on vous demande. Il vous est facile de trouver des tutoriels si nécessaire. Vous pouvez utiliser votre langage préféré comme Java.

Afin de vous éviter d'implémenter sur un vrai cluster avec toute la complexité inhérent, nous allons utiliser les possibilités de "streaming" du système d'exploitation Unix. Mac à aussi cette possibilité. Pour Windows, vous avez des outils comme Cygwin ou des machines virtuelles, vous permettant de faire la même chose. Le streaming est un moyen pour vous permettre de mettre au point vos algorithmes avant de les exécuter sur un vrai cluster comme le font les vrais expérimentateurs.

Le principe du streaming sous Unix est de combiner des opérateurs via des "pipes" (tube). Par exemple `ls | wc` lance deux processus, le premier envoie sur sa sortie standard (qui normalement est connectée vers la console) dans un tube. Ce tube est connecté sur *l'entrée standard* de `wc` (qui normalement est connecté au clavier). L'un produit des données, et la seconde les consomme. Ce principe est intégré dans le Framework Hadoop. Nous nous contenterons du niveau Shell pour l'expérimentation ici.

Nous vous fournirons un lot de données pour votre mini-projet qui se trouve sur *Persee* du nom `appels.tgz` .

Le but du mini-projet est de calculer l'equi-jointure entre une collection A et B. Sous Unix, vous devrez le tester suivant une expression Shell du type:

```
cat A B | map.py | sort | reduce.py > result.txt
```

Le rendu pour le Vendredi 5 Nov

On vous demande d'écrire la requête suivante en map-reduce avec un minimum de Job comme pourrait le faire Hive.

```
-- calls(de, vers, duree)
-- users(nom, prenom, tel, dept, ville)

SELECT ville, count(*)
FROM users, calls
WHERE dept = 78
AND tel = vers
GROUP BY ville
HAVING count(*) > 2;
```

Vous devez rendre:

- Un mini compte rendu expérimentation d'une page expliquant comment vous avez résolu le problème et justifiant vos choix. Une seconde page plus technique qui contient une description de votre implémentation et comment on procède pour exécuter votre code sur votre OS/environnement (à préciser). L'enseignant pourra être amener à exécuter votre code en copiant les lignes de commandes que vous aurez proposé, il suivra vos instructions à la lettre. La note de cette partie tiens compte de la clarté de vos explications.
- une archive compressée contenant le `result.txt` , la copie d'écran d'une trace d'exécution et les codes, par exemple `map.py` et `reduce.py` . Pour le `result.txt` , il doit apparaître au moins 3 tuples. Vous pouvez changer la valeur de sélection de `dept` , et du `Having count` avec une valeur minimum de 2.
- réponse à la question, si je dois implémenter une jointure entre une petite table qui peut tenir dans chaque *mappeur* et une grosse table, est ce possible et comment procéder modifier votre code map-reduce (vous appellerez ce code, `map_petit.py` et `reduce_petit.py` ?
- réponse à la question, est il utile d'utiliser un `combine` ? dans les deux cas, vous devez justifier. une proposition de code est un plus.
- réponse à la question, quel est le rôle de la commande `sort` ?

Le rendu doit se faire sur le site *Persee*. Le travail est à faire en binôme. Les deux binômes déposera sur leur compte *Persee*, il mentionnera le NC du binôme. Tout travaux trop ressemblant entre équipe seront détectés par le site et donc noté selon leurs mérites.

Afin de vous préparer, nous allons étudier le *word count* suivant le principe énoncé précédemment.

Word Count

1. Les indentations de Python Python est basé sur les indentations. Sur l'exemple ci-dessous, les points de <4> à <7> seront exécutés à chaque itération du `for` <3> à cause de l'indentation. Le point <7> sera exécuté pour chaque itération de <6>
2. Le code du Map

```
#!/usr/bin/env python  (1)
import sys (2)

for line in sys.stdin: (3)
    line = line.strip() (4)
    words = line.split() (5)

    for word in words: (6)
        print '%s\t%s' % (word, 1) (7)
```

1. On indique au Framework qu'il s'agit d'un programme Python
2. On a besoin du *module* `sys` pour pouvoir lire sur l'entrée standard
3. Hadoop fournit les données sur l'entrée standard. La boucle for permet de lire, ligne par ligne.
4. La fonction `strip()` permet d'effacer entre autre le caractère retour à la ligne présent dans l.
5. Ici, la ligne est découpé en mots (séparé par un espace ou tab) placé un tableau. Exemple: `a="a;b;cd".split(',')` , on aura `a=['a','v,'cd']`
6. Pour chaque mot du tableau `words` , on va l'envoyer vers le reducer.
7. Pour cela, le 'print' envoie sur la *sortie standard*. On utilise comme délimiteur le tab (i.e. '\t'). Ce délimiteur permet de séparer la clé à gauche, de la valeur. Les '%s' de la ligne correspond au formatage de la ligne comme en Java ou C.

    1. Le code du reducer (version sans tableau)

```
#!/usr/bin/env python
import sys

current_word = None
current_count = 0
word = None (1)

for line in sys.stdin: (2)
    line = line.strip()
    word, count = line.split('\t', 1) (3)

    try: (4)
        count = int(count) (5)
    except ValueError:
        continue (6)

    # this IF-switch only works because Hadoop sorts map output
    # by key (here: word) before it is passed to the reducer
    if current_word == word: (7)
        current_count += count
    else: (8)
        if current_word: (9)
            print '%s\t%s' % (current_word, current_count)
        current_count = count (10)
        current_word = word

if current_word == word: (11)
    print '%s\t%s' % (current_word, current_count)
```

1. Ici on a initialiser la variable à une valeur spéciale `None` qui est équivalent au `NULL` du SQL.
2. Remarquer qu'ici aussi, le framework donne les lignes en provenance de *l'entrée standard*. Les lignes sont triées par hadoop, donc les même clés sont regroupées !
3. Séparation de la partie clé, du reste. Si on avait pas mis le paramètre 1, toute la ligne sera décomposée suivant le séparateur '\t'
4. Comme dans tout langage, on peut attraper les exceptions, notamment lorsqu'on va essayer
5. de convertir la chaîne de caractères en un entier. Si l'exception est levé (donc, ça n'est pas un entier), on reste silencieux
6. en continuant au début de la boucle
7. Il faut avoir à l'esprit qu'Hadoop tri la sortie du Map. Donc, si c'est la même clé, on ne fait qu'incrémenter le compteur.
8. A l'inverse, si la clé a changé, alors on doit produire une sortie
9. Méfiance, pour éviter la première fois, il faut que `current_word` contienne une valeur ! Dans ce cas, ligne suivante, on produit une valeur.
10. On repart, avec le mot suivant
11. Il faut pas oublier le dernier mot !

    1. Version du reducer avec tableau (`reducer2.py`)

Afin de vous préparer pour les exercices demandés par la suite, voici une version équivalente, mais utilisant la structure *dico*. Cette version diffère car elle stocke tous les tuples en mémoire du *reducer* avant de produire le résultat.

PYTHON

```
#!/usr/bin/env python
import sys
word2count = {} (1)

# input comes from STDIN
for line in sys.stdin:
    line = line.strip()
    word, count = line.split('\t', 1)
    try:
        count = int(count)
    except ValueError:
        continue

    if word in word2count: (2)
        word2count[word] += count
    else:
        word2count[word] = count

for word in word2count.keys(): (3)
    print '%s\t%s'% ( word, word2count[word] )
```

1. Ici on définit un tableau associatif (appelé aussi structure dictionnaire). L'idée ici est de stocker tous les tuples
2. Si la clé est déjà connue, on ne s
3. Enfin, on produit le résultat en sortie

Tester ces programmes sur le cluster

Si vous souhtez tester ce programme, il y a

SHELL

```
cat pg5000.txt | map.py | sort | reduce.py > resultat.txt
```

Notez qu'ici la commande `sort` s'applique sur chaque ligne en triant suivant l'ordre alphabétique. Si vous devez trier une clé suivant l'ordre numérique, placer l'option `-n`, i.e. `sort -n`.