Bases de Données Large Echelle - TME 8 et 9

Rédigé par : Ayoub Jhabli
A l'intention de : Hubert Naacke
Rendu le : 17 december 2023

1 Introduction

Les objectifs des TMEs 8 et 9 étaient, en premier lieu, de trouver une méthode pour déterminer les composantes connexes d'un graphe. Ensuite, de reproduire le même travail sur un graphe partitionné (initialement avec 2 partitions), puis avec plusieurs partitions sur des données réelles tout en parallélisant le traitement en utilisant les outils de Spark. Les deux derniers exercices étaient plutôt des questions ouvertes à aborder après avoir accompli les trois premiers.

2 Exercice 2 : calcul parallèle des composantes

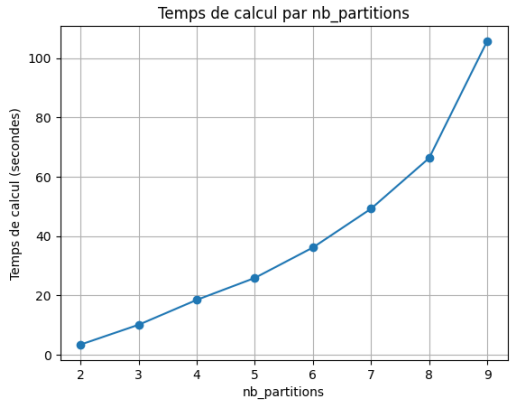
L'exercice 2 était plutôt simple puisqu'on avait déjà implémenté la majorité de ce dont nous avions besoin dans l'exercice 1.

La nouveauté réside dans l'utilisation de la fonction **mapPartitionsWithIndex**, qui renvoie un nouveau RDD en appliquant notre UDF **calculer_composantes** à chaque partition de ce RDD, tout en conservant l'index de la partition d'origine. Finalement, le calcul de *comp_edges* (les paires de composantes) et de *Global_comp* (les composantes globales) est similaire à celui de la question 1)d) dans sa logique.

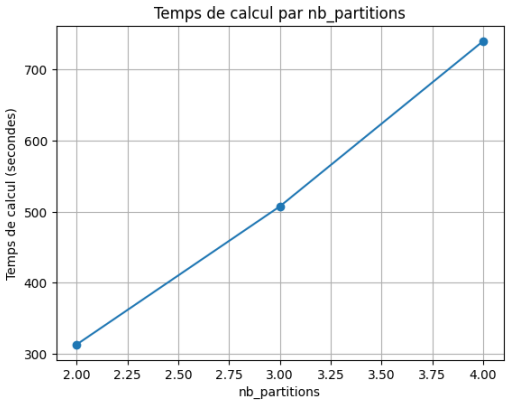
3 Exercice 3 : Calcul de composantes à grande échelle

Le but de l'exercice était de reproduire ce qui est fait dans l'exercice 2, mais plutôt sur des données réelles à grande échelle et de trouver en quelque sorte le nombre optimal de partitions en termes de temps d'exécution pour les deux tables **g10k** et **g1M**. La fonction du code est le suivant :

```
def calculer_composantes_nPartitions(df, nb_partitions=4, v=False):
    df_pn = repartition_dataframe(df, nb_partitions)
    df_composantes_pn = calculate_components_in_partitions(df_pn, v)
    df_composantes_pn_pivot = pivot_on_part_id(df_composantes_pn, v)
    df_Comp_edges = create_comp_edges(df_composantes_pn_pivot, nb_partitions, v)
    df_Global_comp = calculate_global_components(df_Comp_edges, v)
    df_composantes_gl = join_global_components(df_composantes_pn, df_Global_comp, v)
    return df_composantes_gl
```



(a) Temps d'exécution en fonction du nombre de partitions pour g10K.



(b) Temps d'exécution en fonction du nombre de partitions pour g1M.

Comme on peut le constater dans le code, le processus commence par un partitionnement de la table, suivi du calcul des composantes pour chaque partition. Ensuite, une opération de pivot est réalisée sur la colonne *partID* afin d'obtenir un DataFrame avec les composantes pivotées en colonnes. On procède ensuite au calcul des *Comp_edges*, puis des *Global_comp*, et enfin, une jointure est effectuée pour obtenir le résultat final.

On déduit aussi des deux graphes ci-dessus que le nombre de partitions optimal est: 2. Pour g1M, le temps d'éxecution était assez grand, j'ai essayé qu'avec $n\_partitions \in [2, 4]$

4 Exercice 4 : Choix du minimum pour le numéro de composante

Le but de cet exercice est de proposer une solution pour calculer les composantes d'un graphe supposant que ces noeuds sont numérotés de 1 à n, telle que le numéro de composante soit égal au numéro du plus petit noeud appartenant à cette composante. La fonction du code est le suivant :

```
def calculer_composantes_graphe_numerote(df, v=False):
    df_composantes_pn = calculate_components_partitioned(df)
    df_composantes_pn_nmin = calculate_min_n(df_composantes_pn)
    df_composantes_pn = merge_min_n(df_composantes_pn, df_composantes_pn_nmin, v)
    df_composantes_pn = find_joins(df_composantes_pn, v)
    return df_composantes_pn
```

On commence par un calcul de composantes par partition puis on agrège les numéros de nœuds minimums dans chaque composante pour chaque partition. On fait une jointure entre les deux tables calculées. Finalemnt la fonction **find_joins** recherche les points de jonction entre partitions en se basant sur les numéros de nœuds minimums en utilisant une autre fonction **update_composantes_pn** qui actualise les composantes en fusionnant celles identifiées lors de la recherche de jonctions, et ce processus est répété jusqu'à atteindre la stabilité. Avant de lancer cette fonction de calcul de composantes, il ne faut pas oublier de numéroter les noeuds comme il est indiqué dans l'exercice

5 Exercice 5 : Calcul de composantes avec seuillage sur le poids des arcs

Je n'ai pas réussi à trouver une solution optimale. Il est certain que l'ensemble des nœuds avec un poids supérieur à 100 est inclus dans l'ensemble des nœuds avec un poids supérieur à 90. Par conséquent, déterminer les composantes de l'ensemble des nœuds avec un poids entre 90 et 100 est probablement utile. Ensuite, on peut rechercher les nœuds communs entre les composantes de l'ensemble des poids > 100 et celui entre 90 et 100, puis fusionner ces composantes. Cependant, cette méthode est très coûteuse en temps d'exécution. De plus, si l'on suppose que deux nœuds appartiennent à deux composantes différentes dans l'ensemble entre 90 et 100, et que ces deux nœuds appartiennent à une certaine composante dans l'ensemble > 100, fusionner les trois composantes en un seul coup suppose que le traitement ne sera pas parallèle. Sinon, on obtiendra deux composantes distinctes, ce qui est incorrect puisqu'un nœud appartient aux deux composantes.

6 Conclusion

Ces deux TME étaient très intéressants, car le traitement en parallèle des données à grande échelle est une problématique récente en cours de développement.