Projet Spark

Réalisé par Tom Careghi, Nadejda Dorosenco, Erwan Duprey, Juan MAUBRAS
Etudiants en 4IABD1

L'écriture en Spark est particulière. La syntaxe suivante est régulièrement adoptée dans notre projet:

```
df_regrouped.coalesce(1).write.mode("Overwrite").format("csv").save('data/exo2/aggregate')
```

Coalesce(1) permet de préciser l'écriture du fichier en une fois, ce que l'on aura tendance à trouver plus pratique que de se trouver avec de multiples dossiers. Nous avons aussi appris que le code est prévu pour remonter une erreur si nous ne précisons pas que l'on "Overwrite" un fichier. C'est très certainement prévu parce que les systèmes distribués ne sont pas performants pour modifier un fichier et qu'il vaut mieux vérifier ce que l'on écrase avant de réécrire. Spark étant souvent lié à un système de fichier distribué, il est probable que Spark ait la même façon d'agir.

- Python_udf = --- 125.18302869796753 seconds ---
- Scala_udf = --- 106.13712906837463 seconds ---
- No_udf = --- 103.75592088699341 seconds ---

Le résultat est sans appel, Python_udf est beaucoup plus lent mais de manière générale, le programme est lent. 103 secondes en pyspark par exemple c'est long mais il faut remettre les choses dans leur contexte : le temps calculé est le temps de lecture, de calcul (exécution des fonctions) et d'écriture. La lecture et l'écriture dépendent énormément de la machine sur laquelle c'est exécuté; les composants, la mémoire RAM qui était vide ou non, etc… . Mais cela dépend aussi de la configuration de l'environnement, ici nous utilisons un ordinateur sous windows et nous y avons mis une VM Linux que nous manageons via WSL.
Cependant nous somme obligés d'écrire le fichier pour pouvoir lancer les actions vu que le code est lazy.
Nous avons essayé d'utiliser d'autres fonctions pour déclencher comme le collect() mais nous avons rencontré un problème qui est le manque de mémoire en RAM vu que le collect() met tout le dataframe en RAM.
Cependant si nous ne lançons qu'une lecture et qu'une écriture, il met 101 sec, donc si on le soustrait, on obtient environ des temps comme ça (meme si pour l'écriture, il y a une colonne en moins que lors de la réél écriture):

- Python_udf = --- 24 seconds ---
- Scala_udf = --- 5 seconds ---
- No_udf = --- 2 seconds ---

Ces résultats sont tout de suite plus raisonnables.

A propos du Python UDF, bien que l'exercice nous a permis de savoir l'utiliser, cet exercice reste moins "professionnel" car le temps d'utilisation peut s'avérer très long et n'est donc à utiliser qu'en dernier recours par rapport aux autres fonctions qu'offre la bibliothèque Spark.

Dans ces exercices nous avons appris à utiliser des windows function et des UDF. Puis nous avons mis en place des tests unitaires et des tests d'intégration, ce qui nous a permis de vérifier la solidité de notre code et de repérer des erreurs s'il y en avait.