Cours de Big Data

5A – Polytech Paris-Sud – 2023-2024

TP-1 Part-2 : Algorithmique et programmation distribuée en Spark

Gianluca Quercini & Stéphane Vialle

Les exécutions de programmes Spark de ce TP se dérouleront sur l'environnement distribué d'un *cluster Spark-HDFS* du *Data Center d'Enseignement* de CentraleSupélec. Pour les détails de mise en œuvre et d'accès aux fichiers distribués, voir la partie 1 du TP.

Exercice 1 : Calcul de moyennes

On considère différents fichiers CSV contenant des mesures de températures *« Année, Mois, Jour, heure, minute, seconde, température »*, et stockés dans le répertoire **hdfs://sar01:9000/data/temperatures/** d'un système de fichier distribué HDFS.

- *temperatures_86400.csv* : une mesure par jour pour les années 1980 - 2018.
- *temperatures_2880.csv* : une mesure toutes les 2880 secondes pour les années 1980 - 2018.
- *temperatures_86.csv* : une mesure toutes les 86 secondes pour la seule année 1980.
- *temperatures_10.csv* : une mesure toutes les 10 secondes pour les années 1980 - 2018.

On souhaite arriver à générer des couples *(Année, TempératureMoyenne)* par une approche Map-Reduce en Spark.

Question 1.1 : Première implémentation.

- Copiez le fichier ***~cpu_vialle/DCE-Spark/template_temperatures.py*** sur votre compte de TP et créez le fichier ***avg_temperatures_slow.py***
- Complétez ce template en précisant les chemins HDFS du répertoire des fichiers d'entrée, et du répertoire des fichiers de sortie (la racine de votre espace propre sur HDFS) :

```
#map-reduce computation
def avg_temperature_slow(theTextFile):
  avg = theTextFile                                          \
            .map(lambda line: line.split(","))               \
            .map(lambda term: (term[0],   [float(term[6])] ))  \
            .reduceByKey(lambda x, y: x+y)                   \
            .mapValues(lambda lv: sum(lv)/len(lv))
  return avg

#main code
input_hdfs_file_name = ……     #TO COMPLETE – see details in the real python source file
output_hdfs_file_name = ……  # TO COMPLETE – see details in the real python source file
sc = SparkContext()

input_text_file = sc.textFile(input_hdfs_file_name)
results = avg_temperature_slow(input_text_file)
results.saveAsTextFile(output_hdfs_file_name)
```

- Testez votre programme Spark sur le fichier *temperatures_86400.csv*, et vérifiez le contenu du fichier de sortie :

```
spark-submit --master spark://sarXX:7077
             avg_temperatures_slow.py  temperatures_86400.csv
```

A titre de comparaison, voici les paires clé-valeurs que vous devriez obtenir pour les années 2013-1018 sous la forme (année, moyenne) :

```
(u'2013', 8.159945205479453)        (u'2016', 8.337049180327872)
(u'2014', 7.41616438356164)         (u'2017', 7.345287671232872)
(u'2015', 7.788328767123292)        (u'2018', 8.23512328767123)
```

- Complétez le tableau ci-dessous pour 3 fichiers d'entrée différents :

(short) Input file name	*86400*	*2880*	*86*	*10*
File size (MB)	0,36	10,5	9,0	3000,0
Exec time (s)				

Pour récupérer les temps d'exécutions vous chercherez une ligne dans la sortie de Spark contenant quelque chose comme :

INFO DAGScheduler: Job 0 finished: runJob at SparkHadoopWriter.scala:78, took 3.478220 s

- Comment justifiez-vous ces temps d'exécution, sachant que les fichiers *temperatures_2880.csv* et *temperatures_86.csv* ont une taille comparable ?

Question 1.2 : deuxième implémentation

- Copiez votre fichier ***avg_temperatures_slow.py*** dans un fichier ***avg_temperatures_fast.py***
- Cette fois-ci complétez la fonction ***avg_temperature_fast*** en modifiant les 3 lignes marquées « TO DO » ci-dessous, afin d'implanter un calcul map-reduce parallélisant mieux les calculs ET diminuant le trafic réseau entre les *spark-executors* :

```
def avg_temperature_fast(theTextFile):
    avg = theTextFile                                \
            .map(lambda line: line.split(","))       \
            .map(lambda term: ……)                    \      TO DO
            .reduceByKey(lambda x, y: ……)            \      TO DO
            .mapValues(lambda x: ……)                        TO DO
    return avg
```

- Dans le code principal (en bas du fichier) n'oubliez pas d'appeler maintenant la fonction ***avg_temperature_fast*** (au lieu de ***avg_temperature_slow***).
- Complétez le tableau ci-dessous pour **4** fichiers d'entrée différents :

(short) Input file name	*86400*	*2880*	*86*	*10*
File size (MB)	0,36	10,5	9,0	3000,0
Exec time (s)				

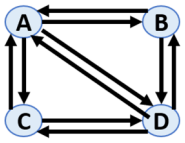
- Comparez globalement les temps d'exécution obtenus à ceux de la question 1.1. Où pensez-vous passer du temps avec cette version du code ? Où pensez-vous en avoir gagné ?

Exercice 2 : Recherche d'amis communs dans un réseau social

On considère un graphe de réseau social encodé dans un fichier texte.

- Une ligne du fichier est une liste d'identificateurs séparés par des virgules : A, B, C, D qui signifie que A a pour amis B, C et D.
- L'ensemble des lignes du fichier décrivent les relations « a pour ami » du graphe. Ex :

A, B, C, D
B, A, D
C, A, D
D, A, B, C



- On supposera les relations symétriques : si on A, B alors on a B, A.

On souhaite obtenir la liste des amis communs à des couples d'individus, sous forme d'un RDD de paires clés-valeurs :

((A, B), [D])
((A, C), [D])
((A, D), [B, C])
((B, C), [A, D])
((B, D), [A])
((C, D), [A])

Rmq : On ne veut représenter chaque couple qu'une fois (comme ci-dessus). Si on présente les amis du couple (A, B), alors on ne représentera PAS le couple (B, A).

On utilise les fichiers suivants du répertoire **hdfs://sar01:9000/data/social-network/** :

sn_tiny.csv	: Réseau social de petite taille, pour tester l'implémentation.
sn_10k_100k.csv	: Réseau social avec 10^4 individus et 10^5 liens environ.
sn_100k_100k.csv	: Réseau social avec 10^5 individus et 10^5 liens environ.
sn_1k_100k.csv	: Réseau social avec 10^3 individus et 10^5 liens environ.
sn_1m_1m.csv	: Réseau social avec 10^6 individus et 10^6 liens environ.

Questions 2.1 : conception d'une solution

- Récupérez et complétez le *template* suivant :

  `~cpu_vialle/DCE-Spark/template_common_friends.py`

- Concevez une solution incluant un reduceByKey(…)
- Concevez une solution incluant un groupByKey()
- Testez et validez vos solutions sur le fichier *sn_tiny.csv*.

Question 2.2 : tests et mesures de performances

- Essayez maintenant vos solutions sur les quatre autres fichiers tests, et notez les temps d'exécution.
- Rassemblez vos mesures dans un tableau (**onglet Q1** du fichier **TP2-Perf.xlsx**) :

Input file	sn_tiny	sn_10k_100k	sn_100k_100k	sn_1k_100k	sn_1m_1m
File size (MB)					
groupByKey (s) (q. 2.2)					
reduceByKey (s) (q. 2.2)					
Min nb of friends (q. 2.3)					
Max nb of friends (q. 2.3)					
Average nb of friends (q. 2.3)					
Estimated nb of intermediate pairs (q. 2.4)					
Measured nb of intermediate pairs (q. 2.5)					

Question 2.3 : calcul des degrés min, max et moyens des nœuds du graphe

- Ecrivez un programme Spark qui calcule en une seule exécution les degrés minimum, maximum et moyens des nœuds du graphe du réseau social (le degré d'un nœud sera son nombre d'amis).

  Rmq : restez dans un RDD le plus longtemps possible et calculez les valeurs min, max et moyenne dans le RDD (ne quittez pas le RDD pour les calculer en python sur des listes classiques).
- Appliquez votre programme aux 5 fichiers de tests pour calculer les nombres min, max et moyens d'amis d'un individu.
- Complétez le tableau de la question 2.2 avec les nombres min, max et moyens d'amis d'un nœud.

Question 2.4 : estimation du nombre de paires intermédiaires

Comme en TD, on supposera que chaque nœud possède un nombre d'amis égal au nombre d'amis moyen.

- Déduisez le nombres de paires intermédiaires (de type ((A,B), X)) générées par votre code de recherche des amis communs, à partir des nombres de nœuds et des nombres moyens d'amis par nœud obtenus à la question précédente.
- Complétez le tableau de la question 2.2 avec le nombre de paires intermédiaires

Question 2.5 : mesure du nombre exact de paires intermédiaires

Les différents nœuds du graphe n'ayant pas un nombre d'amis égal au nombre moyen d'amis par nœud, une mesure du nombre de paires intermédiaires sera plus exacte que l'estimation calculée à la question précédente.

- Modifiez votre code Map-Reduce pour calculer le nombre de paires intermédiaires générées et obtenir ainsi une mesure fiable.
- Complétez le tableau de la question 2.2 avec le nombre de paires intermédiaires mesuré.

Question 2.6 : analyse des performances

Pour les deux solutions (avec un groupByKey, et avec un reduceByKey) tracez le temps d'exécution de votre code de recherche des amis communs en fonction :

1. de la taille du fichier de données,
2. du nombre moyen d'amis par nœud,
3. de l'estimation (calculée) du nombre de paires intermédiaires,
4. du nombre exact (mesuré) du nombre de paires intermédiaires.

Tracez 4 figures différentes.

Quelle figure montre la progression la plus monotone/régulière du temps d'exécution ? Quelle variable caractérise ainsi le mieux le temps d'exécution de votre programme ?

Exercice 3 : Calcul de moyennes et d'écarts types sur de grandes séries de données

On utilise les mêmes fichiers de relevés de températures que dans l'exercice 1, mais on veut maintenant arriver à des triplets de valeurs : *(Année, TempératureMoyenne, EcartType)*.

On peut exprimer l'écart type de N valeurs x_i (avec : $0 \leq i < N$) selon la définition suivante :

$$\sigma = \sqrt{\overline{x^2} - \bar{x}^2} = \sqrt{\frac{\sum_{i=0}^{i=N-1}(x_i{}^2)}{N} - \left(\frac{\sum_{i=0}^{i=N-1} x_i}{N}\right)^2}$$

Cette expression de l'écart type est beaucoup plus « parallélisable » que l'expression habituelle, et nous l'adoptons pour ce TP. Toutefois … elle est numériquement instable ! et donc n'est pas non plus celle utilisée dans les vrais codes de calculs parallèles.

Question 3.1 : implantation et validation

- Proposez une approche Map-Reduce et un code Spark répondant au problème rapidement, pour de grandes séries de données.
- Recopiez à nouveau le fichier **~cpu_vialle/DCE-Spark/template_temperatures.py** pour créer maintenant le script Python ***stddev_temperatures.py***
- Implantez votre solution, puis testez la d'abord sur le fichier *temperatures_86400.csv*. Vérifiez le contenu du fichier de sortie.

  A titre de comparaison, voici les paires clé-valeurs que vous devriez obtenir pour les années 2013-1018 sous la forme (année, (moyenne, écart type)) :

```
(u'2013', (8.159945205479453, 12.095467213807215))
(u'2014', (7.41616438356164, 12.000489970606273))
(u'2015', (7.788328767123292, 11.422338877224695))
(u'2016', (8.337049180327872, 11.508239269066483))
(u'2017', (7.345287671232872, 11.692122262422554))
(u'2018', (8.23512328767123, 11.458240836489342))
```

Question 3.2 : mesures de performances

- Exécutez votre programme pour compléter le tableau ci-dessous :

Input file name	*86400*	*2880*	*86*	*10*
File size (MB)	0,36	10,5	9,0	3000,0
Exec time (s)				

- Comparez les temps d'exécution avec ceux de l'exercice 1. Qu'en déduisez-vous ?

Exercice 4 : Variation du nombre de cœurs et de *Spark executors*

Reprenez le code du calcul de la moyenne et de l'écart type de l'exercice 3.

Vous allez refaire le test de la question 3.2 sur le fichier de données *temperatures_10.csv* de 3 Go, mais en faisant varier le nombre de ressources de calcul :

- La configuration du cluster Spark du TP impose à Spark de n'utiliser qu'un seul cœur (et seulement 1Go de RAM) par *Spark executor*, et par défaut de lancer 1 *Spark executor* par nœud de calcul. Il y a 15 nœuds de calcul dans le cluster, donc par défaut Spark lance 15 processus *Spark executor* : 1 par machine, utilisant chacun 1 cœur et 1Go de RAM.

- Avec le gestionnaire de cluster natif de Spark (mode *Standalone*) on ne peut pas contrôler directement le nombre de *Spark executor* lancés par Spark, mais on peut contrôler le nombre total de cœurs utilisés. Par exemple :

```
spark-submit --total-executor-cores 5
             --master spark://sar01:9000 myprog.py ….
```

  cette commande impose de n'utiliser que 5 cœurs au total, donc 5 *Spark executors* dans notre configuration, donc 5 nœuds de calcul seulement.

Question 4.1 : Exécutez le calcul de la question 3.2 pour compléter les mesures et les courbes du fichier Excel TP1-Perf.xlsx (faites les expériences manquantes de 3 à 15 nœuds).

Question 4.2 : Analysez cette courbe de performances. Vous parait-elle logique (pourquoi) ?

Consignes pour le CR

- **CR en monôme ou bien en binôme pour le 24/10/23 23h59**
- **Indiquez clairement vos nom, prénoms, date et titre de TP**

- **8 pages maximum (« tout inclus ») – police de caractères 11pt**
- Mettez les extraits de codes python répondant aux questions des exercices
- Mettez vos explications sur vos codes, montrant que vous avez compris ce que vous avez fait !
- Mettez les mesures de performances (tableaux, courbes… à vous de décider)
- Mettez les réponses aux différentes questions.

- Vous ne rendrez QUE vote CR (pas de fichiers pythons, pas de fichiers Excel…).
- **Envoyez votre CR par email à :**
  - **Gianluca.Quercini@centralesupelec.fr**

  ET
  - **Stephane.Vialle@centralesupelec.fr**