TP2 : Spark Core/RDD

© Mourad Ouziri
mourad.ouziri@u-paris.fr

Programme-objectifs :

- Traitement de données massives.
- Programmation Big Data avec l'API Spark *Core*.

Documentation :

Spark Scala API doc : https://spark.apache.org/docs/latest/api/scala
Scala API doc : https://www.scala-lang.org/api/2.11.10/#package
Java API doc : https://docs.oracle.com/javase/7/docs/api/

1. Démarrage de l'environnement Spark

Manipulation 1: Télécharger Apache Spark https://spark.apache.org/downloads.html puis décompresser l'archive dans un répertoire local de travail.
Manipulation 2: Démarrer (dans un terminal de lignes de commandes) le *master* de Spark : *./bin/spark-class org.apache.spark.deploy.master.Master*
Manipulation 3: Consulter la page Web du Master, accessible sur le port 8080.
Manipulation 4: Démarrer trois Workers (dans trois terminaux de lignes de commandes) : *./bin/spark-class org.apache.spark.deploy.worker.Worker spark://{host}:{port} -m 1G -c 2*
Manipulation 5: Démarrer le client interactif de Spark (dans un terminal de lignes de commandes à part) : *spark-shell --master {spark://*host:port *ou yarn} --name TP1-Spark*
Manipulation 6: Vérifier que *spark-shell* a bien démarré dans Yarn ou Spark Master.
Manipulation 7: Consulter les commandes disponibles. Utiliser *:help*.
Manipulation 8: Vérifier la bonne création de *SparkContext* : variable *sc* et son type avec *:type*
Manipulation 9: Consulter les packages automatiquement inclus dans la session Spark. Utiliser *:imports*.

2. Distribution de données

Manipulation 10: Créer la liste de nombres de 0 à 10. Utiliser *List* ou *Array* ou *(0 to 10).toList*.
Manipulation 11: Créer une RDD en distribuant cette liste sur les *workers* du cluster Spark. Utiliser la fonction Spark *parallelize*.
Manipulation 12: Afficher la RDD. Utiliser *foreach* et *println*.
Manipulation 13: Expliquer pourquoi rien ne s'affiche. Corriger à l'aide de la fonction *collect*.
Manipulation 14: Consulter le nombre de partitions de la RDD. Utiliser *getNumPartitions*.
Manipulation 15: Afficher les partitions du RDD. Utiliser *glom* et *collect*.
Manipulation 16: Consulter la (ou les) partition(s) attribuée(s) à chaque *worker*. Utiliser l'interface Web des *workers*.
Manipulation 17: Consulter l'algorithme/méthode de partitionnement du RDD. Utiliser l'attribut *partitioner*. Expliquer comment est partitionné le RDD.

3. Traitements simples de RDD numérique

Manipulation 18: Afficher la taille (nombre d'éléments) de la RDD. Utiliser *count*.
Manipulation 19: Afficher les 3 premiers éléments de la liste. Utiliser la fonction *take*. Quelle est le type (transformation/action) de la fonction *take* ?
Manipulation 20: Filtrer les éléments du RDD pour ne retenir que les nombres pairs. Utiliser la fonction *filter*. Afficher le RDD obtenu.
Manipulation 21: Calculer la somme de tous les éléments du RDD. La une fois avec *sum* et une fois avec *reduce*.
Manipulation 22: Calculer la somme des nombres pairs seulement. Utiliser *filter* et *sum*.
Manipulation 23: Calculer la somme des éléments par parité. Le résultat doit être un RDD contenant deux éléments : [("pairs", 20), ("impairs", 25)]. Utiliser *map* et *reduceByKey*.
Manipulation 24: Compter le nombre d'éléments par parité. Le résultat doit être un RDD contenant deux éléments : [("pairs", 4), ("impairs", 5)]. Utiliser *map* et *reduceByKey*.

4. Traitement de données structurées

Manipulation 25: Créer la liste de villes suivante dans une variable immuable ("Paris FR 5", "Stuttgart DE 0.9", "Lyon FR 2", "Londres UK 8", "Berlin DE 4", "Marseille FR 3", "Liverpool UK 1.5", "Munich DE 1"). Une ville est décrite par son nom, son pays et sa population exprimée en millions. Utiliser (la classe abstraite scala) *List* ou *Array*.
Manipulation 26: Afficher la RDD. Utiliser *foreach* et *println*.
Manipulation 27: Extraire le nom des villes seulement puis le pays seulement. Utiliser *map*.
Manipulation 28: Sélectionner et afficher les villes françaises seulement. Utiliser *filter*.
Manipulation 29: Refaire l'extraction du nom des villes (ou des pays) à l'aide de *map* qui prend cette fois-ci la fonction scala nommée : *extractVillePays(chaineVillePays :String, info :String) :String*. *Info* $\in$ {"ville", "pays", "population"}
Manipulation 30: Refaire la sélection de villes d'un pays à l'aide de *filter* qui prend cette fois-ci la fonction nommée à écrire : *isVilleDuPays (element :String, pays :String) :Boolean*.
Manipulation 31: Calculer la population totale de tous les pays. Utiliser *map* et *reduce/sum*.
Manipulation 32: Calculer la population totale par pays. Utiliser *map* et *reduceByKey*.

5. Traitement de fichiers non structurés (RDD de texte)

Manipulation 33: Charger le fichier texte .txt dans Spark sous forme de RDD. Comment le contenu du fichier a-t-il était chargé dans la RDD ?
Manipulation 34: Afficher le nombre de lignes de ce fichier.
Manipulation 35: Extraire seulement les lignes contenant un mot donné.
Manipulation 36: Calculer le nombre de mots total dans le fichier.
Manipulation 37: Calculer le nombre de caractères total dans le fichier.
Manipulation 38: Extraire seulement les lignes ayant une taille (en nombre de mots ou de caractères) minimum/maximum donnée. Utiliser la fonction à écrire : *int tailleElement (String element, String unité)* où *unité* $\in$ *{"mot", "caractère"}* permettant de calculer la taille d'une ligne.
Manipulation 39: Calculer le nombre de mots par ligne.
Manipulation 40: Calculer le nombre d'occurrence de chaque mot dans tout le fichier.
Manipulation 41: Enregistrer le résultat dans le HDFS. Utiliser *saveAsTextFile*

6. Programmation de RDD d'objets

Manipulation 42: Coder la classe : *Personne (numero : Int, nom : String, prenom : String, dateNaiss : LocalDate, ville : String)*. Utiliser *case class*.
Manipulation 43: Charger le fichier *Personnels.csv* dans Spark. Utiliser *textFile*.
Manipulation 44: Créer une RDD de personnes (collection distribuée de type RDD [Personne]) à partir de la RDD précédente. Utiliser *map* et la fonction scala à coder : *textToObjetPersonne (lignePersonne :String):Personne* permettant de créer un objet *Personne* à partir d'une ligne du fichier *csv*. Utiliser la méthode *parse()* de *LocalDate* et *DateTimeFormatter* pour convertir le texte en *LocalDate*.
Manipulation 45: Afficher seulement le nom et la date de naissance des personnes.
Manipulation 46: Sélectionner les personnes nées avant 2000 et habitant une ville donnée.
Manipulation 47: Calculer le nombre de personnes par ville. Utiliser *reduceByKey*.
Manipulation 48: Calculer le nombre de personnes par décennie (années 80, 90, 2000, etc.) de naissance.
Manipulation 49: Afficher le nom et l'âge des personnes. Utiliser *map* et écrire une fonction scala pour le calcul d'âge.
Manipulation 50: Sélectionner les personnes ayant un certain âge minimum/maximum.
Manipulation 51: Calculer l'âge moyen par ville.

7. Jointure de RDD

Manipulation 52: Charger le fichier *Employeurs.csv* dans un RDD d'objets *Employeur* à coder.
Manipulation 53: Calculer la masse salariale des employés.
Manipulation 54: Faire la jointure des fichiers *Personnels.csv* et *Employeurs.csv*. Utiliser *join* (après avoir constitué les paires *(clé, valeur)* avec *map*).
Manipulation 55: Afficher le résultat puis le stocker dans un fichier local/HDFS.
Manipulation 56: Calculer le salaire maximum et minimum par ville.
Manipulation 57: Calculer le salaire moyen par ville.
Manipulation 58: Calculer le salaire minimum, maximum et moyen par décennie (années 80, 90, 2000, etc.) de naissance.

Quelques éléments de sytaxe du langage Scala

Déclarer une variable :
var <nom-de-variable-*mutable*> : <type-optionnel> = <valeur-initiale> ;
val <nom-de-variable-*constante*> : <type-optionnel> = <valeur-initiale> ;
Déclarer une fonction :
def <nom-de-fonction> (<nom-argument> : <type-argument>) : <type-de-retour-optionnel> =
<bloc-d'instructions> ;
*Déclarer une classe **case** :*
case class <nom-de-la-classe> (<nom-attribut>:<type-attribut>, …) // attributs de type **val**
val o1 = <nom-de-la-classe> (<valeur-initiale-attribut1>, <valeur-initiale-attribut2>, …)
Déclarer une classe :
class <nom-de-la-classe> (**var/val** <nom-attribut-constructeur>:<type-attribut>, …) **{**
var/val <nom-attribut-supplémentaire> = <valeur-initiale>
def <nom-de-méthode> (<nom-argument>:<type-argument>) : <type-de-retour> =

```
        <bloc-d'instructions> ;
  }
  val o2 = new <nom-de-la-classe> (<valeur-initiale-attribut1>, <valeur-initiale-attribut2>, …)
```

charger un fichier de code Scala dans l'interpréteur de commandes :

```
  :load <chemin-vers-le-fichier>/<nom-fichier-scala>
```