Big Data : Informatique pour les données et calculs massifs

2A – campus de Metz – 2018-2019

TD2 : algorithmique *Map-Reduce*

Exercice 1 : Analyse statistique de la longueur des mots d'un texte en *Map-Reduce* (fin du TD1)

On souhaite établir des statistiques sur la longueur des mots dans un document, ou un ensemble de documents, à partir d'une application *Map-Reduce*. Proposez un algorithme dans le paradigme *Map-Reduce* pour les 3 analyses suivantes :

Question 1 : Compter le nombre de mots de chaque longueur présente dans le texte (en vue d'établir un histogramme des longueurs de mots).

Question 2 : Compter le nombre de mots de 1 à 5 caractères (inclus), de 6 à 10 caractères (inclus), de 11 à 15 caractères (inclus) et de plus de 15 caractères présents dans le texte.

Rmq : si on génère plusieurs fichiers de sorties, il est nécessaire que l'ordre des noms de fichiers soit celui des sous-ensembles de longueurs de mots.

Question 3 : Obtenir les listes de mots de 1 à 5 caractères (inclus), de 6 à 10 caractères (inclus), de 11 à 15 caractères (inclus) et de plus de 15 caractères présents dans le texte. Il n'est pas demandé de trier les mots à l'intérieur d'une liste, ni d'éliminer les doublons.

Dans tous les cas on décrira les paires clé-valeur et le pseudo-code des traitements utilisés à chaque étape de la solution *Map-Reduce*, et on essaiera d'optimiser ces solutions.

Exercice 2 : Calcul de valeur médiane et d'écart type en Map-Reduce

Un ensemble de documents contiennent les données des transactions commerciales d'une entreprise. Chaque donnée est un *Record* qui contient notamment l'identificateur du client identifié par la clé (ou la balise) « *ClientID* », et le montant de la transaction identifié par la clé « *Price* » (ex : {Price : 1000}), qui peuvent ainsi être facilement extraits de la donnée. Une fois ces données extraites, on cherche à calculer la valeur médiane et l'écart type des montants des transactions avec chaque client.

On utilisera un paradigme *Map-Reduce*, ou les paires clé-valeur d'entrée des *Mappers* seront un offset dans un fichier (clé), et un texte décrivant toute une transaction (valeur). L'extraction du *ClientID* et du *Price* sera supposé ne pas poser de difficulté (Java propose de nombreuses classes permettant d'analyser des documents XML ou JSON…).

Question 1 : Proposer une solution dans le paradigme *Map-Reduce*.

Question 2 : Quelles sont les faiblesses de la solution ?

Question 3 : Si on considère que le nombre de prix possibles est limité (par exemple : 10, 50, 99, 499, 999, 9999,…), et que chaque client a réalisé de nombreuses transactions, alors des optimisations deviennent-elles possibles ?

Question 4 : Quel est l'impact d'une valeur aberrante parmi les données (mauvaise préparation des données) ?

Exercice 3 : Nouveau calcul d'écart-type en Map-Reduce

On considère un ensemble de flux de données, qui sont stockées dans un fichier de grande taille sous la forme d'une suite d'enregistrements structurés (*Record*) :

```
{ fluxId   : LaRefDuFluxDeDonnées,
  mesure   : LaValeurDeLaMesure,
  date     : LaDateDeLaMesure,
  priorite : LaPrioriteDuFLux }
```

On suppose que la classe définissant la lecture des fichiers d'entrée permet de retourner des paires clé-valeurs classiques de type < *offset* , *Record* >, dont on peut facilement extraire la valeur de chaque champs du *record*.

On peut exprimer l'écart type de N valeurs x_i (avec : $0 \leq i < N$) selon deux définitions équivalentes :

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

Question 1 : Concevez et implantez en pseudo-code un calcul de l'écart type de chaque flux, selon la définition 2 de l'écart type.

- Décrivez rapidement une stratégie qui vous semblera optimale, et tirez profit de la définition 2 pour éviter les problèmes que rencontrait la définition 1.
- Précisez les différentes paires clé-valeur et le pseudo-code d'une solution Map-Reduce, en Hadoop, qui vous semblera optimale.
- Prévoyez-vous de créer 1, 2,3…, ou « plusieurs » *Reducers* ?

Question 2 : Avec votre implantation de la définition 2, est-il possible de traiter des flux extrêmement longs ? Pourquoi ?

Exercice 4 : Calculs de moyenne pondérée en Map-Reduce

On considère des rafales de mesures regroupés dans des *Records*, stockés dans un fichier Hadoop (dans le système de fichiers HDFS), et représentant les sorties de différents capteurs :

```
{ moteurID : RefDuMoteur,
  capteurID : RefDuCapteur,
```

```
rafale : [v0,…, v9],
unité : UnitéDeLaMesure,
fiabilitéCapteur : PoidsDeLaMesure }
```

On peut ainsi analyser de multiples mesures concernant l'ensemble des moteurs. Afin d'obtenir un système plus fiable, plusieurs capteurs sont installés sur chaque moteur, qui mesurent la même grandeur, mais qui n'ont pas tous la même fiabilité.

Question 1 : Calcul de la valeur moyenne d'une série de rafales

- Concevez et écrivez en pseudo-code un algorithme *Map-Reduce* (en Hadoop) permettant de calculer la valeur moyenne de toute une série de rafales concernant un même moteur. Chaque mesure participera à la moyenne en étant pondérée par le coefficient de fiabilité de son capteur.

  Concevez une solution simple, constituée seulement de *Mappers* et de *Reducers*.

- Faut-il créer un ou plusieurs *Reducers* ?

Question 2 : Solution optimisée en Hadoop

Utilisez toute la puissance d'expression d'Hadoop pour implanter une solution optimisée, qui notamment ne fasse pas tous les calculs dans les *Reducers*, et surtout qui réduise le trafic dans le *Shuffle & Sort*.

- Définissez les paires clé-valeurs de votre solution (si besoin elles peuvent être différentes de celles de la solution précédente)
- Indiquez le pseudo-code de vos différentes tâches

Question 3 : Elimination de mesures aberrantes et détection des capteurs suspects

En fait, toutes les mesures ne sont pas fiables. Il arrive qu'un capteur soit perturbé et produise une valeur aberrante. Nous considérerons qu'une valeur de mesure inférieure à -100 ou supérieure à +100 est une valeur aberrante.

- Concevez une solution Map-Reduce Hadoop qui, pour chaque moteur, élimine les valeurs aberrantes du calcul de la moyenne ET qui construise la liste des capteurs suspects ayant généré au moins une valeur aberrante.

  Au final, en sortie des *Reducers* on générera des paires similaires à :

  < clé, { moyenne : *ValeurMoyenne*, CaptSuspects : [*capt1*, *capt2*, … *captn*] } >

- Précisez bien les formats de paires clé-valeurs de votre solution en entrée et sortie des *Mappers* et *Reducers*.

Rmq : On supposera qu'on peut faire appel à une bibliothèque de manipulation de listes, dans toutes les portions de codes de la solution :

- LL = Append(e,L) : ajoute l'élément e à la fin de la liste L, et retourne la liste modifiée
  Ex : append(5,[1, 2, 3, 4]) → [1, 2, 3, 4, 5]

- LL = Aplatir(L) : génère une liste d'éléments (LL) à partir d'une liste contenant des éléments ou des sous-listes d'éléments.

  Ex : aplatir([1, [2, 3], 4]) → [1,2,3,4]

- LL = simplifie(L) : élimine les doublons dans une liste d'éléments :

  Ex : simplifie([1, 2, 3, 2, 3, 1, 4]) → [1, 2, 3, 4]