RAPPORT DE PROJET

Big Data, Cloud Computing et Services Web

Axel BRUNEL
Maxime EMONNOT
Manon LACOMBE

I – Introduction

Au cours de ce projet, nous avons traité le contexte d'une base de données du département d'informatique de l'Université de Blois. Cette base de données répertorie l'ensemble des étudiants inscrits aux différentes promotions (L1, L2, L3, M1, M2), les différents cours auxquels ils peuvent assister ainsi que les résultats de chaque étudiant.

L'objectif principal est donc d'apporter une solution permettant de traiter convenablement un problème de cette envergure, étant donné certaines contraintes de volume, de matériel, et de demandes spécifiques. Ces contraintes constituent une certaine problématique, et il est donc nécessaire d'apporter la solution la plus adéquate pour la résoudre.

Pour ce faire, nous choisirons donc dans un premier temps la technologie de stockage que nous utiliserons parmi deux possibilités (HDFS et HBase). À la suite de ce choix, nous structurerons l'organisation des données non modifiables de notre base, que ce soit le nombre de fichiers, la structure des lignes ou colonnes. Enfin, nous réaliserons différentes analyses et traitements sur trois requêtes sélectionnées, correspondant à des cas concrets de requêtes de traitement.

II – Choix de la technologie de stockage

Afin d'apporter une solution à notre problème, vient le choix entre deux systèmes : HDFS et HBase.

Dans le cadre de notre projet, nous avons choisi d'utiliser **HDFS**, car il permettra de traiter simplement des fichiers textes dont la structure sera préalablement définie. De plus, l'utilisation de ces fichiers est assez aisée, car chacun peut rajouter simplement des lignes dans le fichier, à condition de respecter les règles d'écriture dudit fichier.

Il s'agit du principal intérêt de notre solution : permettre la lecture d'un grand nombre de données, aisément extensibles.

III – Organisation des données non modifiables

Après avoir sélectionné notre technologie de stockage, il est important de définir certaines règles concernant l'organisation de nos données. Pour rappel, nous devons traiter différentes informations au sujet des étudiants, les cours auxquels ils peuvent assister, et leur réussite à ces cours.

Il est également important de prendre en compte certains cas, notamment la possibilité pour un cours de changer de nom d'une année à l'autre, la possibilité qu'un étudiant redouble une année, ou bien qu'un cours soit enseigné à un semestre différent des années précédentes en raison d'un changement de maquette.

Compte tenu de ces contraintes et possibilités, nous avons choisi de représenter les données ligne par ligne, à savoir :

- Certaines lignes traiteront uniquement des informations concernant un étudiant. Elles auront la structure suivante:

  E;Num étudiant;Année;Promotion;Nom;Prénom;Date de naissance;Email;Numéro de téléphone;Adresse

  Cette structure permet notamment de traiter des cas où un étudiant est dans la même promotion d'une année à l'autre (redoublement), ou bien déménage d'une année à l'autre, ou encore change de numéro de téléphone. La mention de "E" permettra par ailleurs d'identifier le type de ligne en tant que "Ligne étudiant".

- Certaines lignes traiteront uniquement des informations concernant un cours. Elles auront donc la structure suivante:

  U;Code UE;Année;Nom UE;Enseignant1,Enseignant2,Enseignant3,…

  Cette structure permet ainsi de traiter des cas où un cours change de nom d'une année à l'autre, sachant que le code de l'UE ne changera pas. La mention de "U" permettra d'identifier le type de ligne en tant que "Ligne UE".

- Certaines lignes traiteront des informations concernant les notes d'un étudiant pour une UE. Elles auront la structure suivante:

  N;Code UE;Année;Num étudiant;Note

  Cette structure permet donc de traiter des cas où un étudiant redouble et a donc des notes différentes d'une année à l'autre pour une même UE, en plus des cas classiques. La mention de "N" permettra d'identifier le type de ligne en tant que "Ligne Note".

IV – Analyse et réalisation des tâches

Compte tenu du contexte du projet, il nous a été proposé de choisir 3 requêtes parmi un ensemble de 5 requêtes réalisables à l'aide de nos données :

1. Sortir le relevé de notes selon un **numéro d'étudiant** et une **année scolaire**
2. Sortir le taux de réussite de notes d'un **semestre** selon l'**année scolaire**
3. Sortir le taux de réussite d'une **UE** depuis sa création par rapport à ses différents noms
4. Sortir les taux de réussite de toutes les UE assurées par un **intervenant**
5. Sortir le classement d'étudiants par rapport à leurs notes moyennes selon la **promotion** et l'**année scolaire**

Nous avons donc choisi de traiter les requêtes 2, 3 et 4.

Le code correspondant aux morceaux de pseudo-code ci-dessous peut être retrouvé sur le lien GitHub suivant: https://github.com/MaximeEmonnot/Projet-CloudComputing-Hadoop

A – Requête 2 : Sortir le taux de réussite de notes d'un semestre selon l'année scolaire

Au vu de la structure de nos données, il est important de réaliser en amont un pré-traitement afin de générer des fichiers temporaires, ce qui nous permettra de simplifier les calculs lors de la génération de la réponse finale.

Nous réaliserons donc un fichier temporaire dont les lignes seront de la forme suivante :

```
Année;Semestre;NumEtudiant;Moyenne
```

Nous pouvons ainsi créer ce fichier à l'aide d'un job Map-Reduce, que nous pouvons représenter à l'aide du pseudo-code suivant :

```
map(k, v) :
     if v[0] == "N" :
        (N, codeUE, annee, numEtudiant, note) = split(v, ";")  // On ne traite
que les lignes commençant par "N;", correspondant aux notes
        newK = annee + ";" + substr(codeUE, 0, 3) + ";" + numEtudiant
        emit(newK, note)

reduce(k, [v]) :
     sum = 0
     num = 0
     for note in [v]:
        num++
        sum+=note
     moyenne = sum/num
     emit(k + “;” + moyenne, "")  // annee;semestre;numEtudiant;moyenne
```

Une fois notre fichier temporaire créé, nous pourrons donc traiter la requête, à l'aide d'un nouveau job Map-Reduce, l'objectif étant d'obtenir une sortie de la forme suivante :

```
2012/S7 - 0,88
```

Il sera important de noter la nécessité de définir au préalable nos paramètres, définition que nous réaliserons et représenterons à l'aide de la fonction setup.

```
setup :
     sem="S7"
     anneeScolaire="2012"

map(k, v) :
     (annee, semestre, numEtu, moyenne) = split(v, ";")
     if annee == anneeScolaire && semestre == sem:
        newK = annee + "/" + semestre
        emit(newK, moyenne)

reduce(k, [v]) :
     totalNotes = 0
     notesValides = 0
     for note in [v] :
        totalNotes++
        if note >= 10:
           noteValides++
     taux = (notesValides/totalNotes)
     emit(k + " - " + taux, "")
```

B – Requête 3 : Sortir le taux de réussite d'une UE depuis sa création par rapport à ses différents noms

De la même manière que pour la requête précédente, il sera important de réaliser un pré-traitement permettant la génération de fichiers temporaires, qui nous simplifieront les calculs lors de la génération de réponse finale.

Nous générerons donc un fichier temporaire de la forme suivante :

```
CodeUE;NomUE;TauxRéussite
```

Ce fichier temporaire peut être créé à l'aide du job Map-Reduce suivant :

```
map(k, v) :
     if v[0] == "U" :
         (U, codeUE, annee, nomUE, listeEnseignants) = split(v, ";")
         newK = codeUE + "/" + annee
         emit(newK, nomUE)
     else if v[0] == "N"
         (N, codeUE, annee, numEtudiant, note) = split(v, ";")
         newK = codeUE + "/" + annee
         emit(newK, (note > 10 ? 1 : 0).toString())

reduce(k, [v]) :
     count = 0
     sum = 0
     nomUE = ""
     codeUE = split(k, "/")[0] // k est de la forme CodeUE/Annee
     for v in [v]:
        if v.size() == 1: // v vaut soit 0, soit 1, soit le nom de l'UE. On ne
traite que 0 et 1 dans cette condition.
           count++
           sum += v.toInt()
        else:
           codeUE = v
     tauxReussite = sum / count
     emit(codeUE + “;” + nomUE + ";" + tauxReussite , "")
```

Une fois notre fichier temporaire créé, nous pouvons traiter la requête à l'aide d'un job Map-Reduce, sachant que nous devons obtenir une réponse de la forme suivante :

```
HPC - 0.183
Big Data - 0.812
```

Comme pour la tâche précédente, il est important de définir au préalable nos paramètres :

```
setup :
     code="S07A001"

map(k, v) :
     (codeUE, nomUE, tauxReussite) = split(v, ";")
     if codeUE == code:
        emit(nomUE, tauxReussite)

reduce(k, [v]) :
     count = 0
     sum = 0
     for taux in [v] :
        count++
        sum += taux
     moyenne = (sum/count)
     emit(k + " - " + moyenne, "") // Nom UE - Moyenne
```

C – Requête 4 : Sortir les taux de réussite de toutes les UE assurées par un intervenant

De la même manière et pour les mêmes raisons que les deux précédentes requêtes, nous réaliserons un pré-traitement pour obtenir un fichier temporaire de la forme suivante :

```
NomIntervenant;CodeUE;Annee;NomUE;Note1,Note2,Note3,…
```

Ce fichier temporaire peut être créé à l'aide du job Map-Reduce suivant :

```
map(k, v) :
     if v[0] == "U" :
         (U, codeUE, annee, nomUE, listeEnseignants) = split(v, ";")
         for intervenant in split(listeEnseignants, ",") :
            newK = codeUE + "/" + annee
            value = nomUE + ";" + intervenant
            emit(newK, value)
     else if v[0] == "N"
         (N, codeUE, annee, numEtudiant, note) = split(v, ";")
         newK = codeUE + "/" + annee
         emit(newK, note)

reduce(k, [v]) :
     notes = ""
     [intervenants] = {}
     nomUE = ""
     (codeUE, annee) = split(k, "/)
     for v in [v]:
        if v.contains(";") : // v est de la forme NomUE;NomIntervenant
           (ue, intervenant) = split(v, ";")
           intervenants.push_back(intervenant)
           nomUE = ue
        else : // v est une note
           notes += (notes.empty() ? "" : ",") + v
     for intervenant in [intervenants] :
        emit(intervenant + ";" + codeUE + ";" + annee
                         + ";" + nomUE  + ";" + notes)
```

Une fois notre fichier temporaire créé, nous pouvons traiter la requête à l'aide d'un nouveau job Map-Reduce, en gardant en mémoire que nous devons obtenir un résultat de la forme suivante :

```
Prof Diff
----
S08A001/2017 - Cours Très Difficile - 0,12
S08A001/2018 - Cours Hyper Difficile - 0,08
S09A003/2016 - Cours Moins Difficile - 0,23
S09A003/2017 - Cours Moins Difficile - 0,57
```

Enfin, comme pour les tâches précédentes, il est important de définir préalablement nos paramètres :

```
setup :
     intervenant="Prof Diff"

map(k, v) :
     (nomIntervenant, codeUE, annee, nomUE, notes) = split(v, ";")
     if nomIntervenant == intervenant:
        for note in split(notes, ","): // notes est une liste de notes séparées
par une virgule
           emit(nomIntervenant + ";" + codeUE + ";" + annee + ";" + nomUE, note)

reduce(k, [v]) :
     (intervenant, codeUE, annee, nomUE) = split(k, ";")
     notes = 0
     notesValides = 0
     for v in [v] :
        notes++
        notesValides += (v ≥ 10) ? 1 : 0
     tauxReussite = notesValides / notes
     emit(codeUE + "/" + annee + " - " + nomUE + " - " + tauxReussite, "")
```