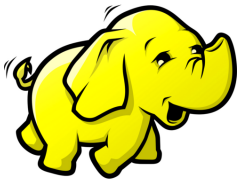
Travaux pratiques synthèse

Jonathan Lejeune




Objectifs

Ce sujet de travaux pratiques met en œuvres les différents systèmes abordés précédemment :

- Hadoop : pour le stockage physique des données (HDFS) et l'attribution des ressources de calcul (YARN)
- Spark : pour le traitement parallèle de données
- Hbase et Cassandra : pour un accès direct et efficace aux données

Pré-requis

Vous devez avoir :

- installé et démarré les serveurs pour HDFS, HBase et Cassandra. Spark sera utilisé en mode local, il est donc inutile de le démarrer en pseudo-distribué, idem pour Yarn.
- installé le driver Spark-Cassandra
- préparé un projet scala eclipse qui inclut dans le build path les différentes librairies définies dans les TP précédents. Veillez à ce que la librairie Spark soit positionnée avant la librairie de Hbase (les deux librairies inclut une même dépendance mais la version de Spark est plus récente ce qui explique sa priorité sur Hbase). Ceci est configurable dans l'onglet *Order and Export* dans la fenêtre de configuration du build path.
- inclure la classe HbaseClient que vous avez codé dans le TP Hbase et qui doit être fonctionnelle.

Exercice 1 – Prise en main de l'interface Spark-Hbase

Le fichier `SparkHbaseConnector.scala` fourni dans les ressources du TP (package *datacloud.synthese*) vous permet de :

- créer un RDD Spark à partir d'une table Hbase (méthode `hbaseTableRDD` de la classe `MySparkContext`, appelable directement depuis un `SparkContext` grâce à une conversion implicite)
- d'écrire un RDD dans une table Hbase existante (méthode `saveAsHbaseTable` de la classe `RDDHbase`, appelable directement depuis un `RDD[Mutation]` grâce à une conversion implicite)

Question 1

Analysez le code de la méthode `hbaseTableRDD` afin de répondre aux questions suivantes :

- Que représente un élément du RDD produit ?
- Comment sont partitionnées les données du RDD produit ?

- Pourquoi utilisons nous la méthode `newAPIHadoopRDD` ?

Question 2

Analysez le code de la méthode `saveAsHbaseTable` afin de répondre aux questions suivantes :

- Comment le RDD est traduit en table Hbase ?
- Pourquoi avons-nous utiliser l'action `foreachPartition` au lieu de l'action `foreach` ?
- Pourquoi les mutations de type `Put` et `Delete` sont-elles ajoutées dans des listes avant d'être envoyées ?
- Que se passerait-il si on avait déclaré `class RDDHbase(rdd:RDD[Mutation])` au lieu de `class RDDHbase[M<:Mutation](rdd:RDD[M])` ?

Question 3

Dans le package `datacloud.synthese`, écrire un `object` scala `LastFmUtil` qui offre une méthode `fillfromFile`. Cette méthode prend en paramètre un fichier texte d'entrée au format de log de LastFM (`File`), un nom de table Hbase de destination (`TableName`) supposée déjà créée, le nom de la column family (`String`) et un sparkcontext. Le but de cette fonction est de remplir avec Spark la table destination à partir des données textuelles du fichier.

Question 4

Tester avec la classe Junit suivante fournie dans les ressources de TP :

`datacloud.synthese.LastfmSparkHbaseTest`

Question 5

Dans le package `datacloud.synthese`, écrire un `object` scala `HbaseSparkUtil` qui offre une méthode `copyTable`. Cette méthode prend en paramètre un nom de table Hbase source (`TableName`) supposée déjà créée et remplie, un nom de table Hbase destination (`TableName`) supposée inexistante, un spark context et une connection Hbase (`Connection`). Le but de cette fonction est de copier à l'identique le contenu de la table source vers la table destination.

Question 6

Tester avec la classe Junit suivante fournie dans les ressources de TP :

`datacloud.synthese.CopyTableTest`

Exercice 2 – Immatriculations de voitures

Le ministère de l'intérieur maintient un fichier de l'ensemble des voitures immatriculées dans le pays. Le service informatique héberge ces données sur le SGBD Cassandra.

Question 1

Exécuter le programme scala `CassandraSchemaBuilder` qui permet de créer et de remplir les tables qui nous allons interroger. Ce programme créé trois tables :

- `Proprietaire`
- `ModeleVehicule`
- `Immatriculation`

Une immatriculation est un numéro d'identification qui relie un modèle de véhicule et l'ensemble des propriétaires successifs au cours de la vie de ce véhicule (le propriétaire actuel étant celui associé à la date la plus récente). L'ensemble des propriétaires est modélisé par une map qui associe une date d'acquisition du véhicule à son propriétaire. On maintient également la date de la première mise en circulation et un statut qui indique si le véhicule associé est "en circulation", "volé" ou "détruit".

Question 2

Vérifier à travers un shell cql que les tables ont bien été créées et bien remplies.

Question 3

Dans le package `datacloud.synthese`, écrire un `object` scala `ImmatUtil` qui offre une méthode `changeAdresse` qui permet de changer l'adresse d'un propriétaire. Cette méthode prend trois paramètres : l'identifiant du propriétaire, la nouvelle adresse et un SparkContext correctement instancié et initialisé.

Question 4

Tester avec la classe Junit suivante fournie dans les ressources de TP :

`datacloud.synthese.ChangeAdresseTest`

Question 5

Dans ce même `object` `ImmatUtil`, écrire une méthode `topThree` qui prend en paramètre une année (`Int`) et un sparkcontext (correctement instancié et initialisé) et qui renvoie un `Iterable` des 3 marques de voiture les plus souvent mis en circulation pour cette année.

Question 6

Tester avec la classe Junit suivante fournie dans les ressources de TP :

`datacloud.synthese.Top3Test`

Le service de vidéosurveillance des axes routiers produit un flux temps réel de logs après un traitement d'analyse d'imagerie. Le service informatique de la police nationale a décidé de déployer un programme permettant de traquer les voitures déclarées volées en croisant en temps réel les données acquises par le service de vidéo-surveillance des axes routiers et la base de données des immatriculations. **L'objectif ici est de mettre à jour en temps réel une table Hbase, qui maintient la liste chronologique des endroits auquel tout véhicule déclaré volé a été repéré.**

Question 7

Dans le même `object` `ImmatUtil`, écrire une méthode `fillTracingTable` qui prend en paramètre :

- un `DStream` de String pour les logs des caméras. Toute voiture circulant devant la caméra engendre un log qui contient 4 mots séparé par un espace : la date du cliché (timestamp), l'immatriculation de la voiture et les coordonnées GPS de la caméra.
- un `TableName` qui indique le nom de la table Hbase où écrire les données
- une `Connection` Hbase
- un `sparkContext`

Elle permet de remplir la table Hbase en temps réel permettant de maintenir pour chaque immatriculation volée la liste chronologique de ses points de passage.

Question 8

Ajouter dans `ImmatUtil` une méthode `getJourneyOf` qui prend trois paramètres : une immatriculation (`String`), un `TableName` (le nom de la table remplie dans la question précédente) et une `Connection` Hbase. Elle renvoie un `Seq[(Long,Long)]` qui est la liste des coordonnées des clichés de l'immatriculation donnée. Cette liste est triée par ordre chronologique.

Question 9

Tester avec la classe Junit suivante fournie dans les ressources de TP :

`datacloud.synthese.TracingTest`