HMIN313 : Introduction à HBase - Exercices complets

1. Enoncé

Nous abordons dans ce TP, quelques principes de base de mise en œuvre au sein du système HBase. HBase admet trois modes de déploiement : un mode autonome (ou local), un mode pseudo-distribué et enfin un mode pleinement distribué. Nous travaillerons sur une instance HBase locale. Dans ce mode autonome, HBase exploite un système de fichier local et non pas HDFS. Les processus d'arrière plan (RegionServer, HMaster et ZooKeeper) sont regroupés et exécutés dans la même machine virtuelle Java JVM.

1. client JRuby : construction de tables et de familles de colonnes, et opérations insertion et suppression de données ("column qualifier", valeur, estampille)
2. client Java : construction de tables et de familles de colonnes, et opérations insertion et suppression de données ("column qualifier", valeur, estampille)

Vous exploiterez dans le cadre de ce td, à la fois le client JRuby et l'API Java. Pour ce faire, vous exploiterez une version un peu ancienne de HBase disponible sur Moodle (pour des versions plus récentes, voir https ://www.apache.org/dist/hbase/stable/). Pour plus de documentation, allez voir la page http : //hbase.apache.org/book/quickstart.html.

```
tar -zxvf hbase-0.94.2-security.tar.gz
```

Listing 1 – Archive cible

Vous démarrerez votre serveur HBase (exploitation en mode local).

```
%Install java, edit conf/hbase-env.sh, uncommenting the JAVA_HOME
./bin/start-hbase.sh
 puis ./bin/hbase shell
```

Listing 2 – Avant de commencer

2. Appropriation

Vous ferez en sorte de construire une table **Commune** caractérisée par plusieurs familles de colonnes, en exploitant le client JRuby. Vous alimenterez cette table à partir de quelques tuples, en prenant la ville de Montpellier en exemple. Un exemple de table relationnelle Commune en extension vous est donnée en annexe. Vous traduirez cette table en table HBase. Vous pouvez également vous aider des fichiers au format json fournis lors du TP CouchDB.

2.1 Question 1

— construire et alimentez la table **Commune** à partir du client JRuby et explorer les commandes de définition et de manipulation des données (create, put, delete, list, scan, ...). L'aide en ligne est consultable avec les commandes `help 'ddl'` et `help 'dml'`, respectivement pour la création et la manipulation de données. Des exemples de manipulation d'une table 'personne' vous sont donnés en annexe. Vous reprendrez ces exemples dans le cadre de la définition de la table Commune (insérer deux à trois tuples de communes).

— opérer le même travail à partir d'un projet Java sous Eclipse et des API dédiées. A cet effet, vous définirez une classe applicative dotée d'une méthode permettant d'afficher (scan) le contenu d'une colonne (dernière valeur) pour tous les tuples de la table Commune.

— consultez l'information concernant le serveur master de la base Hbase à l'adresse `http://localhost:60010`. En particulier, combien de "Region" et de "Store" ont été définis pour organiser au mieux l'information contenue dans **Commune** ?

2.2 Question 2

1. Manipulez l'exemple de filtre fourni, et proposez un exemple de consultation qui pourrait avoir un sens dans le contexte de l'information sur les communes, par exemple retourner les communes qui ont plus de 150000 habitants en 2010 ou encore renvoyer le nombre de communes par numéro de département.
2. Exploiter l'exemple `ParcoursScanPersonne.java` pour compter le nombre de tuples de Commune.
3. Enrichir la classe `ParcoursScanPersonne.java` et la méthode `dumpResult` pour lister tout le contenu d'une table (Commune), l'ensemble des valeurs d'une colonne comprises.

2.3 Question 3

Une fois, les modèles de données établis, élaborez une approche pour automatiser l'alimentation en données de votre système HBase à partir de fichiers tabulés.

2.4 Question 4

Une fois les principes de construction et de consultation des tables acquis, réfléchissez à quelle conduite à tenir pour gérer des données enrichies par des descriptions provenant par exemple de l'Insee, du site des impôts ou bien de l'IGN. Vous pouvez également penser à l'ajout de familles de colonnes dans **Commune**

— proposer à ce titre la construction de nouvelles tables à partir du client JRuby ou Java

2.5 Question 5 : coprocesseurs

Vous reprendrez l'exemple de la classe `ContactsAgregation.java` qui s'appuie sur un Coprocessor prédéfini de type endpoint. Construisez une classe qui applique un calcul sur la base d'une agrégation. Un exemple de coprocesseur Observer vous est également donné. Vous le testerez.

2.6 Question 6 : map-reduce

A partir de l'exemple MapReduce qui vous est fourni :

— vous dériverez un exemple approchant pour Commune

— vous pouvez vous inspirer des exemples trouvés sur `https://github.com/apache/hbase/blob/master/src/main/asciidoc/_chapters/mapreduce.adoc#mapreduce.example.read`

3. Annexe

3.1 Extension Table Commune

```
CODE_I NOM_COM                                    POP_1975 POP_2010
------ ---------------------------------------- ---------- ----------
34248 SAINTE-CROIX-DE-QUINTILLARGUES    127.033553 561.720409
34249 SAINT-DREZERY                      587.63955 2207.40611
34025 BASSAN                            795.304324 1599.4016
34250 SAINT-ETIENNE-D'ALBAGNAN           253.31857 287.52352
34251 SAINT-ETIENNE-DE-GOURGAS          216.105575 489.314637
34252 SAINT-ETIENNE-ESTRECHOUX          406.570257 245.980104
34253 SAINT-FELIX-DE-L'HERAS             25.248617 36.8979349
34254 SAINT-FELIX-DE-LODEZ              519.740374 1209.95666
34255 SAINT-GELY-DU-FESC                2010.13406 8528.24947
34256 SAINT-GENIES-DES-MOURGUES         825.375932 1623.96003
34257 SAINT-GENIES-DE-VARENSAL          218.473421 209.311992
34167 MONTELS                           127.655942 246.150356
34168 MONTESQUIEU                       34.3551636 63.8836005
34169 MONTFERRIER-SUR-LEZ               1682.12888 3320.80271
34017 AUMES                             307.185836 455.743575
34170 MONTOULIERS                       209.369513 237.112177
34171 MONTOULIEU                        38.7548087 161.147619
34172 MONTPELLIER                       191767.053 260959.188
34173 MONTPEYROUX                       774.815254 1206.2612
34174 MOULES-ET-BAUCELS                 131.447644 845.71166
34175 MOUREZE                           78.5494361 176.923015
34176 MUDAISON                          735.409812 2506.47942
```

Listing 3 – Tuples exemples

3.2 Exemples d'interaction avec l'interpréteur JRuby

```
Créer une table 'personne' avec une famille de colonnes 'contact'.
Vérifier que la table a bien été créée puis afficher la description de la table
    'personne'
Modifier la famille de colonnes 'contact' pour qu'elle affiche jusqu'à 4 versions des
    valeurs de ses colonnes.
Vérifier la modification en affichant à nouveau la description de la table.
Ajouter dans la table 'personne', le tuple '176073424356' avec la valeur 'Martin'
    pour la colonne 'contact:nom'
Lister le contenu de la table 'personne'
compter le nombre de personnes
Ajouter pour le tuple '176073424356', la valeur '39' pour la colonne 'contact:age'
Ajouter dans la table 'personne', le tuple '2860738256345' avec la valeur 'Dupond'
    pour la colonne 'contact:nom'
Ajouter une valeur quelconque pour la colonne 'contact:prenom' du tuple
    '2860738256345', puis modifier la valeur de cette colonne 4 fois de suite.
Afficher le tuple '2860738256345'.
```

```
  Afficher la valeur courante de la colonne '2860738256345'-'contact:prenom'.
  Afficher la valeur précédente de la colonne '2860738256345'-'contact:prenom'.
  Afficher toutes les valeurs précédentes de la colonne
      '2860738256345'-'contact:prenom'.
  Lister les lignes dont la clef commence par '286'.
  Supprimer la cellule '176073424356'-'contact:nom' et lister le contenu de la table.
  Supprimer la famille de colonne 'contact' de la table (à l'aide de la commande alter).
  Supprimer la table 'personne'

 --- réponses
  create 'personne', 'contact'
  list
  disable 'personne'
  alter 'personne', NAME => 'contact', VERSIONS => 4
  enable 'personne'
  describe 'personne'
  put 'personne', '176073424356', 'contact:nom', 'Martin'
  scan 'personne'
  count 'personne'
  put 'personne', '2860738256345', 'contact:nom', 'Dupond'
  put 'personne', '2860738256345', 'contact:prenom', 'Philoe'
  put 'personne', '2860738256345', 'contact:prenom', 'Zoe'
  put 'personne', '2860738256345', 'contact:prenom', 'Arsinoe'
  put 'personne', '2860738256345', 'contact:prenom', 'Aglae'
  put 'personne', '2860738256345', 'contact:prenom', 'Berthe'
  put 'personne', '2860738256345', 'contact:prenom', 'Agathe'
  get 'personne', '2860738256345'
  get 'personne', '2860738256345', 'contact:prenom'
  get 'personne', '2860738256345', {COLUMN => 'contact:prenom', TIMERANGE => [0,
      1516423672432]}
  -- timestamp de Agathe : 1442245190698
  get 'personne', '2860738256345', {COLUMN => 'contact:prenom', TIMERANGE => [0,
      1516423672432], VERSIONS => 4}
  get 'personne', '02', {COLUMN=>'contact:prenom',VERSIONS=>3}
  scan 'personne', {FILTER => "PrefixFilter('286')"}
  ou bien
  scan 'personne', STARTROW => '286', ENDROW => '287'
  delete 'personne', '176073424356', 'contact:nom'
  scan 'personne'
  disable 'personne'
  alter 'personne', 'delete' => 'contact'
  enable 'personne'
  scan 'personne'
  disable 'personne'
  drop 'personne'
  list
```

Listing 4 – ordres

3.3 Eléments de syntaxe pour répondre à la question MapReduce

```
public static class MyMapper extends TableMapper<Text, IntWritable> {
 public static final byte[] CF = "cf".getBytes();
 public static final byte[] ATTR1 = "attr1".getBytes();

 private final IntWritable ONE = new IntWritable(1);
 private Text text = new Text();
```

```
  public void map(ImmutableBytesWritable row, Result value, Context context) throws
      IOException, InterruptedException {
    String val = new String(value.getValue(CF, ATTR1));
    text.set(val); // we can only emit Writables...
    context.write(text, ONE);
  }
}

In the reducer, the "ones" are counted (just like any other MR example that does this),
    and then emits a Put.

public static class MyTableReducer extends TableReducer<Text, IntWritable,
    ImmutableBytesWritable> {
  public static final byte[] CF = "cf".getBytes();
  public static final byte[] COUNT = "count".getBytes();

  public void reduce(Text key, Iterable<IntWritable> values, Context context) throws
      IOException, InterruptedException {
    int i = 0;
    for (IntWritable val : values) {
      i += val.get();
    }
    Put put = new Put(Bytes.toBytes(key.toString()));
    put.add(CF, COUNT, Bytes.toBytes(i));

    context.write(null, put);
  }
}
```

Listing 5 – Map et Reduce