Corrigé de l'examen

***Module :* La Programmation Pour le Big Data**

Questions du cours : (7 points)

1. Les trois caractéristiques du Big Data sont le **Volume**, la **Vélocité** et la **Variété.** (1 point)

   Volume : le volume représente la quantité de données qui croît à une vitesse exponentielle, c'est-à-dire en pétaoctets et en exaoctets.

   Vélocité : La vélocité fait référence à la vitesse à laquelle les données augmentent. Par exemple, les médias sociaux contribuent grandement à la vitesse de croissance des données.

   Variété : la variété fait référence à l'hétérogénéité des types de données (vidéos, audios, csv, etc…). Ces différents formats représentent donc la variété des données.

2. C'est le Nœud Maître qui est chargé de stocker les métadonnées de tous les fichiers et répertoires. Il contient des informations sur les blocs, la création d'un fichier et l'emplacement de ces blocs dans le cluster. (1 point)

3. Lorsqu'un client Hadoop soumet un job pour exécution : (1 point)

   - Le JobTracker demande au NameNode où se trouvent les blocs correspondants aux noms de fichiers donné par le client.
   - Le job tracker détermine quels sont les noeuds les plus appropriés pour exécuter les traitements en tenant compte de leur Co-localisation ou proximité.
   - Le Job Tracker envoie au Task Tracker sélectionné le travail à effectuer.
   - Exécution des tâches Map et Reduce.

4. Les *DataNodes* et les *Task Trakers* envoient des signaux de pulsation au *NameNode* et au *Job Tracker* pour les informer de leur existence et de leur nombre de slots disponibles. Si le signal n'est pas reçu, cela indiquerait des problèmes avec le nœud ou le Task Trakers. (1 point)

5. Différences entre *MapReduce V1 (MRv1)* et *MapReduce V2 (MRv2)* : (1 point)

   - Dans *MRv1,* Map Reduce est responsable à la fois du traitement et de la gestion des clusters, tandis que dans *MRv2*, le traitement est pris en charge par d'autres modèles de traitement et YARN est responsable de la gestion des clusters.
   - *MRv2* est plus performant que *MRv1* avec près de 10 000 nœuds par cluster, contrairement à *MRv1* qui est limité à 4 000.
   - Dans *MRv1,* le NameNode est seul point d'échec (Problème de disponibilité -*SPoF*-). Cependant, dans le cas de *MRv2,* chaque fois qu'un NameNode échoue, il est configuré pour la récupération automatique.
   - *MRv1* fonctionne sur le concept des slots alors que *MRv2* travaille sur le concept des conteneurs et peut également exécuter des tâches génériques.

6. Cet Objet permet au mapper et reducer d'interagir avec le reste du système Hadoop. Il inclut des données de configuration et des interfaces pour que le Job émettre des résultats. (1 point)

7. La typologie exhortant à l'utilisation de HBase est celle des bases de données NoSQL orientées colonnes. (1 point)

Exercice 1 : (5 points)

La bonne réponse est : la proposition b) (2 points)

Argumentation de la réponse : Etant donné que la taille par défaut des blocs HDFS est fixée à 12 octets, ainsi le nombre de blocs obtenus à partir des fichiers *Prix1.txt* et *Prix2.txt* sera égale à 3.

Aussi, en se basant sur le code source de la classe *mapper* (fourni à l'examen), on s'aperçoit clairement que chaque ligne d'un bloc coïncidera soit au prix maximum ou minimum, étant donné que le couple clé/valeur d'entrée de notre mapper correspond à (LongWritable **Key**, Text **Value**). Ainsi, la valeur finale de la variable maxDiff correspondant aux trois (03) blocs est respectivement **31.00**, **33.00** et **00.00**

Le cluster Hadoop a pu exécuter les trois instances de mapper vu qu'il permet d'en exécuter jusqu'à 4 en parallèle. Voilà pourquoi nous avons obtenue trois fichiers distincts. (3 points)

Exercice 2 : (8 points)

StatistiquesMapper.java

```
package Examen;

import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.*;

public class StatistiquesMapper extends MapReduceBase implements Mapper <LongWritable, Text, Text,
IntWritable> {
        private final static IntWritable one = new IntWritable(1);

        public void map(LongWritable key, Text value, OutputCollector <Text, IntWritable> output,
Reporter reporter) throws IOException {

                String valueString = value.toString();
                String[] InfoPaysUnique = valueString.split(",");
                output.collect(new Text(InfoPaysUnique [7]), one);
        }
}
```

StatistiquesReducer.java

```
package Examen;

import java.io.IOException;
import java.util.*;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.*;

public class StatistiquesReducer extends MapReduceBase implements Reducer<Text, IntWritable, Text,
IntWritable> {

        public void reduce(Text key, Iterator<IntWritable> values, OutputCollector<Text,IntWritable>
output, Reporter reporter) throws IOException {
```

```
                Text cle = key;
                int frequencePays = 0;
                while (values.hasNext()) {
                        IntWritable value = (IntWritable) values.next();
                        frequencePays += value.get();
                }
                output.collect(cle, new IntWritable(frequencePays));
        }
}
```

StatistiquesJobs.java

```
package Examen;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;

public class StatistiquesJobs {
        public static void main(String[] args) {
                JobClient Client_Vente = new JobClient();
                JobConf job_conf = new JobConf(StatistiquesJobs.class);
                job_conf.setJobName("Ventes_Pays");
                job_conf.setOutputKeyClass(Text.class);
                job_conf.setOutputValueClass(IntWritable.class);
                job_conf.setMapperClass(StatistiquesMapper.class);
                job_conf.setReducerClass(StatistiquesReducer.class);
                job_conf.setInputFormat(TextInputFormat.class);
                job_conf.setOutputFormat(TextOutputFormat.class);

                /* Pour information :
                arg[0] = Nom du dossier input enregistré dans HDFS
                arg[1] =  Nom du dossier output
                */
                FileInputFormat.setInputPaths(job_conf, new Path(args[0]));
                FileOutputFormat.setOutputPath(job_conf, new Path(args[1]));

                Client_Vente.setConf(job_conf);
                try {
                        JobClient.runJob(job_conf);
                } catch (Exception e) {
                        e.printStackTrace();
                }
        }
}
```