Université Mohamed Boudiaf M'sila **M1 Intelligence Artificielle**
Faculté MI, Dépt Informatique **M'sila, Le 12/02/2022**

Examen Final

Big Data et Science de Données

Exercice 1 (4 points)

Considérons un dossier HDFS nommé *patchesFolder* qui contient deux fichiers : *patches2018.txt* de taille 513MB et *patches2019.txt* de taille 515MB. Supposons que le cluster Hadoop utilisé peut supporter jusqu'à 5 instances mapper en parallèle. Le nombre d'instances reducer est 2 et la taille d'un bloc HDFS est 512MB. Quel est le nombre d'instances du mapper utilisées lorsqu'on exécute une application MapReduce sur les deux fichiers du dossier *patchesFolder* ?

Exercice 2 (4 points)

Considérons un dossier HDFS nommé *inputData* qui contient les fichiers suivants :

Nom du fichier	Taille	Contenu du fichier
Temperature1.txt	61 Octets	2016/01/01,00:00,0 2016/01/01,00:05,-1 2016/01/01,00:10,-1.2
Temperature2.txt	63 Octets	2016/01/01,00:15,-1.5 2016/01/01,00:20,0 2016/01/01,00:25,-0.5
Temperature3.txt	62 Octets	2016/01/01,00:30,-0.5 2016/01/01,00:35,1 2016/01/01,00:40,1.5

Supposons que le cluster Hadoop utilisé peut supporter jusqu'à 10 instances mapper en parallèle. Soit le programme MapReduce suivant exécuté sur les fichiers ci-dessus. Donner le résultat après exécution.

/* Mapper */

```
class MapperBigData extends Mapper<LongWritable, Text, Text, DoubleWritable> {
        protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
                String fields[] = value.toString().split(",");
                String date = fields[0];
                Double temperature = Double.parseDouble(fields[2]);
                // Emit (date, temperature)
                context.write(new Text(date), new DoubleWritable(temperature));
        }
}
```

/* Reducer */

```
class ReducerBigData extends Reducer<Text, DoubleWritable, Text, DoubleWritable> {
        @Override
        protected void reduce(Text key, // Input key type
                Iterable<DoubleWritable> values, // Input value type
                Context context) throws IOException, InterruptedException {
        double maxTemp = Double.MIN_VALUE;
        // Iterate over the set of values and compute the maximum temperature
        for (DoubleWritable temperature : values) {
                if (temperature.get() > maxTemp) {
                        maxTemp = temperature.get();
                }
        }
        // Emit (date, maximum temperature)
        context.write(key, new DoubleWritable(maxTemp));
        }
}
```

Exercice 3 (4 points)

Considérons un dossier HDFS *InputLog* qui contient deux fichiers : *log1.txt* de taille 1024MB et *log2.txt* de taille 256MB. Supposons que le cluster Hadoop utilisé peut supporter jusqu'à 5 instances mapper en parallèle. Quelle est la taille d'un bloc HDFS si on veut forcer l'application MapReduce pour exécuter 5 instances du mapper en parallèle sur les fichiers du dossier *InputLog* ?

Exercice 4 (8 points)

On veut analyser l'atmosphère d'une région. Pour cela on installe des capteurs pour récupérer la valeur du PM10 (Particulate Matter) dans l'air. Ces valeurs sont stockées dans un fichier où chaque ligne a le format suivant :

*sensorId, date, PM10 value (µg/m3)***\n**

Ecrire les fonctions Map et Reduce dans l'objectif de déterminer pour chaque capteur la moyenne du PM10. La figure ci-dessous donne un exemple des données en entrée et du résultat attendu après exécution du programme MapReduce.

- Input file

```
s1,2016-01-01,20.5
s2,2016-01-01,30.1
s1,2016-01-02,60.2
s2,2016-01-02,20.4
s1,2016-01-03,55.5
s2,2016-01-03,52.5
```

- Output pairs (s1, 45.4)
  (s2, 34.3)

BONNE REUSSITE
Dr. Tahar Mehenni

Université Mohamed Boudiaf M'sila **M1 Intelligence Artificielle**
Faculté MI, Dépt Informatique **M'sila, Le 08/03/2021**
Durée : 60 Minutes

Correction de l'Examen Final

Big Data et Science de Données

Exercice 1 (4 points)

Dossier *patchesFolder :*

Fichier1 : *patches2018.txt*, taille= 513MB
Fichier2 : *patches2019.txt*, taille= 515MB

Nombre maximum d'instances Mapper : 5
Bloc HDFS = 512MB.
Fichier1 : on utilise 2 blocs
Fichier2 : on utilise 2 blocs

Le nombre d'instances Mapper utilisées lorsqu'on exécute une application MapReduce sur les deux fichiers du dossier *patchesFolder* doit supporter les quatres blocs des données en entrée. **On utilise donc 04 instances de Mapper.**

Exercice 2 (4 points)

Dossier InputData :

Nom du fichier	Taille	Contenu du fichier
Temperature1.txt	61 Octets	2016/01/01,00:00,0 2016/01/01,00:05,-1 2016/01/01,00:10,-1.2
Temperature2.txt	63 Octets	2016/01/01,00:15,-1.5 2016/01/01,00:20,0 2016/01/01,00:25,-0.5
Temperature3.txt	62 Octets	2016/01/01,00:30,-0.5 2016/01/01,00:35,1 2016/01/01,00:40,1.5

Nombre maximum d'instances Mapper : 10
La fonction Mapper détermine les couples (date, temperature) pour chaque fichier
La fonction Reduce détermine la valeur maximale de la température *maxTemp pour* chaque date
Le résultat du programme MapReduce est le couple (2016/01/01, 1.5)

Exercice 3 (4 points)

Dossier *InputLog :*

Fichier1 : log1*.txt*, taille= 1024MB
Fichier2 : log2*.txt*, taille= 256MB

Nombre maximum d'instances Mapper : 5
Si on veut forcer l'application MapReduce pour exécuter 5 instances du Mapper, on doit avoir 5 blocs de données en entrée. La taille totale des deux fichiers est : 1024MB+256MB=1280MB, si on divise la taille totale des données en entrée sur le nombre prévu de blocs, on aura : 1280/5=256MB, **d'où la taille d'un bloc HDFS pour cette application est égale à 256MB**

Exercice 4 (08 points)

```
/* Mapper */
class MapperBigData extends
        Mapper<LongWritable, // Input key type
                Text, // Input value type
                Text, // Output key type
                FloatWritable> {// Output value type
        protected void map(LongWritable key, // Input key type
                Text value, // Input value type
                Context context) throws IOException, InterruptedException {

                // Split each record by using the field separator
                // fields[0]= first attribute - sensor id
                // fields[1]= second attribute - date
                // fields[2]= third attribute - PM10 value
                String[] fields = value.toString().split(",");
                String sensorId = fields[0];
                float PM10value = Float.parseFloat(fields[2]);

                // emit the pair (sensor_id, reading value)
                context.write(new Text(sensorId), new FloatWritable(new Float(PM10value)));
        }
}

/* Reducer */
class ReducerBigData extends
        Reducer<Text, // Input key type
                FloatWritable, // Input value type
                Text, // Output key type
                FloatWritable> { // Output value type

        @Override
        protected void reduce(Text key, // Input key type
                Iterable<FloatWritable> values, // Input value type
                Context context) throws IOException, InterruptedException {

                int count = 0;
                double sum = 0;

                // Iterate over the set of values and sum them.
                // Count also the number of values
                for (FloatWritable value : values) {
                        sum = sum + value.get();
                        count = count + 1;
                }

                // Compute and emit average value
                // Emits pair (sensor_id, average)
                context.write(new Text(key), new FloatWritable((float) sum / count));
        }
}
```

Rédigé par T. Mehenni