TP_6_ MAPREDUCE_KMeans

Module : Big data

Filière : II-BDCC 2

Nom : KAFANDO Tounwendsida Bertrand

Première Implémentation

La première est de faire le clustering des points, chaque point est caractérisé par x et y.

Création de la classe Mapper :

Il est composé d'une fonction setup() qui charge les centroïdes à chaque étape à partir du caches.

```
protected void setup(Mapper<LongWritable, Text, Text ,Text>.Context context) throws IOException, InterruptedException {
    //centers.clear();
    centres.clear();
    URI[] uri= context.getCacheFiles();
    FileSystem fs=FileSystem.get(context.getConfiguration());
    //InputStreamReader is=new InputStreamReader(fs.open(new Path(uri[0])));
    BufferedReader reader=new BufferedReader(new InputStreamReader(fs.open(new Path(uri[0]))));
    String ligne="";
    while((ligne=reader.readLine())!=null){
        String tab[]=ligne.split( s: ",");
       // double centre=Math.sqrt( Math.pow(Double.parseDouble(tab[0]),2)+Math.pow(Double.parseDouble(tab[1]),2));
        Centroid centroid=new Centroid( centre: 0,Double.parseDouble(tab[0]),Double.parseDouble(tab[1]));
        centres.add(centroid);
    }
}
```

Ensuite viens la fonction map<> qui reçoit les points à classer ligne par ligne. Je split chaque point et grâce à une boucle je calcule la distance entre le point et chaque centroïde et j'enregistre le centre le plus proche. J'envoie comme

sortie pour l'opération shuffle le centre le plus proche sous forme de point et le point concerné.

```
@Override
protected void map(LongWritable key, Text value, Mapper<LongWritable, Text, Text, Text>.Context context) throws IOException, InterruptedException {

  //  double p=Double.parseDouble(value.toString());
    double min=Double.MAX_VALUE,d;
          Centroid  nearest_center=new Centroid();
    String tab[]=value.toString().split( s: ",");

    Point point=new Point(Double.parseDouble(tab[0]),Double.parseDouble(tab[1]));

    for (Centroid c:centres
         ) {
        d=Math.sqrt(Math.pow(point.getX()-c.getX(),2)+Math.pow(point.getY()-c.getY(),2));
        if (d<min){
            min=d;
            nearest_center=c;
        }
    }
    nearest_center.getPoints().add(point);   You, 07/04/2022 02:44 • Uncommitted changes
     Point centre=new Point(nearest_center.getX(),nearest_center.getY());

    context.write(new Text(centre.toString()),value);

}
```

Création de la classe Reducer:

Il reçoit en entrer le centroïde sous forme de Text en clé et la liste des points proches de ce dernier.

Je calcule la somme des x et y de ses points desquelles j'obtiens la moyenne qui servira de centroïde à la prochaine itération.

Ensuite j'écris en sortie cette moyenne et l'ancien centroïde.

```
@Override
protected void reduce(Text key, Iterable<Text> values, Reducer<Text, Text, Text, Text>.Context context) throws IOException, InterruptedException {
   double sommeX=0,sommeY=0,meanX,meanY;
   int nb_points=0;
    Iterator<Text> it=values.iterator();
    while (it.hasNext()){
        String pts[]=it.next().toString().split( s: ",");
        sommeX+=Double.parseDouble(pts[0]);
        sommeY+=Double.parseDouble(pts[1]);
        nb_points++;
    }
    meanX=sommeX/nb_points; meanY=sommeY/nb_points;
    context.write(key,new Text(new Point(meanX,meanY).toString()));
}
}
```

Création de la classe KmeansDriver:

On crée la configuration initiale pour un job normale ; c'est-à-dire spécifier la classe Jar , la classe Mapper et la classe Reducer ; les types de sorties outputclass et outputvalue

Mais nous devons répéter cette opération jusqu'à un certain nombre d'itérations ou que les centroïdes ne changent plus. Ici je déclenche je boucle infinie et à job je récupère les nouvelles données que je compare aux anciens centroïdes. S'ils sont égaux j'arrête la boucle sinon je l'enregistre dans le cache et je recommence le processus.

```
int iteration=0;
Path file=new Path( pathString: "/input/datav2.txt");

while (true) {
        You, 38 minutes ago • Uncommitted changes
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf,  jobName: "Kmeans job");


    job.setJarByClass(KmeansDriver.class);
    job.setMapperClass(KmeansMapper.class);
    job.setReducerClass(KmeansReducer.class);

    job.setMapOutputKeyClass(Text.class);
    job.setOutputValueClass(Text.class);

    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(Text.class);

    job.setInputFormatClass(TextInputFormat.class);
    job.setOutputFormatClass(TextOutputFormat.class);


    job.addCacheFile(new URI( s: "hdfs://localhost:9000/input/centerv2.txt"));

    FileInputFormat.addInputPath(job, file);
    FileOutputFormat.setOutputPath(job, new Path( pathString: "/output/oup"+iteration));

    job.waitForCompletion( verbose: true);
```

```java
FileSystem fs=FileSystem.get(conf);
// replace centroids with new centroids from last output file after the end of every job //
FSDataOutputStream out = fs.create(new Path( pathString: "hdfs://localhost:9000/input/centerv2.txt"), overwrite: true);
BufferedWriter bw = new BufferedWriter(new OutputStreamWriter(out));
// get new centroids from last output
InputStreamReader is = new InputStreamReader(fs.open(new Path( pathString: "hdfs://localhost:9000/output/oup" + iteration + "/part-r-00000")));
BufferedReader br = new BufferedReader(is);
String line = null;
StringBuilder old_centroid = new StringBuilder();
StringBuilder new_centroid = new StringBuilder();   KAFANDO, 07/04/2022 03:53 • kmean avec mapreduce
while ((line = br.readLine()) != null) {
    String part[]= line.split( s: "__");
    //new centroids
    new_centroid.append(part[1]);
    new_centroid.append("\n");
    //old centroids
    old_centroid.append(part[0]);
    old_centroid.append("\n");
}
// if old centroids == new centroids or iterations>=10 -> end while
//new_centroid.toString().equals(old_centroid.toString())
System.out.println(old_centroid.toString());
System.out.println(new_centroid.toString());
//don't forget to delete space
if((new_centroid.toString().replaceAll( s: "\\s+", s1: "")).equals((old_centroid.toString().replaceAll( s: "\\s+", s1: "")))|| iteration>=10){
    System.out.println("stop");
    break;
}
// save new centroids to centerMRI.txt
bw.write(new_centroid.toString());
bw.close();
br.close();
iteration++;
```

Test:

data sur hadoop

eansReducer.java | Point.java | Centroid.java | KmeansDriver.java | centerv2.txt | datav2.txt

```
12,20
45,29
67,54
98,3
56,89
45,66
53,43
15,45
87,23
65,15
45,18
67,98   KAFANDO, 07/04/2022 03:53 • kmean avec mapreduce
76,45
```

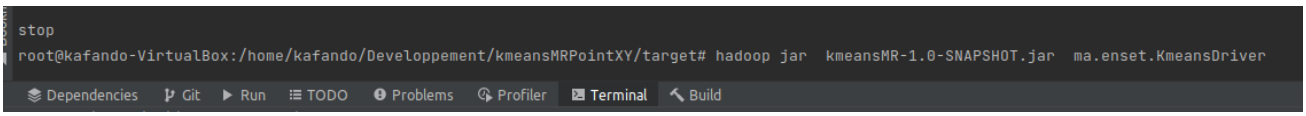
meansReducer.java | Point.java | Centroid.java | KmeansDriver.java | centerv2.txt

```
12,11
40,50   KAFANDO, 07/04/2022 03:53 • kmean avec mapreduce
90,20
```

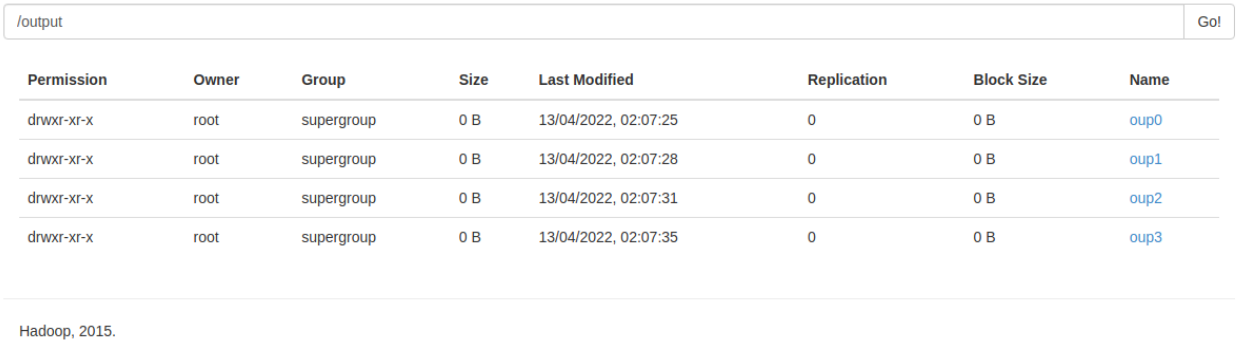
/input Go!

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	root	supergroup	8 B	28/03/2022, 15:57:27	1	128 MB	center.txt
-rw-r--r--	root	supergroup	0 B	13/04/2022, 02:07:35	1	128 MB	centerv2.txt
-rw-r--r--	root	supergroup	39 B	29/03/2022, 01:58:00	1	128 MB	data.txt
-rw-r--r--	root	supergroup	47 B	28/03/2022, 15:56:56	1	128 MB	data1.txt
-rw-r--r--	root	supergroup	77 B	07/04/2022, 02:20:24	1	128 MB	datav2.txt

Hadoop, 2015.

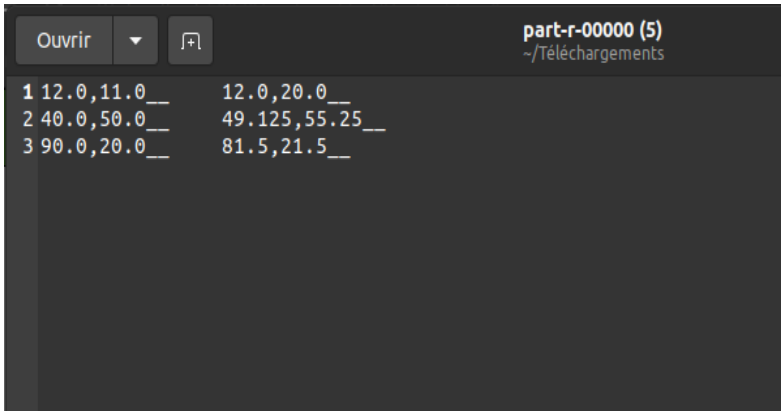


Les centroïdes restent fixent juste après 4 job

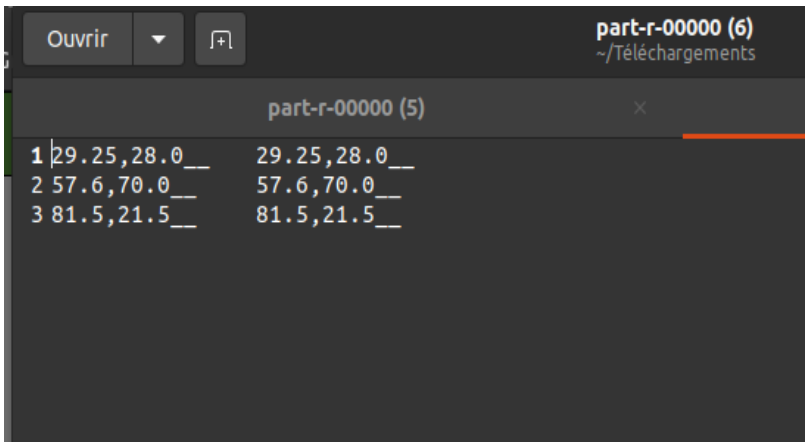


Résultat :

Oupo (12,12),(40,50),(90,20) sont les entrées



Les valeurs de sorties , on voit que les centroïdes sont égaux.



Deuxième Implémentation

La deuxième implémentation de kmeans et de faire le clustering d'une image IRM cérébrale (Figure 2). L'image est en niveau de grille (grayscale image) ou la valeur de chaque pixel est entre 0 et 255 (0 représente la couleur noire et 255 représente la couleur blanche), cette image montre trois parties du cerveau à savoir la matière blanche, la matière grise et le liquide céphalorachidien, l'objectif et de savoir les pixels de chaque partie.

Création de la classe Mapper :

Je récupère les valeurs des centres à partir du fichier cache

```
public class KmeansMapper extends Mapper<LongWritable, Text, Text,Text> {
    List<Double> centroide = new ArrayList<>();

    @Override
    protected void setup(Mapper<LongWritable, Text, Text, Text>.Context context) throws IOException, InterruptedException {
        centroide.clear();
        URI uri[]= context.getCacheFiles();
        FileSystem fs=FileSystem.get(context.getConfiguration());
        InputStreamReader is=new InputStreamReader(fs.open(new Path(uri[0])));
        BufferedReader br=new BufferedReader(is);
        String line=null;

        while ((line=br.readLine())!=null){
            centroide.add(Double.parseDouble(line));
        }
    }   KAFANDO, Today • KmeansIRM
```

Ensuite je recupère les données de l'image ligne par ligne et je cherche la distance minimale entre les centres et la valeur de la couleur du pixel. Je retourne ensuite le centre le plus proche et le pixel pour l'operation shuflle.

```
    @Override
    protected void map(LongWritable key, Text value, Mapper<LongWritable, Text, Text, Text>.Context context) throws IOException, InterruptedException {
        Pixel p = new Pixel();
        p.lineToPixel(value.toString());//split et add
        double min=Double.MAX_VALUE,d;
        double nearest_center = 0;
        for (double c:centroide) {
            d = Math.abs(c-p.getValue());
            if (d<min){
                min=d;
                nearest_center=c;
            }
        }
        context.write(new Text(String.valueOf(nearest_center)), value); // value (p.x,p.y,p.value)
    }
```

Création de la classe Reducer:

Après l'opération shuffle reducer reçoit une paire de key qui le point le plus près, value qui contient une liste des pixels proche de ce centre.

Je calcule la moyenne qui me servira de nouveau centroïde et je retourne l'ancien centroide key ,le nouveau mean et les pixels.

```
public class KmeansReducer extends Reducer<Text,Text,Text,Text> {
    @Override
    protected void reduce(Text key, Iterable<Text> values, Reducer<Text, Text, Text, Text>.Context context) throws IOException, InterruptedException {
        double somme = 0;
        StringBuilder pixels = new StringBuilder();
        int nb_points=0;
        Iterator<Text> it=values.iterator();
        while(it.hasNext()){
            String iterationVal = it.next().toString();
            pixels.append(iterationVal+"/"); //
            //
            Pixel p = new Pixel();
            p.lineToPixel(iterationVal);
            somme += p.getValue();

            nb_points++;
        }
        double mean= somme/nb_points;
        context.write(new Text( string: key+","+mean),new Text(pixels.toString())); // output will be ( old_centroid,new_centroide  pixels[] )        // pi
    }
}   KAFANDO, Today • KmeansIRM
```

Création de la classe KmeansDriver:

C'est le même principe que pour les points. Je transforme l'image en matrice de data à chaque début de mon job.

```
public class KmeansDriver {
    public static void main(String[] args) throws IOException, URISyntaxException, InterruptedException, ClassNotFoundException {

        Configuration conf = new Configuration();
        FileSystem fs = FileSystem.get(conf);

        int iteration=0;
        Path file=new Path( pathString: "hdfs://localhost:9000/input/image.txt");
        if ( fs.exists( file )) { fs.delete( file,  b: true ); }

        BufferedWriter br1 = new BufferedWriter( new OutputStreamWriter(fs.create(file)) );

        StringBuilder data = new StringBuilder();

        // get image from hdfs
        BufferedImage image = ImageIO.read(fs.open(new Path( pathString: "hdfs://localhost:9000/input/brain_mri.gif")));

        int w = image.getWidth();
        int h = image.getHeight();
        // image to data
        for(int i=0;i<w;i++){
            for(int j=0;j<h;j++) {
                int pixelVal = image.getRGB(i,j) & 0xFF; // 'getRGB(i,j) & 0xFF' normally returns blue color value
                if(pixelVal!=0)    // to ignore black pixels, so they won't affect the process
                    data.append(i+","+j+","+pixelVal+"\n");
            }
        }

        // save data to imageData.txt
        br1.write(data.toString());
        br1.close();
```

Recommended plugin available fo

Ensuite je déclenche une infinie qui s'arrêtera lorsque les centroides ne changeront plus ou lorque l'on atteint 20 iterations.

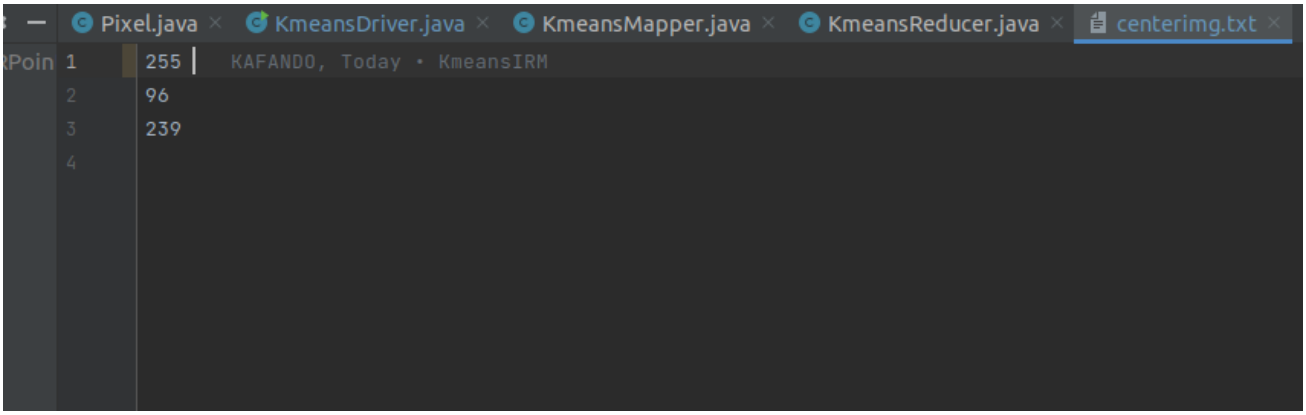
```
while (true) {
    Job job = Job.getInstance(conf, jobName: "Kmeans job");
    job.setJarByClass(KmeansDriver.class);
    job.setMapperClass(KmeansMapper.class);
    job.setReducerClass(KmeansReducer.class);
    job.setMapOutputKeyClass(Text.class);
    job.setOutputValueClass(Text.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(Text.class);
    job.setInputFormatClass(TextInputFormat.class);
    job.setOutputFormatClass(TextOutputFormat.class);
    job.addCacheFile(new URI( s: "hdfs://localhost:9000/input/centerimg.txt"));
    FileInputFormat.addInputPath(job, file);
    FileOutputFormat.setOutputPath(job, new Path( pathString: "/output/RMI"+iteration));
    job.waitForCompletion( verbose: true);
    // replace centroids with new centroids from last output file after the end of every job //
    FSDataOutputStream out = fs.create(new Path( pathString: "hdfs://localhost:9000/input/centerimg.txt"), overwrite: true);
    BufferedWriter bw = new BufferedWriter(new OutputStreamWriter(out));
    // get new centroids from last output
    InputStreamReader is = new InputStreamReader(fs.open(new Path( pathString: "hdfs://localhost:9000/output/RMI" + iteration + "/part-r-00000")));
    BufferedReader br = new BufferedReader(is);
    String line = null;
    StringBuilder old_centroid = new StringBuilder();
    StringBuilder new_centroid = new StringBuilder();
    while ((line = br.readLine()) != null) {...}
    // if old centroids == new centroids or iterations>=10 -> end while
    //new_centroid.toString().equals(old_centroid.toString())
    System.out.println(old_centroid.toString());
    System.out.println(new_centroid.toString());
    //don't forget to delete space
    // if old centroids == new centroids or iterations>=10 -> end while
    if(new_centroid.toString().equals(old_centroid.toString()) || iteration>=10){
        break;
    }
```

Lorsque la boucle s'arrête je récupère les centroïdes et les pixels et je refait les trois images.

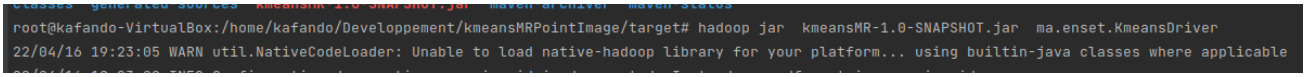
```
    }
    // create images "gray_matter.gif", "white_matter.gif" , "cephalo_rachidien.png" from obtained output
    InputStreamReader is = new InputStreamReader(fs.open(new Path( pathString: "hdfs://localhost:9000/output/RMI" + iteration + "/part-r-00000")));  // read l
    BufferedReader br = new BufferedReader(is);

    String line1 = null;
    ArrayList<Centroid> centroids = new ArrayList<>();
    while ((line1 = br.readLine()) != null) {...}

    String[] imagesPath = {"cephalo_rachidien.gif", "white_matter.gif", "gray_matter.gif"};

    // sort centroids array by center attribute
    centroids.sort(Comparator.comparingDouble(Centroid::getCenter));

    // create output images
    for(int j=0;j<3;j++){
        // instanciate black image
        BufferedImage bImage = new BufferedImage(w, h, BufferedImage.TYPE_INT_RGB);
        Centroid c = centroids.get(j);
        for(Pixel p:c.getPixels()){
            bImage.setRGB(p.getX(), p.getY(), image.getRGB(p.getX(), p.getY()));
        }

        ImageIO.write(bImage, s: "gif", fs.create(new Path( pathString: "hdfs://localhost:9000/outputs/"+imagesPath[j])));
    }



}

}
```

Test :

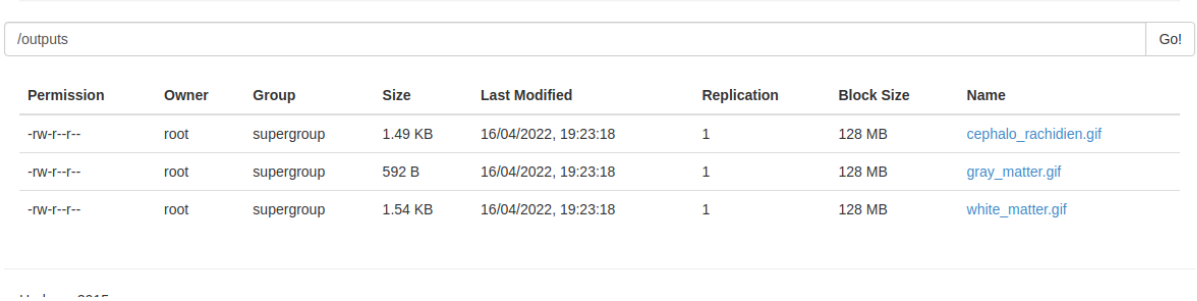
Les centroïdes initiales



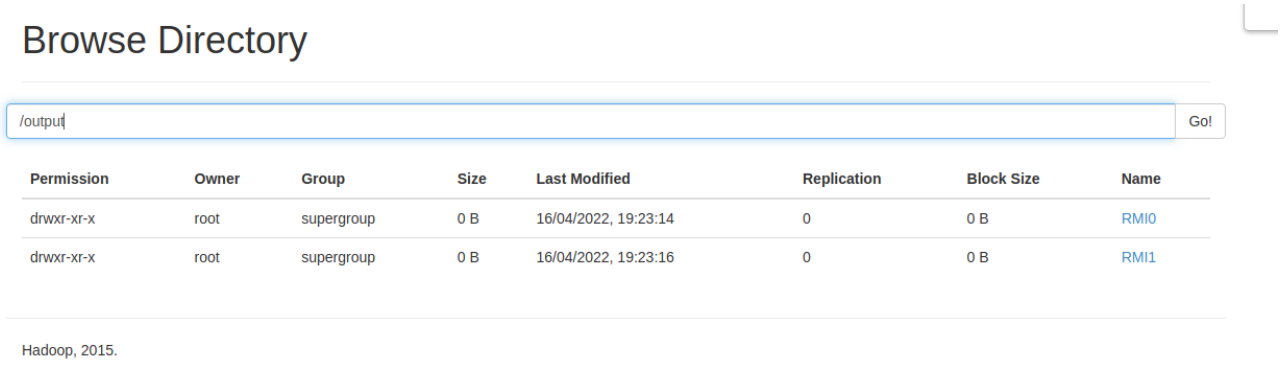
Excution sur hadoop

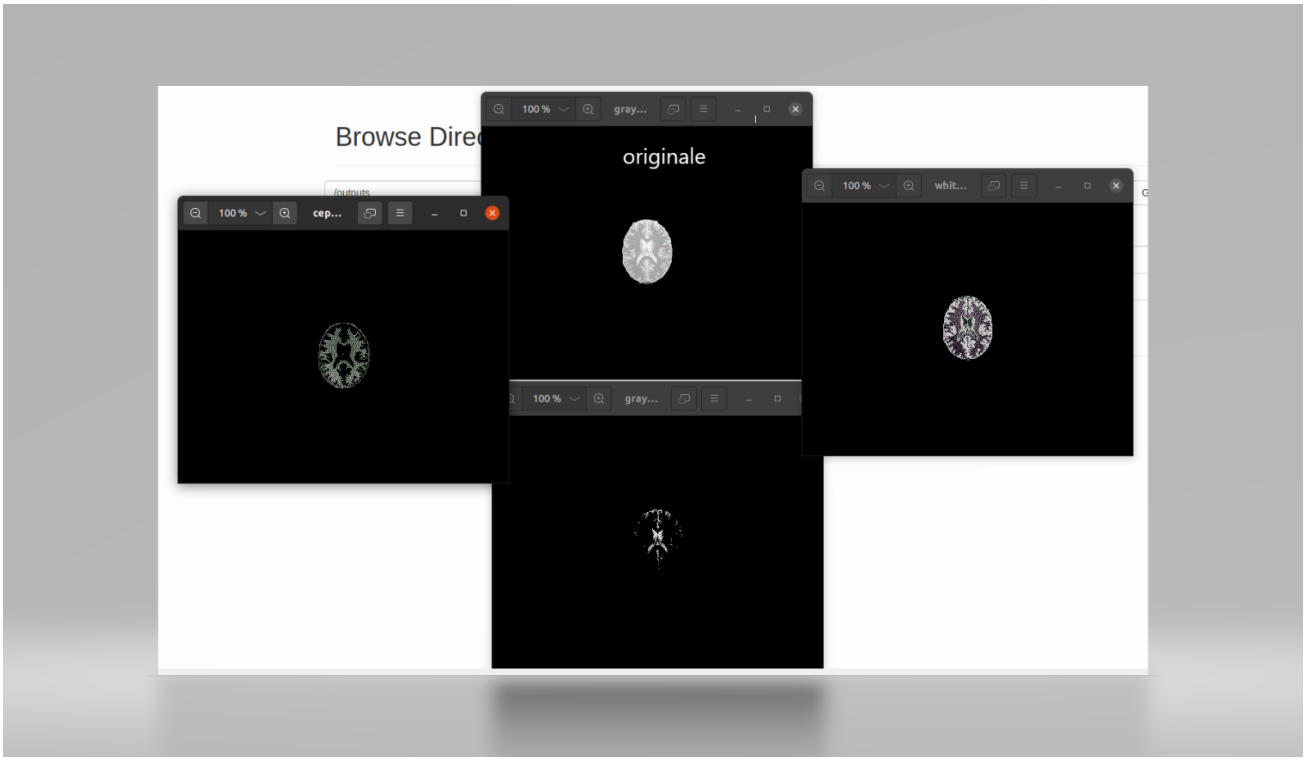


Sorties :



Arret après deux excutions, surement parce que j'ai des centres qui tendent vers les valeurs voulues.




Browse Dire
originale
100 %
gray...
100 %
cep...
100 %
whit...
100 %
gray...