

UNIVERSITÉ DE BORDEAUX

**Programmation large échelle
Projet**

**Visualisation des données sous
forme d'une carte de chaleur**

Par:

HOCINI Mohamed Fouad
MAURICE Bastien

Enseignant :
David Auber

February 2, 2018

1 Introduction

Le but de ce projet est de visualiser des données de géolocalisation de grande taille sous forme d'une carte de chaleur interactif avec un gradient de couleur qui représente les hauteurs/populations de chaque paire (latitude, longitude) présente dans le jeu de données après le traitement de ces données, les sauvegarder ainsi que les récupérer pour les afficher dans notre application web avec la possibilité de faire des différents niveaux de zoom sur la carte.

2 Partie Application donnée

2.1 Choix techniques

Pour le traitement de données, nous avons choisi d'utiliser Spark et non pas du MapReduce, vu les performances et la simplicité de Spark ainsi que la facilité de mettre en place HBASE pour le sauvegarde des données que nous avons traité avec Spark.

2.2 Traitement des données et enregistrement dans HBASE

Dans ce projet, nous avons travaillé sur les données des deux fichiers *dem3lat1ng.txt* de 212.7 GO de taille et le fichier *worldcitiespop.txt* qui contiennent respectivement des positions différentes (latitude et longitude) avec les hauteurs et (pays, ville, régions et la population) pour le deuxième.

Le traitement des fichiers de données a été mis dans deux projets : projet, projet2 pour le premier fichier de données et projet pour le deuxième.

Nous avons commencé par la création des différentes tables HBASE avec une vérification lors du lancement du programme pour vérifier si la table demandé existe déjà ou non, si oui nous la désactivons et nous la supprimons avant de procéder à sa création de nouveau.

Nous avons créé une classe WorldCitiesCordonnees pour définir une structure avec tout les champs que nous allons utilisé par la suite pour retourner les données demandés:

Listing 1: Structure à retourner

```
1 | WorldCitiesCordonnees(String pays, String ville, String  
|   region, double population, double latitude, double  
|   longitude)
```

Après cela, nous avons créé notre job Spark ainsi que la RDD avec comme entrée args[0] qui sera notre fichier de données, que nous avons utiliser un split(',') afin de séparer tout les lignes et avoir les différents champs:

- *worldcitiespop* : les champs de ce fichier (pays, region, ville, population, latitude et longitude) que nous enregistrons dans un put afin de l'ajouter dans notre Hbase après avoir passer par un filtre de notre rdd pour éliminer

les valeurs incorrectes par exemple le cas où la latitude et/ou la longitude dépasse les bornes des intervalles ainsi que les pays, régions ,villes ou le champs population est vide.

Listing 2: Filtrage de la rdd

```

1 | JavaRDD<WorldCitiesCordonnees> rddFiltred = data.
|     filter((x) -> x.population != -1 && x.latitude <
|             90 && x.latitude > -90 && x.longitude < 180 && x.
|             longitude > -180);

```

Nous avons définie trois niveaux de zoom pour ce fichier: par Pays, par région ou par ville, et à chaque fois de ces niveau, nous avons fait la somme de la population de chaque critère de zoom ainsi que la moyenne de la latitude longitude de ce dernier afin d'avoir une ligne à écrire sur Hbase à chaque fois.

- *dem3* : les trois champs de ce fichier (*latitude, longitude, hauteur*) que nous les avons rendu sous forme décimal, après cela nous avons filtrer toutes les lignes invalide là où il manque un des trois choix ou qui contient une valeur de position (latitude, longitude) qui dépasse les bornes de l'intervalle.

Puisque ce fichier est trop grand et afin de pouvoir le traiter, nous avons créer des intervalles avec des tailles différentes tout dépend du niveau de zoom que nous allons utiliser, et pour chaque intervalle, nous créons une RDD avec un pas qui correspond à la taille de notre tuile qui dépend du niveau de zoom et comme valeur de hauteur la somme de toutes les valeurs qui sont dans cette intervalle.

Listing 3: Filtrage et moyenne lat, log

```

1 | double latMin = -90; double latMax = 90;
2 | double longMin = -180; double longMax = 180;
3 |     //Niveau de zoom 1
4 |     double pasLat= 1.40625;
5 |     double pasLong= 2.8125;
6 | JavaRDD<Dem3Cordonnees> rddIntervalle = rddFiltred
|         .filter((x) -> x.latitude < (indicei+
|             paslatitude1) && x.latitude > indicei && x.
|             longitude < indicej && x.longitude > indicej+
|             paslongitude1);
7 | JavaPairRDD<PointGPS, Double> avgposition =
|         rddIntervalle
8 |         .mapToPair(new PairFunction<Dem3Cordonnees,
|                         PointGPS, Double>() {
9 |             @Override
10 |             public Tuple2<PointGPS, Double> call(
|                 Dem3Cordonnees intervalle) throws
|                 Exception {

```

```

11 |         return new Tuple2<PointGPS, Double>(new
|             PointGPS((indicei+paslatitude1)/2,(indicej+paslongitude1)/2), intervalle.
|             hauteur);
12 |
13 |

```

Ce traitement va nous permettre d'utiliser des collect() pour créer une liste avec toutes nos valeurs que nous venons de calculer.

Par la suite nous avons ajouté toutes les valeurs (nouvelle latitude, nouvelle longitude, somme hauteur) dans un put afin de l'ajouter dans notre Hbase.

A la fin de cette étape, nous récupérons les données et nous mettons dans un Put pour les enregistrer dans notre table Hbase en précisant les champs Family et une numérotation des lignes sous forme de (ROW+i).

Listing 4: Remplissage de la table Hbase

```

1 for (int i = 0; i < mydata.size(); i++) {
2     Put put = new Put(Bytes.toBytes("ROW" + i));
3     put.addColumn(FAMILY, Bytes.toBytes("pays"), Bytes.
4         toBytes(mydata.get(i).pays));
5     put.addColumn(FAMILY, Bytes.toBytes("ville"), Bytes.
6         toBytes(mydata.get(i).ville));
7     put.addColumn(FAMILY, Bytes.toBytes("region"), Bytes.
8         toBytes(mydata.get(i).region));
9     put.addColumn(FAMILY, Bytes.toBytes("population"),
10        Bytes.toBytes(mydata.get(i).population));
11    put.addColumn(FAMILY, Bytes.toBytes("latitude"), Bytes.
12       toBytes(mydata.get(i).latitude));
13    put.addColumn(FAMILY, Bytes.toBytes("longitude"), Bytes
14       .toBytes(mydata.get(i).longitude));
15    table.put(put);
16 }

```

Après cela, on compile notre projet Maven pour générer le fichier jar, et dans le terminal à l'aide de la commande:

```
mohhocini@beetlejuice:~$ spark-submit --num-executors 10 --class bigdata.Projet
--jars $(find /home/hadoop/cluster2017/hbase/lib/ -name *.jar | tr '\n' ',')espaces/travail/M2/PLE/Projet/target/Projet-0.0.1.jar espaces/travail/M2/PLE/Projet
/target/Projet-0.0.1.jar /raw_data/worldcitiespop.txt
```

Nous avons pu voir le côté performance, en comparant nos programmes sur les deux fichiers worldCitiespop et *dem3lating*, vu la différence de taille entre les deux.

3 Partie Application web

3.1 Choix des technologies

Nous utiliserons Node pour le backend avec le module Express. C'est un choix intéressant car il nous offre une solution robuste et très facile de mettre en place.

Encore une fois pour la facilité de la gestion du zoom, et la facilité pour l'insertion de points directement via leur latitude/longitude, on utilisera GoogleMap avec leur API pour l'insertion et la modification de celle ci via des données. Ainsi, nous n'avons pas besoin de faire nous même nos propres projections mercator de la carte du globe, et de gérer nous même la conversion de latitude/longitude vers un plan 2D X,Y.

Pour l'affichage des points de couleurs sur la carte, nous utilisons l'ensemble des outils proposé par leur api. Ce n'est pas la solution qui offre les meilleures performances (lorsqu'ils y a affichage de beaucoup de points), mais celle ci nous permet de modifier facilement les différents gradients de couleurs, rayons et intensité de ceux-ci, selon le zoom choisit. Ainsi, nous aurons des tailles et couleurs de marqueurs (cercle dans notre cas, qui représenter soit une moyenne d'un pays, d'une région ou des villes en terme de population). Chacun d'entre eux représente une ligne venant de HBase.

3.2 Lancement du serveur

Nous avons un fichier js qui permet de créer le serveur express :

```
1 | var express = require('express');
2 | var app = express();
3 | var path = require('path');
4 |
5 | app.use(express.static(path.join(__dirname, 'public')));
6 |
7 | app.get('/', function(req,res){
8 |   res.sendFile(path.join(__dirname+'public/index.html'
|     ));
9 | });
10 |
11 | app.listen(8080);
```

Nous pourrons tester notre app sur la page : <http://localhost:8080/> avec la commande :

```
1 | node app.js
```

3.3 Affichage de la carte et navigation

On va créer une instance de GMap. Nous avons donc pas à nous occuper de la gestion de la navigation et du zoom, comme pour une map façon SVG.

Concernant le paramétrage, on aura la possibilité d'injecter tel ou tel fichier, entre dem3 et worldcity. De plus, nous aurons la possibilité de paramétrer l'affichage, tel que le choix de la taille et couleur des points, ou encore de reset la map.

```
1 | var map = new google.maps.Map(document.getElementById('map'), {
```

3.4 Lien entre web et donnée

Pour cela nous allons devoir utiliser des services http rpc, car le rest étant bloqué au cremi. On utilisera le module rpc client hbase disponible sur NPM.

```
1 | client = hbase({
2 |   zookeeperHosts: ['localhost'],
3 |   zookeeperRoot: '/hbase',
4 | });
```

Il va falloir ensuite se connecter sur la bonne table, et récupérer les data. Il va falloir la filtrer car il y a énormément d'attribut inutile.

```
1 |
2 | const scan1 = client.getScanner('worldCitiesCountry');
3 | var s = scan1.toArray((err, res) =>{
4 | })
```

On va ensuite former le fichier json qui sera envoyé au front. On va créer des n-uplets sous forme de tableau.

```
1 | var jsonData = {
2 |   pays: [{lat:3333, long:3333, pop:3333}, ...]
3 | },
```

Le côté front va récupérer le fichier JSON précédent via une requête http avec jquery, et le parser pour pouvoir récupérer les infos. Ainsi pour chaque ligne parsé du json, on pourra par la suite créer un cercle selon la lat/long/pop et l'insérer sur notre map. Les cercles ainsi créé auront entre eux une taille différente, selon la valeur min et max au seins de notre échantillon de donnée.



Zoom par pays



Zoom par région



Zoom par ville

4 Problème rencontré

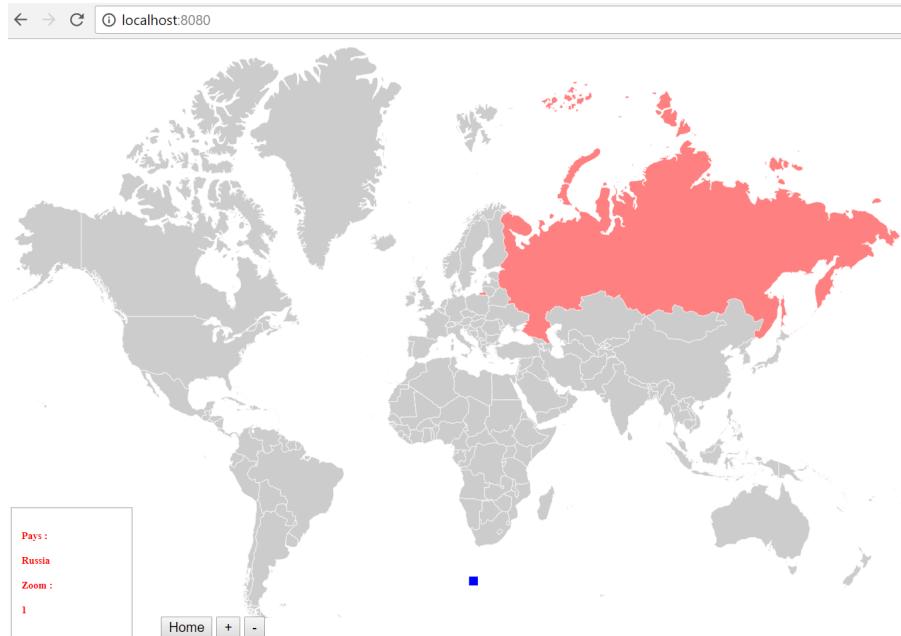
4.1 Traitement du fichier dem3

Nous avons rencontré beaucoup de problème pour lancer notre projet avec le fichier dem3.lat.lng de plus de 200 GO contrairement au fichier worldcitiespop, parce que lors du lancement il nous retourne une erreur comme quoi il n'y a pas assez de mémoire pour finir l'exécution.

Par la suite, nous avons rendu compte que nous faisons des .collect() sur des RDD de très grande taille et que c'était ça le problème, par la suite nous avons essayé de bien diviser nos données sur des petites tuiles pour pouvoir mettre des .collect() dedans afin de les mettre sur Hbase.

4.2 Choix de projection de carte

Nous étions partie à l'origine pour un simple affichage d'une carte mondiale sous forme de SVG. Nous avions alors la possibilité de sélectionner des pays, retrouver tel ou tel attribut de tel ou tel pays, de pouvoir le sélectionner et d'avoir 10 niveaux de zoom. Le soucis étant non pas pour la lecture de coordonnées, ou la façon de le dessiner sur notre SVG, mais nous avions des soucis pour convertir les latitudes et longitudes de chaque emplacement géographiques, vers un couple de coordonnée de plan (X,Y). En effet nous savons que les lat/long sont des ensemble entre -180/180 et -90/90. Et nous avons un repères sur notre carte allant de 0|x|taille map et 0|y|hauteur map. Cette carte du monde en 2D est une projection cartographique du globe. Mais il existe plusieurs types de projection, dont chacune ont des spécificités différentes concernant les angles et distances (mercator, etc..). On ne pourra donc avoir quelque chose de précis en faisant nous même nos propre estimation des dégrées. Nous étions ensuite partie pour utiliser une librairie appelé D3.JS pour la manipulation de carte svg et l'insertion de point, couplé à GeoJSON et TopoJSON pour la lecture des données. Mais ces librairies étant plutôt complexe à manipuler, on a finit par choisir une autre librairie plus légère et qui s'adapte avec l'api de GoogleMap. Mais au final, le rendu donnée par la librairie HeatMapJS n'était pas tellement la mieux adapté pour un tel type de projet et de placement de point. Après avoir testé 3 façons différentes de gérer la vue de la carte, on utilisera finalement GMap et son API.



Voici la première version de carte façon SVG, avec le zoom et sélection de pays fonctionnel. Mais on a du changer de méthode car la conversion de point lat/long en couple X,Y ne peut être fait de façon précise, Le point bleu devant représenter Paris après notre fonction de conversion...

Nous

5 Conclusion

Ce projet nous a permis de se familiariser avec Spark ainsi que Hbase, nous avons pu traiter les données des deux fichiers de données worldcitiespop et dem3, les filtrer et les enregistrer dans la base de données Hbase, mais par manque de temps nous n'avons pas pu finir avec le fichier dem3 pour le remplissage des différentes tables ainsi de les afficher ce qui ne change pas du tout de l'affichage de worldCities qui a été réalisé, et pour worldcities nous avons réalisé trois niveaux de zoom qui marchent et nous avons réussi aussi à afficher nos données de Hbase sur une carte avec la possibilité de changer de niveau de zoom: zoom0, zoom1 et zoom 2 et que nous avons utilisé des reducebykey pour calculer:

- Niveau de zoom 0: Calculer la moyenne de la population pour chaque pays.
- Niveau de zoom 1: Calculer la moyenne de la population pour chaque région.
- Niveau de zoom 2: Calculer la moyenne de la population pour chaque ville.

Concernant la partie web, nous savons nous connecter sur notre base HBase pour y récupérer nos données, et les envoyer au format JSON qui seront par la suite récupéré via des requêtes http depuis le front. Nous savons aussi comment afficher une carte et y placer des points selon des paramètres récupérés depuis l'interface graphique, pour y avoir un gradient de couleur correspondant à ces valeurs.