Spark et son écosystème pour le traitement de données LiDAR

Mattia Bunel
OVH, SV3D, DOT, IGN

29 juin 2023

Situation actuelle

Clichés ou bandes ?

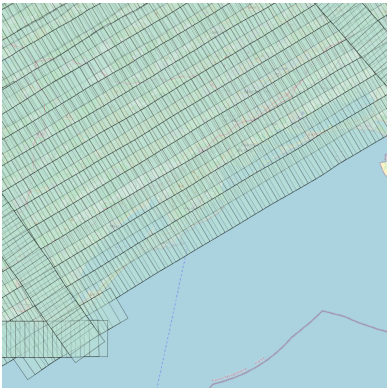

Figure 1 : Emprise des clichés pour la BD Ortho de l'Hérault

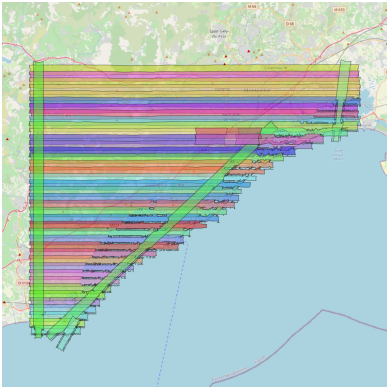

Figure 2 : Bandes de vol pour le bloc LiDAR HD MQ

Limites

Le traitement par lots a certaines limites :

- Sujet aux effets de bords
- Fausse simplicité

⇒ Une piste : changer de paradigme

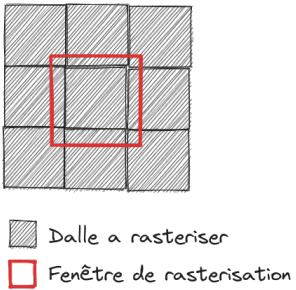

Figure 3 : Illustration du processus de rasterisation d'un ensemble de dalles LiDAR

Le paradigme *map-reduce*

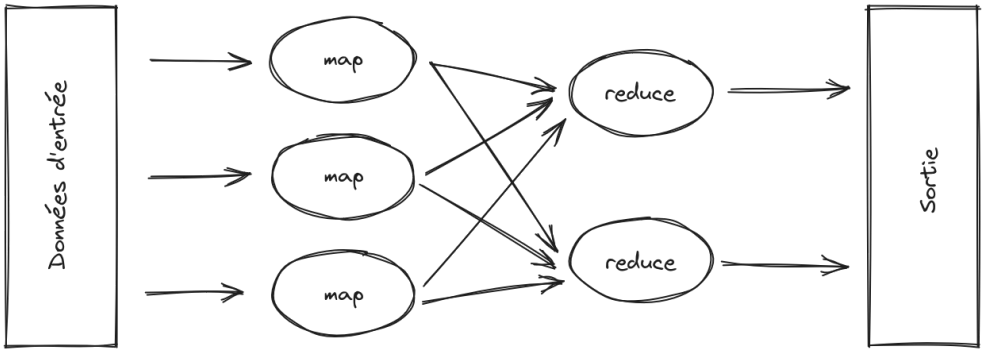

Figure 4 : Principde du paradigme *map reduce*[1]

1. D'après :
`https://algodaily.com/lessons/what-is-mapreduce-and-how-does-it-work`

Le « *Hello World* » du *map reduce*

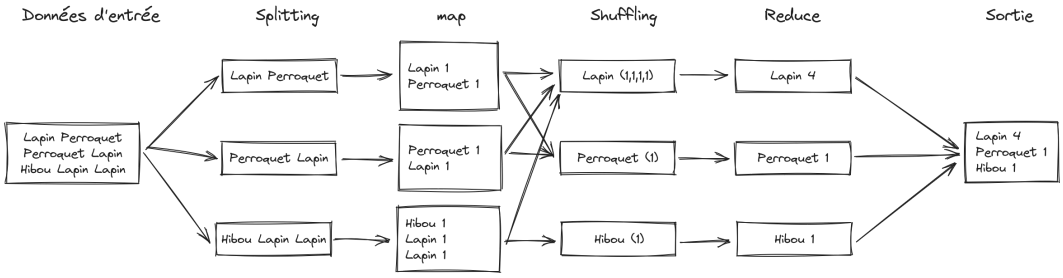

Figure 5 : Calcul du nombre d'occurrences d'un mot à l'aide du paradigme *map reduce* [2]

2. D'après :
`https://algodaily.com/lessons/what-is-mapreduce-and-how-does-it-work`

Qualités et défauts du *map reduce*

Avantages :

- Parallélisation transparente
- Scalabilité

Limites :

- Coût du *shuffling*
- Tous les algorithmes ne sont pas faciles à paralléliser (*e.g.* triangulation de Delaunay)

Apache Spark

Apache Spark : un framework pour le calcul distribué

Spark est un des *frameworks* implémentant le paradigme *map reduce*

`Caractéristiques principales:`

- Grammaire plus riche que le simple *map-reduce*
- Nombreuses API (Scala / Java, Python, R, SQL)
- Composants facultatifs : `MLib`, `Graph X`, *etc.*

Détails Techniques :

- Logiciel libre, porté par la fondation Apache
- Développé depuis 2014
- Écrit en Scala

Spark et l'écosystème Hadoop

Spark est fortement lié à l'écosystème Hadoop :

- Utilisation d'HDFS comme système de fichier
- IO natif pour les formats Apache Parquet, Apache Avro, Apache ORC, *etc.*
- Fort lien avec d'autres projets de la fondation apache : HBase, Airflow, *etc.*

Spark : Hello World

```
// Import d'un fichier texte (stocké en HDFS)
val textFile = sc.textFile("hdfs://input.txt")

// Map & reduce
val counts = textFile.flatMap(line => line.split(" "))
                 .map(word => (word, 1))
                 .reduceByKey(_ + _)

// Enregistrement du résultat dans un fichier (HDFS)
counts.saveAsTextFile("hdfs://output.txt")
```

Apache Sedona

« Spatial is Special »

Par défaut Spark ne permet pas la gestion de données spatiales :

- Pas de types géométriques (`POINT, LINE, MULTIPOLYGON,` *etc.*)
- Pas de fonctions spatiales (`ST_DISTANCE, ST_ConvexHull,` *etc.*)
- Pas d'indexation

Sedona

Sedona[3] est une extension de **Spark** gérant les données spatiales 

Sedona ajoute à **Spark** :

- Les types du modèle *simples features* (OGC)
- Les prédicats SQL du même modèle
- L'IO de données spatiales (`.shp`, `.tiff`, `.geoparquet`, *etc.*)
- **L'indexation** et la **répartition** spatiale

Détails techniques :

- Projet libre, incubé par la fondation Apache
- Développé depuis 2016
- Java et Scala
- Basé sur la **JTS** et **Geotools**

3. Auparavant *GeoSpark*.

Répartition spatiale de la donnée

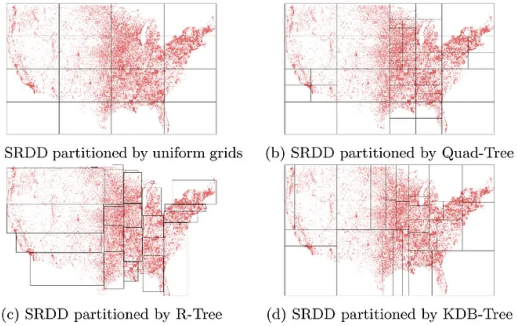

Figure 6 : dsq [4]

4. Extrait de `https://medium.com/towards-data-science/geospark-stands-out-for-processing-geospatial-data-at-scale-548077270ec0`

Sedona : *Hello world*

```
// Import d'un fichier parquet
val parquetFileDF = spark.read.parquet("input.parquet")

// Définition d'une table SQL associée au Dataframe
parquetFileDF.createTempView("lidar")

// Transformation en une table spatiale
val df = spark.sql("""
    SELECT ST_Point(X, Y, Z) as geom
    FROM lidar;
    """)

// Écrite en geoparquet
df.write.format("geoparquet").save("output.parquet")
```

Sedona : K plus proches voisins

```
var sRdd = Adapter.toSpatialRdd(df, "point")

// Indexation et partitionnement
sRdd.spatialPartitioning(GridType.KDBTREE)
sRdd.buildIndex(IndexType.RTREE, true)

// Définition d'une géométrie
val geometryFactory = new GeometryFactory()
val pointObject = geometryFactory.createPoint(
    new Coordinate(-84.01, 34.01)
)

// Calcul des plus proches voisins
val result = KNNQuery.SpatialKnnQuery(sRdd, pointObject, 20, false)
```

Sedona et les nuages de points

Sedona est mal adaptée au LiDAR :

- Géométries en 2,5D
- Pas d'indexation 3D
- Pas d'IO pour les formats nuage de points

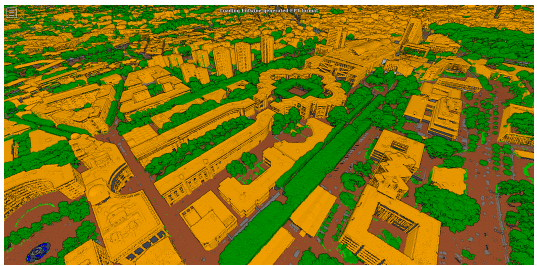

Figure 7 : Données LiDAR HD classées (Montpellier)

⇒ Utile pour les données vectorielles, moins pour les nuages de points

Contourner le problème ?

- D'autres approches existent, mais peu actives (Spark-3D)
- Communauté LiDAR de plus en plus dynamique (Hobu inc., Lutra consulting), mais pas sur ces questions
- L'avenir : Sedona avec un backend CGAL ?

Conclusion

- Paradigme *map-reduce* adapté aux données LiDAR
- Pas bibliothèques adaptées
- Piste de recherche et de développement importante

Merci de votre attention